Master *Radiation and its Effects on MicroElectronics and Photonics Technologies* (RADMEP)

PHYSICAL IMPLEMENTATION WORKLOAD EFFICIENCY IMPROVEMENT

Master Thesis Report

Presented by

Tarique Hasan

and defended at

University Jean Monnet

9th September 2024

Academic Supervisor(s): Prof. Dr. Sylvain Girard, University Jean Monnet

Host Supervisor: Julien Chevalier, Director of Design Enablement, Arm, France

Jury Committee:

Dr. Arto Javanainen, University of Jyväskylä

Prof. Sylvain Girard, University Jean Monnet

Prof. Dr. Ir. Paul Leroux, Katholieke Universiteit Leuven

Prof. Frédéric Saigné, University of Montpellier

PHYSICAL IMPLEMENTATION WORKLOAD EFFICIENCY IMPROVEMENT

Declaration

I, hereby, declare that this thesis entitled, “Physical Implementation Workload Efficiency Improvement” submitted to University of Jean Monnet, France, is the outcome of the research work performed by me, under the research collaboration among my supervisor, Dr. Sylvain Girard, University Jean Monnet, and Julien Chevalier, Director of Design Enablement, ARM, France.

I hereby declare that except where specific reference is made to the work of others, the contents of this thesis are original and have not been submitted in whole or in part for consideration for any other degree or qualification in this, or any other university. This dissertation is my own work and contains nothing which is the outcome of work done in collaboration with others, except as specified in the text and acknowledgments.

Author: Tarique Hasan

MSc. in RADMEP

Session: 2022-24

EMJMD RADMEP

University of Jyväskylä, KU Leuven, Jean Monnet University

Current academic email: tarique.hasan@etu.univ-st-etienne.fr

Personal email: tarique.eee.du@gmail.com

Thesis organization

The present thesis consists of 6 chapters. Chapter 1 is an introduction to the research topic and the importance and motivation for the work. Chapter 2 introduces the host organization's work, principles, and contributions to the field. Chapter 3, titled RTL-GDSII Implementation, briefly introduces post-RTL steps to tapeout. Chapter 4 describes cloud computing, models, and load management systems for EDA on the cloud. Chapter 5 outlines the working procedures, experiment constraints, tools, and software limitations during the internship. Chapter 6 is comprised of results and a discussion of the characterizations and data evaluation. In chapter 7, the major findings of this thesis are highlighted, along with potential directions for future research.

Abstract

EDA workloads on both cloud and on-premises require an understanding of the jobs on the host for the best turnaround time and cost efficiency. A lot of EDA jobs are launched with over demanded compute resources than necessary. They often fail if launched with resources less than what is required, thus increasing turnaround time. It is important to have prior knowledge of resource requirements according to design size and workload nature. This work explores synthesis, floorplan, clock tree synthesis, placement, and routing jobs for a small and bigger Arm CPU core design on compute clusters within the Arm's existing flow. The jobs dependency in terms of runtime and maximum memory utilization on multi-core jobs are investigated on different AMD and Intel machines on AWS cloud servers. It is found that a small design size does not benefit from parallelism. On the other hand, a bigger design has significantly reduced runtime for the implementation jobs when launched with multi-threaded CPUs. This work provides Arm with a method to extract information on EDA jobs with their flow and a schema that can be used for machine learning models to predict and build optimum job configuration.

Table of Contents

Table of figures

Acronyms

RTL: Register Transfer Level

GDSII: Graphical Design System

EDA: Electronic Design Automation

IP: Intellectual Property

HDL: Hardware Description Language

VHDL: VHSIC Hardware Description Language

FSM: Finite State Machine

LEC: Logic Equivalence Check

DFT: Design For Test

ATPG: Automatic Test Pattern Generation

BIST: Built In Self-Test

CTS: Clock Tree Synthesis

STA: Static Timing Analysis

LVS: Layout Versus Schematic

DRC: Design Rule Check

ERC: Electrical Rule Check

APS: Absolute Priority Scheduling

PnR: Placement and Routing

TAT: Turnaround time

Chapter 1

1.1 Introduction

The semiconductor industry has impacted all modern technologies directly or indirectly. The modern era is the era of computing and data. And that is possible due to the growth of computing chips. Today's chips contain billions of transistors, with continuous increase in number. As with the growth, the complexity of designing and manufacturing chips also increased. The chip design process is usually a sequence of standard steps of various simulations and analyses before moving into the fabrication or manufacturing stage. A digital chip design methodology can be divided into two steps: pre-RTL (Register Transfer Level) and implementation. Pre-RTL flow consists of several steps, from idea realization to functional implementation, using different design strategies. Considering the major steps, the implementation stage follows synthesis, physical implementation, timing, power analysis, and signoffs before converting the design to a layout or GDSII (Graphics Design System) file, which goes to foundries for chip fabrication. These steps require a lot of time, and repetitive performance, efficiency, and functionality testing is needed. A lot of computational, timing, and financial efforts are put into the implementation steps for productions [1] [2].

For a design with millions of gates, the testing, verification, and implementations can't be done manually; therefore, there is a need for EDA (Electronic Design Automation) tools. EDA tools automate these tasks and perform computations needed to generate desired data for the subsequent steps. Every design company uses these tools to automate their jobs for design productions. The computational tasks of these EDA tools require computing resources that don't come with a single hardware. So, the jobs are usually deployed on cloud or in-house compute farms [3]. For optimum and efficient production, it is imperative to understand the behavior of these jobs in compute clusters by characterizing EDA tools on these compute platforms. This will effectively reduce turnaround time to production and meet the market schedule [4] [5]. This motivates Arm, the world's leading processor IP (Intellectual Property) provider, to conduct the research through this master thesis project, "Physical Implementation Workload Efficiency Improvement." In this thesis work, I try to present the collection and evaluation of data of jobs launched for implementation flow steps with Arm internal flow.

1.2 Objective of the research

The objective of the research is to characterize different EDA tool jobs in terms of turnaround time and CPU time on computing machines with varying properties like the number of CPUs, memory, and machine type within Arm infrastructure. The data collected during this work will provide insights into the EDA tool's jobs deployment and performance and outline the best practices for production. The absolute data and the specific parameter information are all Arm internal and confidential. So, this publication will not publish the features of the EDA tools, host machine types, design information, and comparisons.

2 Chapter 2

2.1 Host organization: Arm

Arm is a leading semiconductor design and software manufacturing company. It pioneered high efficiency, low power, low-cost CPU, GPU, and NPU designs and technologies used by other semiconductor companies and original equipment manufacturers. Arm offers a wide range of processors to address the performance, power, and cost demands in mobile, IoT, laptop, and automotive markets. Arm was officially founded as a company in November 1990 as Advanced RISC Machines Ltd, as a joint venture between Acorn Computers, Apple Computer (now Apple Inc.), and VLSI Technology (now NXP Semiconductors N.V). Arm moved from single-product design to IP business in 1993 and continues to follow the same business model.

In the mobile phone era, Arm showed its true capabilities and potential, empowering more than 99% of smartphones today. More than 70% of the world's population uses Arm-based products. Around 280 billion Arm-based chips have been shipped to date, and 50% of all chips with processors use Arm products. With this massive success and contribution, Arm has expanded to meet the demands of the computing market. Its global headquarters are located in Cambridge, and 43 offices are spread across 21 countries in Asia, Europe, and the US. Arm is an international corporation, employing over 7,000 people worldwide, roughly 3,000 of whom work in the UK [6].

Arm, Sophia site is in the Sophia Antipolis Business Park in the South of France. It is a bilingual office of people from different nationalities and backgrounds. It is part of the worldwide collaboration network of Arm, consisting of more than 500 employees. The two main departments that operate at the Sophia site are CPU design and Physical IP design. Many teams are involved in various activities on-site or collaboratively with others internationally [7].

This thesis project's work was conducted under the Methodology and support team within Design Enablement. The Design Enablement team consists of more than 60 people from France, India, and the USA who work on various projects.

3 Chapter 3

3.1 RTL to GDSII Implementation

RTL-GDSII implementation refers to transforming a design (front-end) into manufacturable geometries in GDS formats. It starts with RTL design, and the resulting outcomes are polygon representations used for the fabrication stage. The GDS file is generated after certain consecutive stages of refinement, domain transformation, and optimizations. And they are done using a variety of EDA tools. RTL to GDSII implementation involves synthesis, floorplan, placement, clock tree synthesis, routing, signoff, and analysis before tape out for manufacture. Figure 3-1, shows the top level of

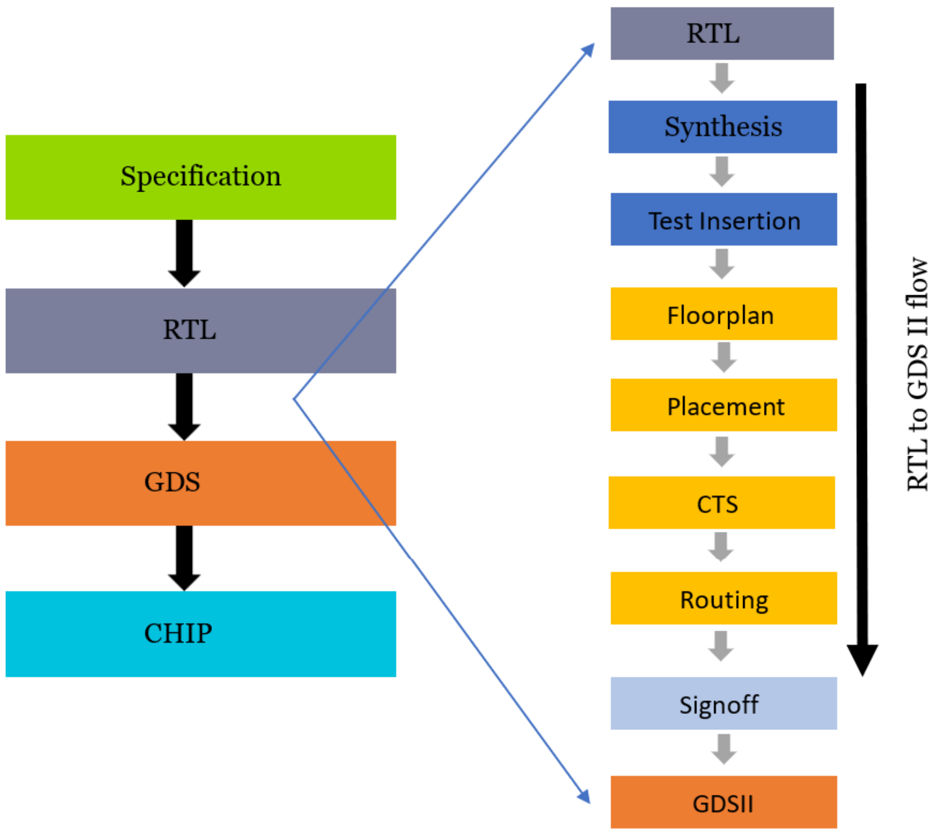

Figure 3-1:RTL to GDSII implementation workflow.

RTL to GDS implementation flow. From floorplan to routing stages is called physical implementation.

3.1.1 Logic synthesis

Logic synthesis plays a pivotal role in the semiconductor design flow. It is the first step in the RTL to GDS flow, where an RTL (Register Transfer Level) model is converted to its gate-level netlist mapped to a specific technology. This process is crucial as it derives the complex layout level of a design, which is abstracted by the behavioral and architectural layers. The functionality of a chip is described by HDL (Hardware Description Language) languages like Verilog or VHDL, and all the constructs in HDL are synthesizable. RTL refers to codes that can be synthesized. A synthesis tool, functioning like a compiler, maps HDL constructs onto the library of gates called standard cells. In general, a logic synthesis tool takes design (RTL), library, and constraints files to generate a netlist file, which can be a layout, schematic, or text file describing the connectivity of the gates. In Figure 3-2, the general input and output files for logic synthesis are shown. The input RTL and the output netlists are expected to be functionally the same. However, a synthesis tool can generate a netlist that functionally differs from the original RTL input. After synthesis, a logic equivalence check (RTL vs netlist) is done to determine the functional discrepancies.

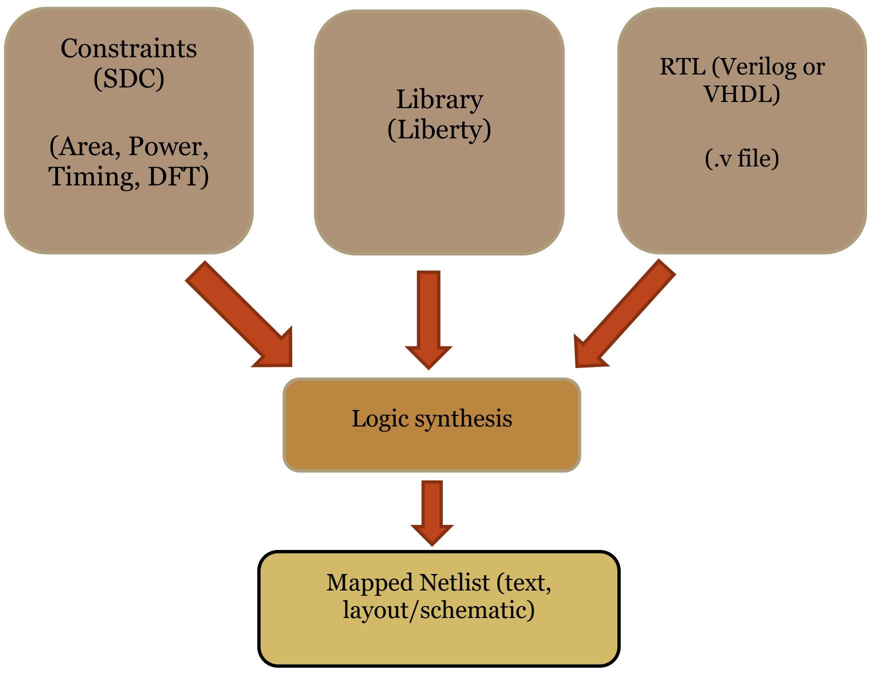

Figure 3-2:Logic synthesis inputs and outputs.

Logic synthesis automates and simplifies complex design. It can manage design details, bugs, and optimizations, thus improving productivity that would be impossible for manual design of complex circuits [1] [8].

The output netlist generation follows certain steps. These synthesis flow steps or tasks are RTL code parsing, elaboration, optimization, and translation to technology-dependent standard cells, primitive gates, registers, memory units, and FSMs, as shown in Figure 3-3.

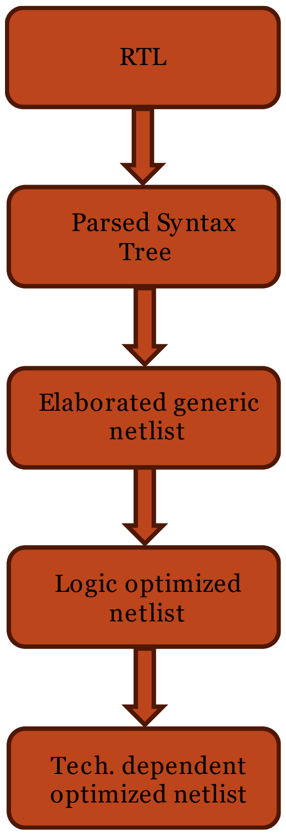

Figure 3-3: General Logic synthesis steps.

3.1.2 Test Insertion

In VLSI design, Design for Testability (DFT) techniques are used to improve the circuit's testability after logic synthesis. These are some external circuitries added into the functional design so that the chip can be tested for faults or defects after fabrication. This additional hardware reduces or at least helps find the faults after production. The key test insertions that are performed:

Scan Insertion: It is the simplest type of DFT used in IC testing. They are Flip-Flops (FFs), which are linked in a single or many chains to load and unload test stimuli and responses. The timing of the MUX introduced by scan insertion at each FF must be considered while functional optimizations on the design are being performed.

Test Points Insertion: Test points are introduced to improve a design's fault coverage or increase the number of test patterns generated by automatic test pattern generation (ATPG) for the same fault coverage. It is better practice to insert this during synthesis.

BIST insertion: Built in self-tests are embedded input test pattern generator and output analyzers. This makes testing more accessible, and the IC can test itself without external hardware.

Boundary Scan: This technique involves adding additional logic to the I/O pins of a device, allowing the testing of interconnects between devices on a board without requiring physical access to the board’s traces.

Memory BIST (MBIST): This is a specialized version of BIST used to test embedded memory elements in a design.

Automatic Test Pattern Generation (ATPG): ATPG tools automatically generate test vectors that can detect or classify manufacturing faults.

These techniques help ensure that the manufactured chips are defects-free, improving yield and reliability [9].

3.1.3 Floorplan

Floorplan is the very first step in physical design. The layout of the design is defined in this step. The design's physical size and aspect ratio are defined and go into the tool as inputs. It creates boundaries like core area, die area, blockages, place pins, I/O pads, insertion of power grids, and hard macros. The tool creates rows and columns according to standard cells' and other blocks' area, speed, and design constraints. A good floorplan ensures the best performance, power consumption, and area usage. These metrics are optimized in every stage of physical design iteratively. Still, a good initial state established with a good floorplan can reduce the iteration and provide top-level timing closure, as it depends on how cells and blocks are placed next. But there is no best floor plan for a design. Usually, the one that meets the design requirement is followed, while there is always room for improvement. The outcome of the floorplan works as the physical constraint of the subsequent steps. Depending on design size and complexity, Floorplan iterations can take days to weeks. Although this step can be expensive in terms of time and money, it’s crucial in physical design.

The main input files are usually synthesized netlist (.v file), physical libraries for standard cells (.lef files), design constraints file (.sdc file), timing libraries (.lib file), MMMC view file. After floor planning, the descriptions and information on the placement of standard cells, macros, I/O pads, and power grids are written in a design exchange format (.def) file used in the placement stage. Area optimization is done after floorplan to get the optimized area constraint [1] [9].

3.1.4 Placement

Placement is the step when standard cells and various modules are physically placed within the chip die area. During floorplan, big macros and block positions are defined, whereas, during placement, millions of standard cells are defined on the layout. The connections between cells and blocks are also calculated. During placement, the wiring is not done, estimates of wire length are created, and placement is done to meet the timing constraints. Optimizations are performed to minimize the total estimated wire length. Congestion problems are also handled by moving out cells from high-congestion regions during this stage.

A placer tool also checks and fixes position legalization, alignment, and gap filling between cells for compaction. Power and thermal optimizations are also performed to improve energy efficiency and avoid hotspots.

3.1.5 Clock Tree Synthesis

In VLSI design, clock tree synthesis is one of the critical steps in the physical design flow. This creates a topology that connects the clock from the clock port to the clock pin of sequential cells in the design. The primary goals of CTS are to balance the skew and minimize the insertion latency. Skew is defined as the difference in the arrival times of the clock signal to reach any pair of registers. It is either local, between registers with a valid timing path, or global, between any pair of registers regardless of a valid timing path. Balancing of skew is vital for satisfying all design constraints. On the other hand, insertion latency refers to the time delay that the clock takes to travel from the clock port to the clock pin of a sequential element. The delay is brought about by the parasitic capacitance and resistance of the nets.

To balance these delays, clock inverters and clock buffers are used to create the clock tree. There are several structures to build a clock tree to achieve a minimum insertion delay as well as to balance the skew. Some are the H-Tree structure, X-Tree structure, Geometric Matching Algorithm (GMA), Pi Tree structure, and Fish bone. Adding inverters and clock buffers in the clock signal path introduces additional delay of their own. But for the delay and clock time balance, they are very useful. Another benefit of these elements in the clock signal path is that they can be used to strengthen the signal, which is degraded due to inevitable parasitics.

CTS aims to find the right balance, ensuring minimal skew and insertion delay while maintaining signal integrity. This complex optimization problem is a critical aspect of VLSI design [10].

3.1.6 Routing

Routing is the process of establishing connections between different standard cells, the macros, and the pins, utilizing connecting wires to achieve the required design layout. After CTS and optimizations, the routing tool establishes pathways between cells, buffers, blockages, and I/O pins with metal and vias. The physical connectivity is determined by the logical information provided in the netlist. A router tool's primary job is to route the data signals. Power lines and clock signal paths are usually routed during the creation of the power delivery network and clock tree synthesis, respectively. As there are many components in a digital chip, routing requires a lot of computational time and power. Routing must respect timing, design rules like LVS and DRC, and congestion constraints while performing physical connections. This is why routing is one of the most complicated steps in digital design.

Routing is usually divided into two tasks – global routing and detailed routing. Global routing assigns tracks for nets while minimizing global cell congestion and diversions. Detail routing lays the actual layout of nets. It uses the regions provided by global routing. Detail routing tries to make sure there are no DRC violations and improves signal integrity. After routing, signal integrity optimizations and insertion of filler cells and metal fills are performed.

3.1.7 Signoff

Signoff refers to the methods of verifying design rules of foundry, manufacturability, and original functionality in the netlist. The verification techniques are both logical and physical. It is only a reporting stage. The changes in the layout or design are not made during the signoff. However, this signoff reports the violations that need to be fixed in the implementation stage of the flow. Signoff can be done on the front end of the design, which is on RTL. Logic Equivalence Checks (LEC), usually done after synthesis, is a front-end check. Backend checks a list of checks done post layout. After the layout is finished, the circuit and parasitics are extracted from the layout. The backend checks are performed after the extraction stage. Major signoff checks are static timing analysis (STA) for timing constraints between data and clock signals in synchronous circuits, layout versus schematic (LVS) check for comparing circuit extracted netlists from layout and logical source netlist to check functionality of the final layout, electrical rule check (ERC) for reporting faulty connections in the layout, design rule checks (DRC) to check if foundry provided rules are respected and the design is manufacturable. These rules can vary according to technology and foundry [1] [11].

4 Chapter 4

4.1 Cloud computing and Job management

Cloud computing refers to the technology where computing services and resources like data storage, network, CPUs, and memories are hosted and used by end users without owning and maintaining any physical computing systems. The physical computing systems are provided and maintained by cloud service providers. Customers can request and use the required resources from the pool of near-infinite technology servers.

As the complexity of computing, like scientific simulations, data analytics, graphics processing, etc., increased over time, the need for a powerful computer for heavy tasks also increased. However, using a heavy single computing system is expensive and often unsuitable as different tasks require different systems. Parallelizing multiple CPUs to work together on resource-hungry tasks solves the problem but still lacks flexibility, and often, users are bound to use one vendor-specific system. Cloud computing solves the problem by providing various computing nodes to multiple customers through time-sharing. It is a distributed computing service model, enabling multiple customers to get access to systems as per their demands and budgets. Figure 4-1 shows a general cloud computing model that can provide many services to any end-user over the Internet.

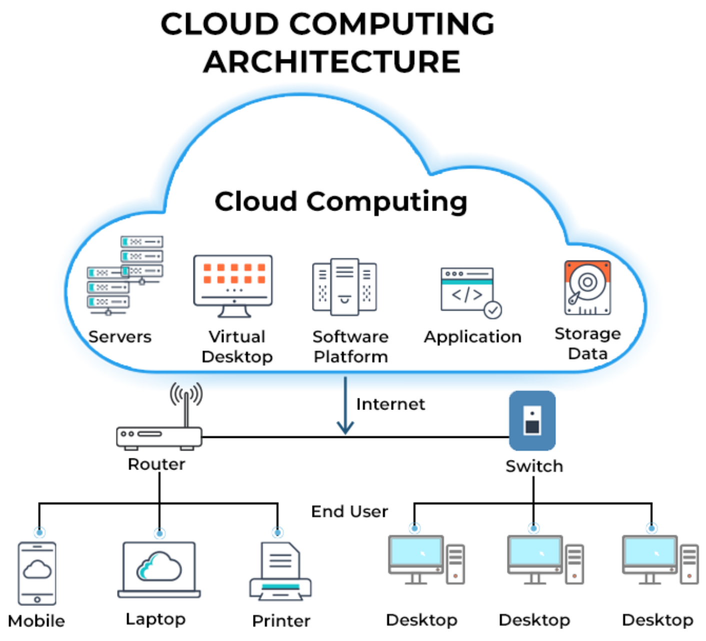

Figure 4-1:Cloud computing structure with various services and applications [13]

Cloud servers, often called data centers, are physical servers consisting of many computers connected to work together over a very high-speed network. There can be applications servers, storage facilities, and a specific software platform for clients from anywhere.

4.2 Job management

A job within the context of cloud computing refers to the execution of computing tasks. It can be a simple command or a heavy application. Before a compute node is used for job execution, a job scheduling software works as an intermediary in between the user and the compute node. This software is often called a Job scheduler. This software works as a central managing hub for organizing, communicating, and distributing jobs on the compute hosts. It is designed to suit an organization's needs and can be customized as per needs. This software can monitor jobs and report resource usage and failures. The automation of job executions, prioritization of jobs, and automatic recovery in terms of failure make management of a large number of submitted jobs easier.

LSF (Load Sharing Facility) from IBM is a job scheduling and resource management utility that provides a system that can take user job requests, find suitable compute resources for execution and monitor job progress. Figure 4-2 shows how LSF handles job requests.

A cluster is a group of computers (hosts) that work together. Each computer in a cluster is connected to each other with high-speed network and each host runs its own instance of operating system. One of the hosts from the cluster works as a master host. A master host does not execute any submitted job. It handles scheduling and dispatching of jobs.

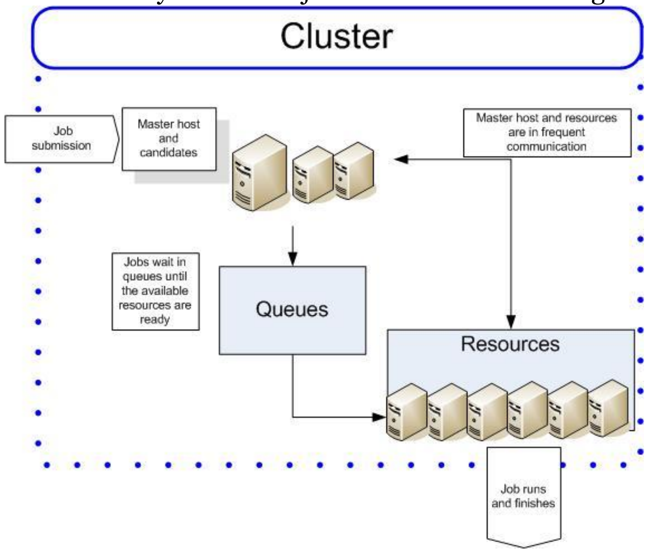
Cluster
Job submission
Master host and candidates
Master host and resources are in frequent communication
Jobs wait in queues until the available resources are ready
Queues
Resources
Job runs and finishes

Figure 4-2: Overview of Load Sharing Facility (LSF) framework [14].

The master host is always in constant communication with compute hosts for job deployments and monitoring the progress of jobs. There are some candidate hosts listed for the backup service of the master host in case the current master host faces failure or shuts down.

When jobs are submitted, they go to the queue first. It works like a container for all jobs. All the jobs in the queue are in wait states. LSF checks resource demand and availability for submitted jobs while the jobs are in the queue. When LSF finds the suitable compute host available, it deploys the job on the compute host, and the job is executed. The dispatch of jobs is ordered according to job priority. LSF has different methods of controlling job priorities. They can be automatic job priority escalation, user-defined, and absolute priority scheduling (APS). These methods allow administrators to customize priority management according to needs by allowing them to define job priority value according to job-related factors.

5 Chapter 5

5.1 Methodology

For the characterization of the implementation workloads, three design test cases were chosen. The test cases are for Arm CPU cores. Two of the three test cases, are from same CPU design family and type, only differing in technology nodes (3nm and 5nm designs). For discretionary reason, the specific design names, information, host information, and the comparison of EDA tool performance metrics shall be avoided in this report.

The design testcases will be labelled as Design 1 (3nm), Design 1 (5nm), Design 2 (3nm). The type-number refers to the type of CPU core. Design 1 is smaller than Design 2, considering the number of gates and instances. The tests were done on both Intel and AMD machines. The host type will be labeled as Host1 and Host2 without specifying the type from the two in Chapter 6. Hyperthreaded processors were chosen for all the jobs.

The implementation workload refers to synthesis and placement and route (PnR) flow steps. For synthesis and PnR steps, Cadence tool genus and innovus were used, respectively. For Static Timing Analysis (STA), tempus from Cadene was used. As the flow supports different EDA tools integration, Synopsys tools – Formality, Fusion compiler, Primetime, RTL Architect, StarRC, and Tweaker can also be used. However, within the scope of the thesis, these tools were not covered. Benchmarks of the tools can be a point of interest in the future.

ARM internal implementation flow was used to deploy the implementation tasks. The steps were defined by the flow and were kept unchanged for different tests. A design flow is comprised of scripts used to chain different EDA tools for well-defined, flawless implementation tasks. The flow is designed according to design objectives and can vary from design to design, as well as EDA tools features and versions. During the experiment of each implementation step, all runs were made under the same conditions and with the same version of the flow. It's always better to collect data with repeated measurements. However, within the limited duration of the thesis, only a single run was performed for each implementation job.

The schema was built considering multiple factors that can impact the runtime. Tool version, technology, scenario for the test run, host type, AWS instance size and type, host CPU frequency, cache memory, and operating system information were collected to understand the compute time dependency on the host specification and tool features. All jobs were run with on-demand pricing. The performance metrics, CPU time, average and maximum memory usage, maximum swap availability, runtime, and turnaround time were collected from the LSF log. In Chapter 6, we present runtime and memory dependency on varying numbers of CPUs on hosts booked for each implementation step.

6 Chapter 6

6.1 Results and Analysis

Synthesis:

Synthesis was performed both using the superflow (lsynth, logical synthesis) and sub flow steps – generic synthesis (syn_gen), mapping (syn_map), and optimizations (syn_opt). The runtime and maximum memory usage by the Cadence genus tool for the sample designs are shown in Figure 6-1.

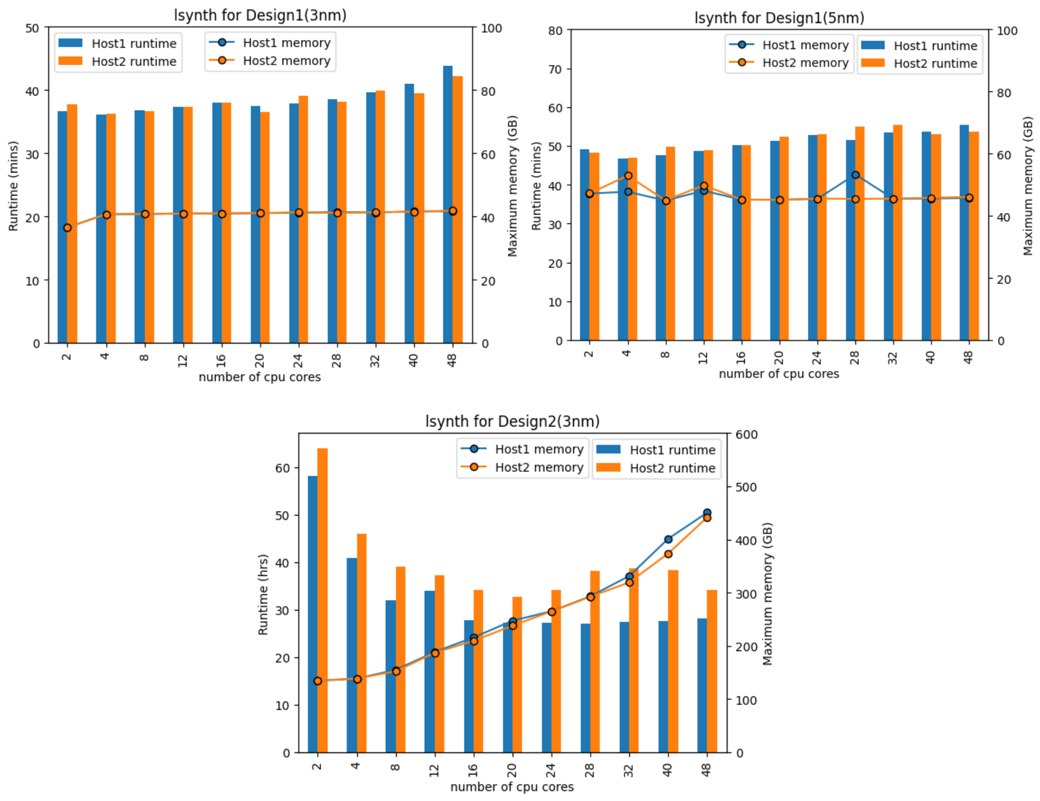

Figure 6-1:Resource usage of logical synthesis superflow step.

For both technologies and hosts, logical synthesis takes around 35 minutes and 50 minutes, respectively, for Design1. It is evident that, for smaller designs like Design1, the tool is not benefitting from multi-core runs. Multithreaded runs are efficient and automatically enabled according to task load. For smaller designs like Design1, it is better not to launch synthesis with multi-CPU to reduce the compute cost. The maximum memory usage is also almost constant for different CPU runs. For 3nm and 5nm nodes, 50 GB and 60 GB can be demanded as safer limits, respectively, for logical synthesis. On the other hand, Design2 is found to leverage the benefits of multi-CPU core runs. We see the reduced runtime as the number of CPU cores increases by up to 20. Then, the runtime becomes constant as we go beyond 16 cores. The memory usage also increases with increasing cpu. Host1 always outperforms Host2 for the bigger core design, giving a maximum of 18% performance boost for 16 core and 8 core runs. But the absolute run time is smaller in 16 core run configuration. So, for design cores comparable to Design2 can use 16 cores and around 300 GB of memory safe limit with host type Host1 for optimum performance for logical synthesis tasks. For 4 cores and 28 cores

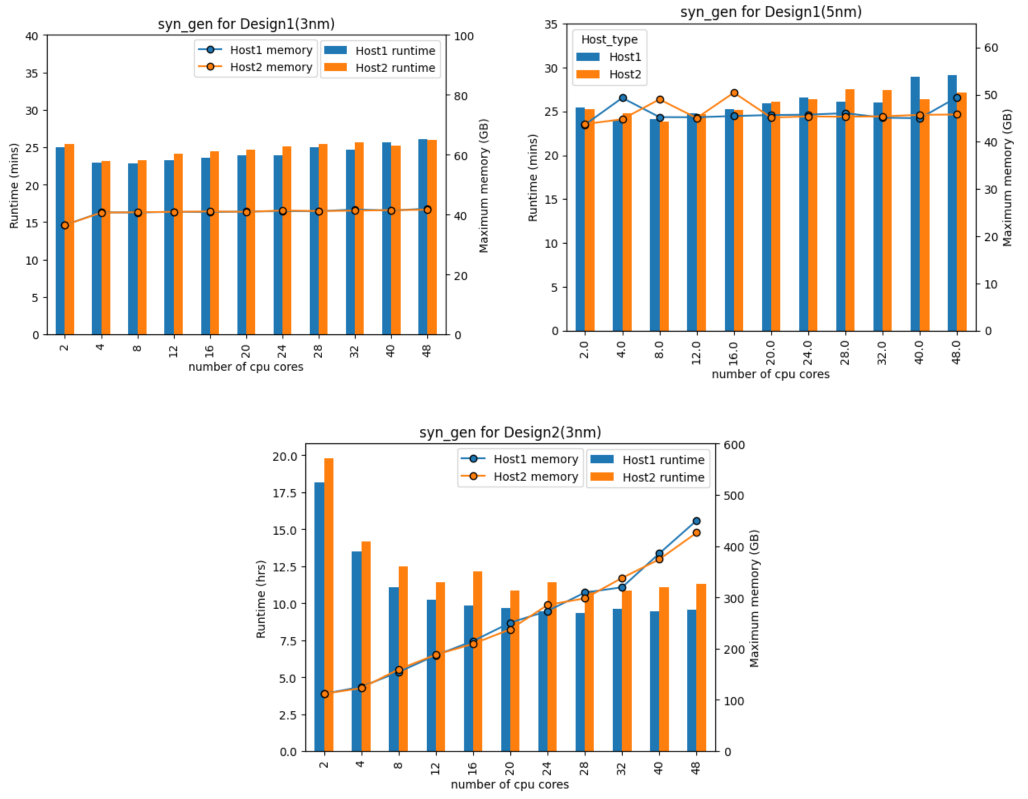

Figure 6-2: Generic synthesis (syn_gen).

runs, around 5 GB of extra memory is consumed by Host2 and Host1, respectively. It is speculated to be statistical and can be verified by repeated measurements. During the tests, only single-run data was collected. The first step of synthesis is generic synthesis. The runtime and memory usage for generic synthesis are shown in Figure 6-2.

The syn_gen step takes up around 62% of the logical synthesis step's runtime. For this substep, the trend is similar to that of lsynth. The runtime is almost constant, around 25 minutes. For increasing the number of CPUs above 16 cores, both Host1 and Host2 take more time to finish the job. For Design2, syn_gen utilizes a multi-core setup and provides reduced time. The efficiency improvement is saturated at 16 core runs. Host1 is almost 18% faster than Host2 in this case. Memory usage linearly increases with CPU cores and similar, around 250 GB for both Host1 and Host2. The optimum configuration can be 16 cores of Host1 CPU with a 250 GB of memory limit.

In Figure 6-3, similar metrics for technology mapping of synthesis are shown. For mapping, a similar trend of lsynth and syn_map is found. There is no improvement in increasing the CPU for Design1. For 3nm and 5nm nodes, syn_map takes around 12 minutes and 15 minutes, respectively. The max memory usage is around 15 GB for both nodes. For more heavier computation in the case of Design2, 20 cores run has the minimum runtime for both Host1 and Host2. Host1 is again faster than Host2 by 10%, 16%, and 23% for 16 cores, 20 cores, and 24 cores run, respectively. The maximum memory usage increases with increasing CPU, with a maximum of nearly 380 GB for 48 core runs. The runtimes for 16 and 20 cores runs are 4.92 and 4.55 hours, respectively, for Host1. Although 20 core run is only around 22 minutes faster than 16 core run, the pricing for increased resources needs to be checked as equivalent to reduced runtime for 16 core run test. It is not significant in this test, but it should be considered for production runs on bigger scale.

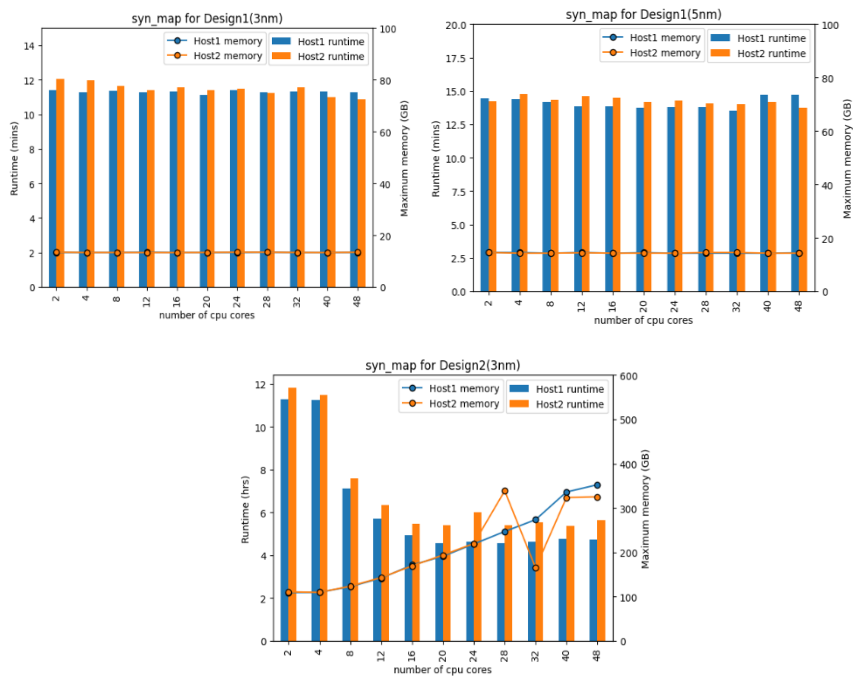

Figure 6-3: Technology mapping (syn_map).

In Figure 6-4, results of post mapping optimizations are shown.

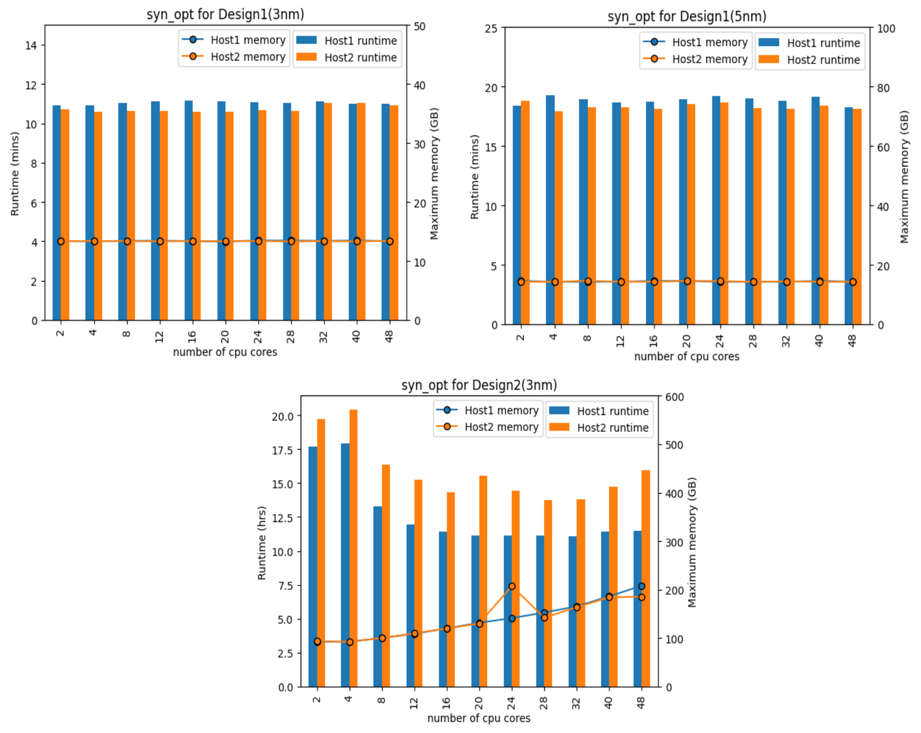

Figure 6-4: syn_opt runtime and resource utilization.

For 3nm and 5nm test cases, runtimes are 11 minutes and 17 minutes, respectively. Maximum memory demand should not be more than 15 GB. Host2 is faster in this case, but only by a minute or less, and does not show a significant difference. The optimization step is faster on Host1 for Design2. The task leverages the multi-core CPUs, and the minimum point is for 16 core runs with 160 GB of memory utilization.

Floorplan:

In Figure 6-5, the runtime of floorplan (fplan) step for Host1 and Host2 is presented. The runtime for Design1 slightly improved, this time for multithreaded runs. For both

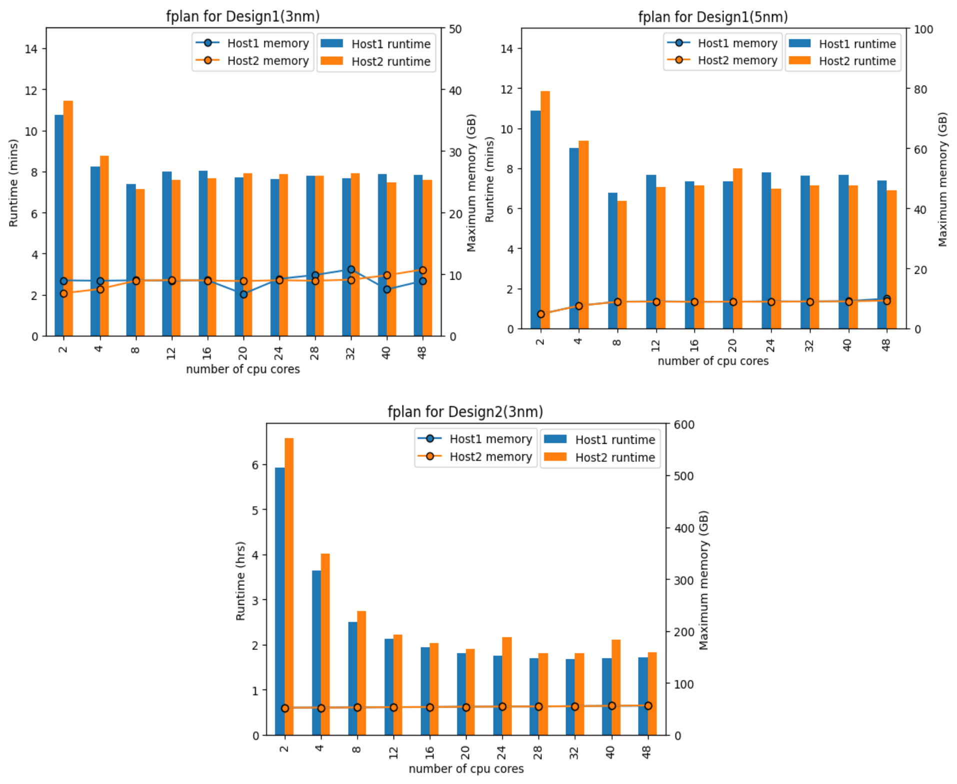

Figure 6-5: Floorplan runtime and resource utilization.

3nm and 5nm, the minimum runtime is achieved for 8 cores run with 10 GB of memory usage on Host2. In the case of Design2, Host1 is faster in all runs and runtime reduction saturates beyond 16 core test case. Memory usage is almost constant, around 55 GB.

Prects: Prects is a placement and optimization step. It runs a check-design command to check if there are any major problems in the design prior to running placement. Then, it runs placement on the design, followed by optimizations. It is a long task and depends on the size of the design. Figure 6-6, shows prects runtime and resource utilization. For

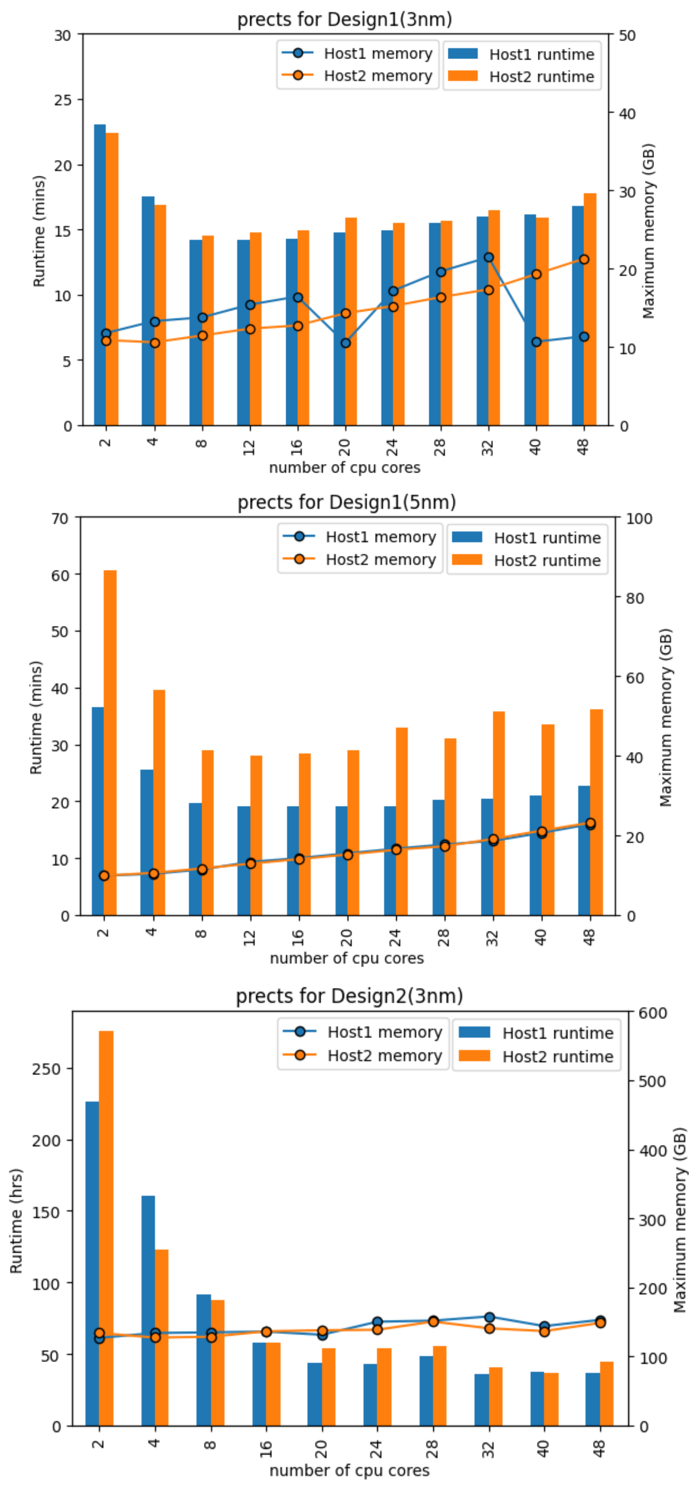

Figure 6-6: Pre clock tree synthesis step (prects).

Design1 at 3nm nodes. Host1 and Host2 provide similar performance, and there is not any significant change after 8 cores run. Memory usage increases with the number of cores booked for the job except for the case of 20, 40, and 48 cores test run on Host1. More tests need to be done to verify if this is a statistical variation or host specific reason. For Design1(5nm), the fastest configuration for the job is 8 cores, 20 GB on Host1. For Design2(3nm), 32 cores with 200 GB of reserved memory would be the fastest option on Host1 with 19 hours of runtime.

CTS: During the CTS step, actual clock tree synthesis takes place. Figure 6-7, shows the

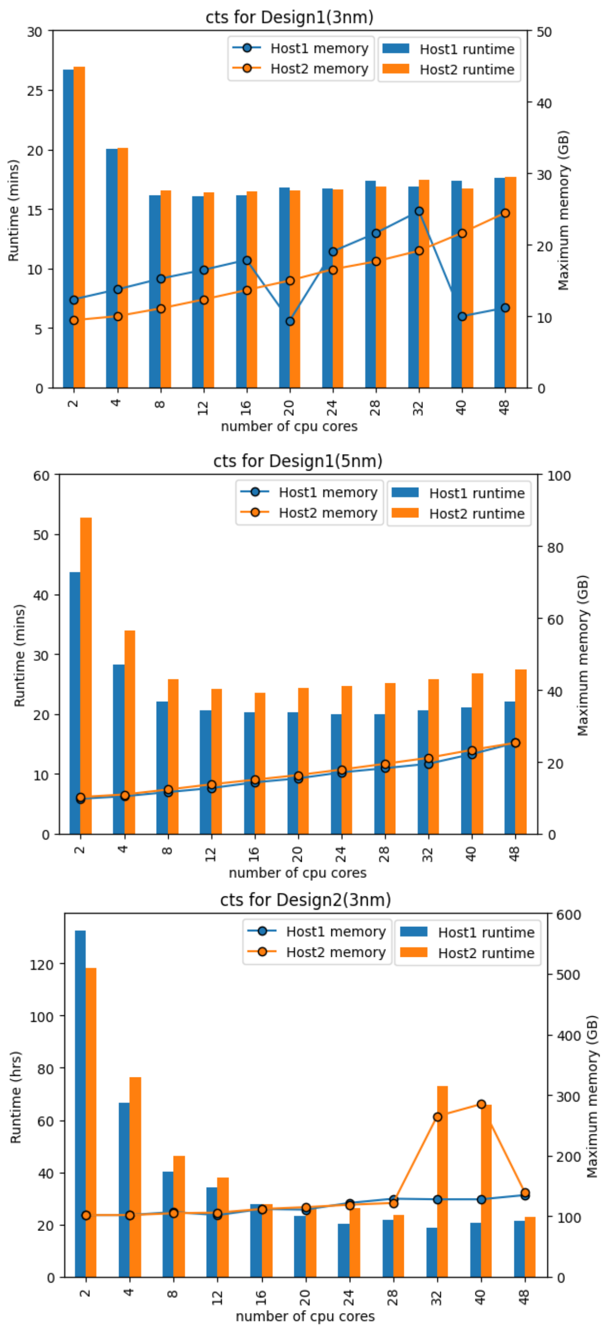

Figure 6-7: Clock Tree synthesis (CTS).

metrics for CTS. For Design1 (3nm), 8 cores run is the optimum choice. The 5nm design takes more time for CTS. As they are not exactly the same, the technology dependency on turnaround time can't be established. The minimum runtime is achieved with 16 cores run with 20 GB of memory utilization. For Design2, on Host1, the minimum compute time is achieved with 32 cores run, being 19 hours, whereas, on Host2, the minimum compute time is 22 hours with 48 cores. For 32 and 40 cores run on Host2, the runtime is out of the general trend. As both jobs were run on the same host instance of the same number of CPUs, memory, and specifications, it is speculated to be due to some random events that caused the delay or multiple different jobs on the same host. Further test runs will be required to establish the correct cause.

Postcts: During postcts, place and area optimizations and design checks are performed. The results of the test runs are shown in Figure 6-8. Postcts for Design1 (3n) is the fastest

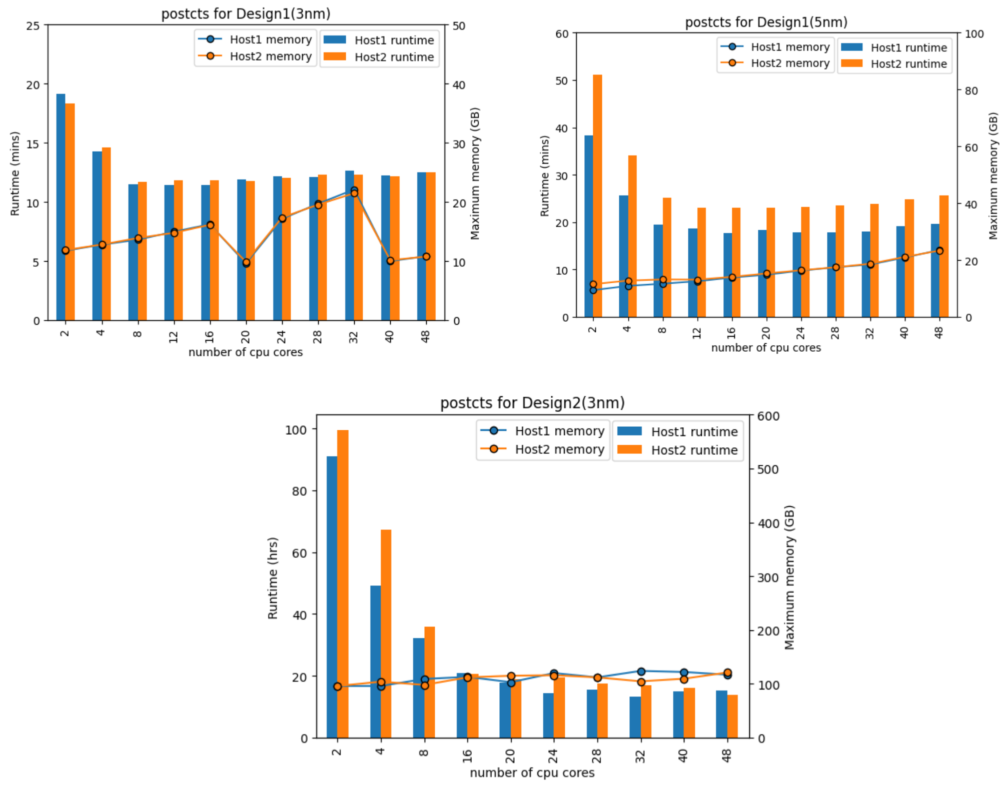

Figure 6-8: Postcts step.

on Host1 with 8 cores and 15 GB configuration, whereas Design1(5nm) is fastest on Host1 with 16 cores and 20 GB of memory specification. The step is fastest on Host1 with 32 cores and 140 GB of memory for Design2. Host2 takes the same time for 48 cores run. 24 cores run is only 1h slower than 32 cores run for an overall 8 cores difference. So, the cost of a bigger CPU or a longer time for a cheaper host CPU can be considered according to budget and schedule.

Route: Routing run data for test designs are presented in Figure 6-9. For Design1(3nm), routing does not take much advantage of multi cores run. 8 cores, 20GB

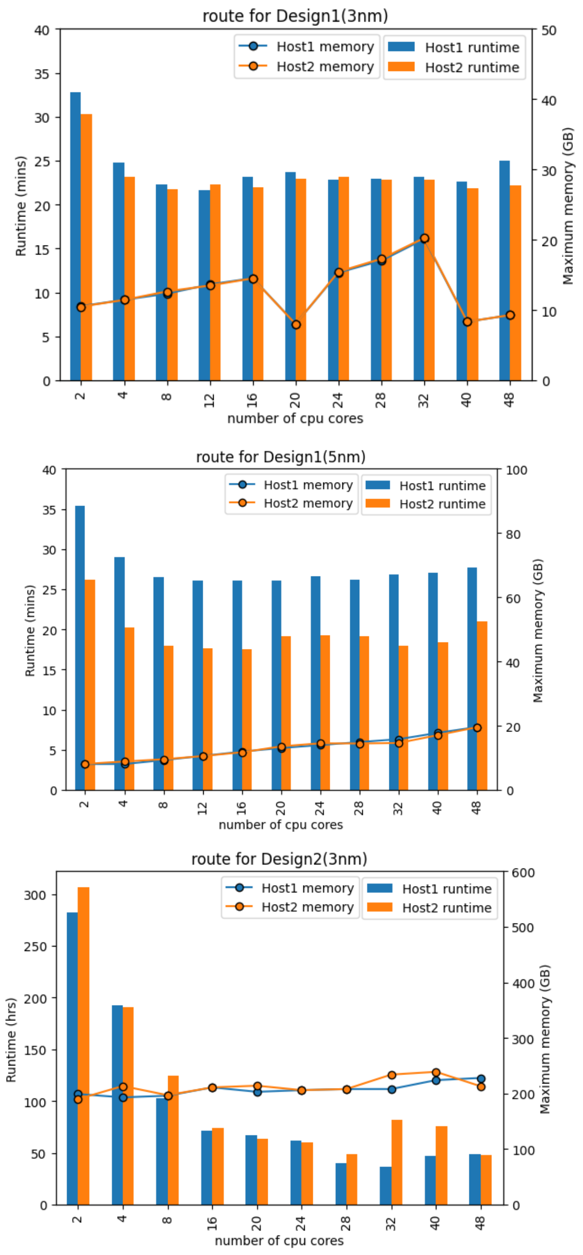

Figure 6-9: Routing.

of memory on Host1 is the optimum configuration. For Design1(5nm), Host2 is faster than Host1 by around 10 minutes in most of the test runs. Memory usage is almost the same for all the tests and linearly increases with higher CPU demand. The optimum is 8 cores and 15 GB of memory on Host2. For Design2, the fastest run was around 37 hours for 32 cores run on Host1. For Host2, the run time keeps getting faster as the number of demanded CPUs increases, the lowest being 46 hours for 48 cores run. The memory utilization is same for both hosts. For routing, Host1 of 32 cores and 250 GB of memory is the optimum host.

Postroute: The minimum for postroute runtime is for 8 cores run with 30 GB memory

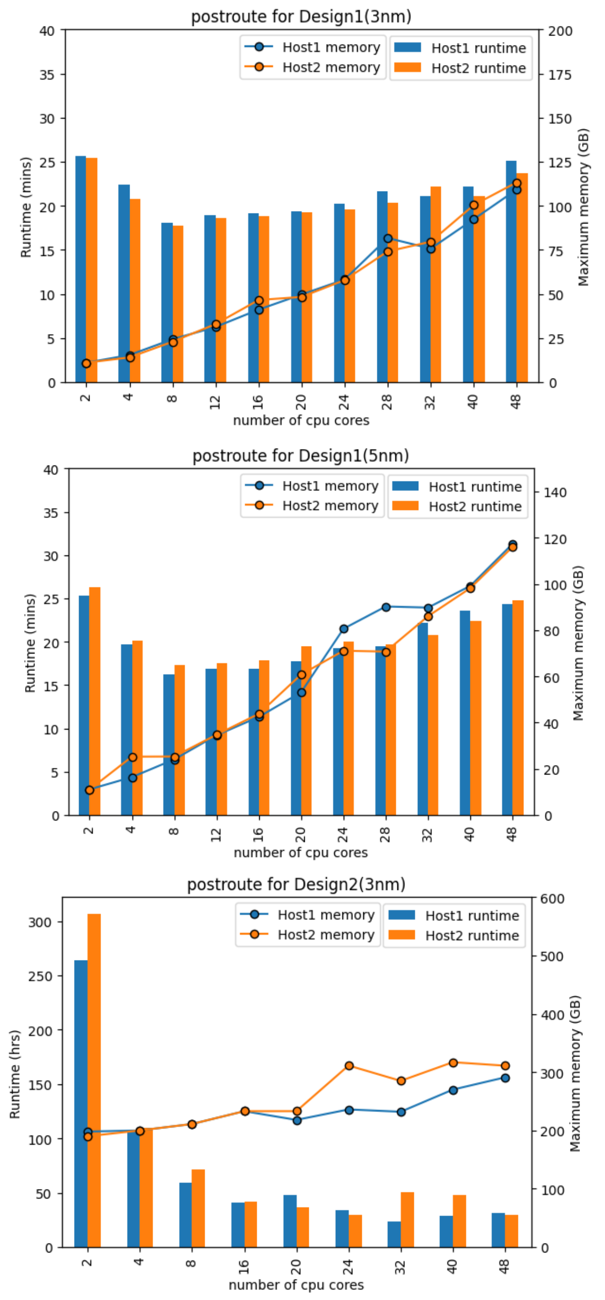

Figure 6-10: Postroute.

for both Design1(3nm) and Design1(5nm), as shown in Figure 6-10. For Design2(3nm), on both hosts, the job gets faster as the number of cores is increased, with runtime being 23.5 hours for 32 cores on Host1 and 29 hours for 24 cores run test. Memory usage increases to a maximum of 330 GB as the multi-core utilization increases.

STA: Static Timing Analysis (STA) was run only on the smaller designs Design1(3nm) and Design1(5nm) as shown in Figure 6-11.STA for Design2(3nm) was not within the scope of this internship.

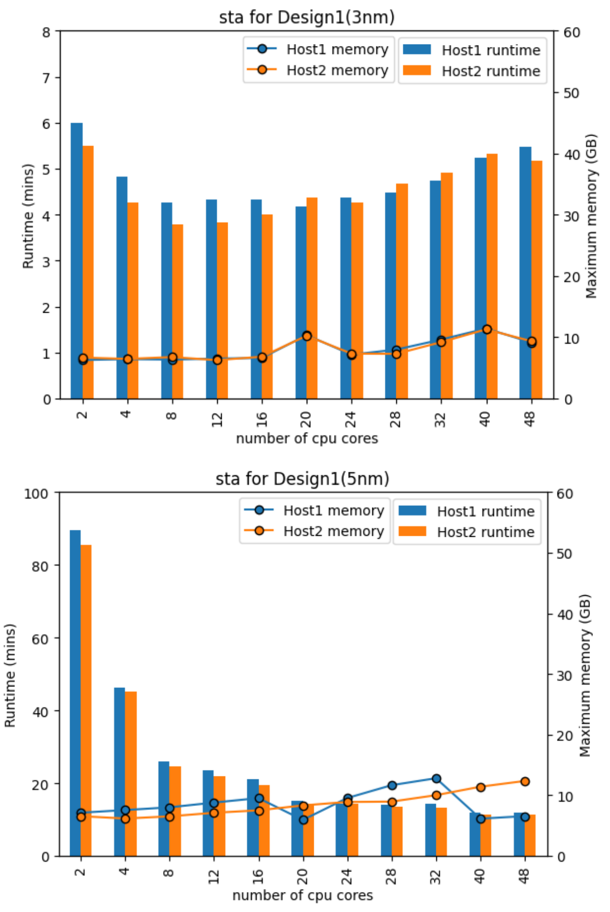

Figure 6-11: Static Timing Analysis.

STA on Design1(3nm) is very short and finishes within maximum of 4~6 minutes. STA on Design1(5nm), on the other hand, takes more than an hour for 2 cores to run and goes down to 12 minutes for 48 cores. A better understanding of the STA load on hosts can be understood when running jobs for bigger designs. Memory utilizations are the same for both hosts in this case.

Finally, we propose that with a design of size comparable to Design2(3nm), 16 cores Host1 CPUs with memory of not more than 300 GB should be chosen for logical synthesis superflow step and sub-steps (syn_gen, syn_map, syn_opt). For the floorplan stage, 16 cores Host1 CPUs with 60 GB memory give the best TAT. For all the subsequent steps, prects, cts, and postcts, 32 cores Host1 CPUs can be the fastest compute hardware. The memory demand should be higher, more than 250 GB and 330 GB for route and postroute, respectively. For other steps, 200 GB can be a safe limit. For both Design1(3nm) and Design2(5nm), parallel computing is not rewarding for synthesis (both superflow step and sub flow steps) and floorplan. As a very small design core, it does not represent longer and more complex computations. In this case, 8 or 16 cores with a maximum 80 GB memory capacity should be well suited for physical implementation jobs.

7 Chapter 7

7.1 Conclusion and Future Work

This thesis work was conducted to understand load implementation loads on cloud and on-premises hosts so that jobs can be better and more efficiently deployed in the future. For most implementation jobs, Host1 is found to be faster than Host2 for both synthesis and physical implementation workloads. It is found that multi-threaded runs are leveraged by the tool only for bigger designs with a high (million or more) number of gates.

The data harvested for the thesis research will provide insight into EDA loads on clouds and optimizations of the jobs for ARM. The efficiency heavily depends on software, i.e., EDA tools and host specifications. So, the results can be used for future reference for EDA vendor support. In the future, these metrics can be used to build an optimization and prediction model for EDA workloads on the cloud. This work has helped Arm to define optimum runs for the cases under study (small/big design tests) for 3nm and 5nm technology nodes. It has also helped to define a data model that can be used for systemic data collection for use with Machine Learning models on top of it in the future and automatically build optimum testcase for any design /technology.

8 References

[1] S. Saurabh, Introduction to VLSI Design Flow, Cambridge: Cambridge University Press, 2023.

[2] "what will that chip cost," semiengineering, 30 Oct. 2023. [Online]. Available: https://semiengineering.com/what-will-that-chip-cost/. [Accessed 25 July 2024].

[3] J. M. A. a. S. S. N. Sehgal, "Is the EDA Industry Ready for Cloud Computing?," *IETE Technical Review,* vol. 33, p. 345–356, October 2015.

[4] "Google, AMD, and Siemens demonstrate the power of the cloud for EDA," Siemens Resource Center, 2024.

[5] A. H. a. S. Reda, "Characterizing and Optimizing EDA Flows for the Cloud," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems,* vol. 41, pp. 3040-3051, 2022.

[6] "The Official History of Arm," arm, [Online]. Available: https://newsroom.arm.com/blog/arm-official-history.

[7] "Arm Sophia Antipolis Office," arm, [Online]. Available: https://careers.arm.com/france-sophia-antipolis-office.

[8] A. G. a. K. W. K. Srinivas Devadas, Logic synthesis, New York: Mcgraw-Hill, 1994.

[9] K. Abbas, Handbook of Digital CMOS Technology, Circuits, and Systems, Springer Cham, 2020.

[10] G. W. a. S. J. a. Y. W. a. G. Zhang, "An efficient clock tree synthesis method in physical design," *2009 IEEE International Conference of Electron Devices and Solid-State Circuits (EDSSC),* pp. 190-193, 2009.

[11] W. Ries, "Rule-based implementation of correct and efficient VLSI design rule checking," pp. 205-209, 1989.

[12] S. P. G. C. P. V. A. Nanda, "Resource Aware Scheduling for EDA Regression Jobs," in *Euro-Par 2019: Parallel Processing Workshops*, Göttingen, Germany, 2019.

[13] P. P. a. C. BasuMallick, "What Is Cloud Computing? Definition, Benefits, Types, and Trends," spiceworks, 09 February 2022. [Online]. Available: https://www.spiceworks.com/tech/cloud/articles/what-is-cloud-computing/. [Accessed 25 July 2024].

[14] "IBM Spectrum LSF, LSF, load sharing facility, introduction," IBM, [Online]. Available: https://www.ibm.com/docs/en/spectrum-lsf/10.1.0?topic=overview-lsf-introduction. [Accessed 25 July 2024].