AUTOMATION OF NETWORK ON CHIP DESIGN FLOW AND CREATION OF A FRAMEWORK FOR PPA (POWER, PERFORMANCE, AREA) IMPROVEMENT ANALYSES

MTech Thesis

By
ARAVA VEERA VENKATA SNEHITHA

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE
JUNE 2021

(This page is intentionally left blank)

AUTOMATION OF NETWORK ON CHIP DESIGN FLOW AND CREATION OF A FRAMEWORK FOR PPA(POWER, PERFORMANCE, AREA) IMPROVEMENT ANALYSES

A THESIS

Submitted in partial fulfilment of the
requirements for the award of the degree
of
Master of Technology
in
Electrical Engineering
with specialization in
VLSI Design and Nanoelectronics

by
ARAVA VEERA VENKATA SNEHITHA

DEPARTMENT OF ELECTRICAL ENGINEERING
INDIAN INSTITUTE OF TECHNOLOGY INDORE
JUNE 2021

(This page is intentionally left blank)

INDIAN INSTITUTE OF TECHNOLOGY INDORE

CANDIDATE'S DECLARATION

I hereby certify that the work which is being presented in the thesis entitled **AUTOMATION OF NETWORK ON CHIP DESIGN FLOW AND CREATION OF A FRAMEWORK FOR PPA(POWER, PERFORMANCE, AREA) IMPROVEMENT ANALYSES** in the partial fulfilment of the requirements for the award of the degree of **MASTER OF TECHNOLOGY** and submitted in the **DEPARTMENT OF ELECTRICAL ENGINEERING, Indian Institute of Technology Indore**, is an authentic record of my own work carried out during the time period from July 2019 to June 2021under the supervision of Dr Mukesh Kumar, Associate Professor, IIT Indore and Mr Amit Prakash, Senior Manager, Qualcomm India Pvt Ltd, Banglore.

The matter presented in this thesis has not been submitted by me for the award of any other degree of this or any other institute.

07-06-2021

Signature of the student with date
(ARAVA VEERA VENKATA SNEHITHA)

This is to certify that the above statement made by the candidate is correct to the best of my/our knowledge.

7/06/2021 07 June 2021

Dr Mukesh Kumar **Mr Amit Prakash**

ARAVA VEERA VENKATA SNEHITHA has successfully given his/her M.Tech. Oral Examination held on**7th June 2021**.

7/06/2021 07 June 2021

Signature(s) of Supervisor(s) of M.Tech. thesis
Date:

Convener, DPGC
Date: 07/06/2021

Signature of PSPC Member #1
Date: 08-06-2021

Signature of PSPC Member #1
Date: 08.06.2021

(This page is intentionally left blank)

ACKNOWLEDGEMENTS

I want to take this opportunity to express my deepest gratitude to my research supervisor **Dr MukeshKumar**, Associate Professor at IIT Indore. I am indebted to him for giving his valuable time to guide me all the way technically as well as non-technically. I am very much thankful to him for motivating me in my distressed conditions. Finally, encouragement and guidance helped me to shape my future and the completion of my MTech.

I would like to wish my sincere appreciation to my co-supervisor, **Mr Amit Prakash**, Senior Manager at QUALCOMM India Pvt Ltd, Banglore. He convincingly guided and encouraged me in the project as well as helped me to gain a big leap from campus to corporate. His pieces of advice kept me going on in this work,and this would not have been possible without his inputs and counselling. I am also thankful to **Ms Swathi Rajput**, PhD Scholar at IIT Indore, who helped me to sharpen my VLSI knowledge. I would also like to thank my friends, Riya Verma and Nikita Golait for the idea's discussion, support, and needful comments on my research work.

I want to acknowledge the great people of the QUALCOMM community, who provide me with training and technical support to fulfil my requirement for research. The enthusiasm and incentive given by this organization gave me the strength to stand up against the difficulties that I faced during my whole year. It was a memorable year at QUALCOMM as Research Intern.

I sincerely acknowledge the support of the **IIT Indore community** to provide resources, lab equipment, and facilities for my research work. Also, I would like to thank the **Ministry of Human Resource and Development (MHRD)** for providing TA Scholarships.Last but not least, I wish to acknowledge the support and great love of my family, my mother, father, and my brother for emotional support.

(This page is intentionally left blank)

DEDICATION

Dedicated to my parents and frontline COVID warriors

(This page is intentionally left blank)

Abstract

High latency, bandwidth, and throughput requirements have originated the deployment of multi-core processors in SoCs.As the transistor sizes shrink and the IPs(Intellectual Property)used in the chip increases, the quality of service begins to crumble. The conventional cross-bar and bus approach prove to be impotent. Network on the chip is the contemporary communication network technology that facilitated point-to-point communication among the Processing elements providing scalability, high bandwidth and operating frequencies.

Design automation has been a vital area of research for at least three decades. Due to the intricacy involved in the system and the increased complexity of the fabrication techniques, chips' design has become a crucial task. Automating the activities involved in the design flow lessens human errors and saves time. For complex architectures like Network-on-chip, the area populated by each sub-element like buffers, interface units, power-related elements, and debug probes is paramount for analysis and planning on design reuse and improvements. A framework is created that visualises the area from the area data files and projects it into different technology nodes for given scaling factors. Debug infrastructure present in the Network-on-chip subsystem captures the time-stamp and hang state data. The debug data dumps are an enormous amount of data to be analysed and identify the cause of hangs; here, a framework is proposed that identifies if the debug register’s data corresponding to the hang state, captures the data lessening the efforts and time invested for debugging.

Scaling down the technology node adds complications like more leakage and more static power dissipation, high delays owing to reduced dimensions of wires of interconnect. The shift of focus is now towards integrating electrical interconnect with optical NOC for reducing the issues faced. Owing to the compatibility of Silicon-Photonics with the CMOS technology, ONOC router architecture is proposed, the design flow and its Performance analysis are discussed.

(This page is intentionally left blank)

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

NOC – Network-on-Chip

SOC – System-on-Chip

ML – Machine Learning

DS – Data Science

AI – Artificial Intelligence

RTL – register transfer level

BASH - Bourne-Again Shell

GPU – Graphical Processing Unit

DSP - Digital Signal Processor

PE – Processing Elements

CPU – Central Processing Unit

AMBA AXI – Advanced Microcontroller Bus Architecture Advanced eXtensible Interface

NI- Network Interface

CDC – Clock Domain Crossing

UPF – Unified Power Format

CLP – Conformal Low Power

STA – Static Timing Analysis

IP – Intellectual Property

ONOC - Optical Network-on-Chip

MRR – Microring resonator

Chapter 1 Introduction

1.1 Introduction

Network-on-chip has become a popular and endorsed network-based communication subsystem offering parallelism and scalability. It has masters and slaves connected upstream, and downstream side; and has interface units that convert generic protocols like the AMBA AXI to NoC specific protocol(packet-based transmission). The emerging trends in performance requirements like high latency, low power consumption, higher clock rates (typically in GHz), instruction and data parallelism in various devices like smartphones, videogaming, accelerating, artificial intelligence/machine learning, sizing, and shrinking transistor sizes have replaced the traditional bus architectures with NOCs. The NOC is also severely scaled down to meet the scaling and performance requirements. The upsurge in design complexity and miniaturisation, processes involved has been the originator of automation either in the design of NOC or for the performance, area and power analyses.

Data organisation, processing and analysis play a key role in gaining insight knowledge. Data processing is now a popular field due to advancements in fields like Data Science, Artificial Intelligence and Machine Learning[1]. Area analysis of NOC elements, i.e. debug, power, interface elements in technology, allows us to estimate the area consumed in other technology nodes through scaling factors. It helps in area estimation, optimising design and removal of redundant elements in future.

Automation of design flow reduces the manual effort and time spent on RTL Quality analysis. Design flow has been automated using Python 3.6 and BASH scripting to reduce manual efforts. Results have shown that the time consumed for RTL quality checking has reduced.

1.2. Network On-Chip basics

Network-on-chip communication infrastructure permits design reuse while preserving the performance of the chip. NOC provides the flexibility for various IP blocks and computational resources to process high data rates (>100Mbps) like in Video processors, GPUs, and DSP cores[3].

The components of NOC includes

1. PE- Processing elements could be Processor cores, CPU, memory, PCI, USB. These elements initiate transactions as required by broadcasting signals via the links.
2. Routers –Routers are connected to the processing elements and other routers in the network. These regulate the traffic flow using the routing logic.
3. NI- Network Interface bridges the link between the router and the core.
4. Links will join routers to the on-chip network[4].

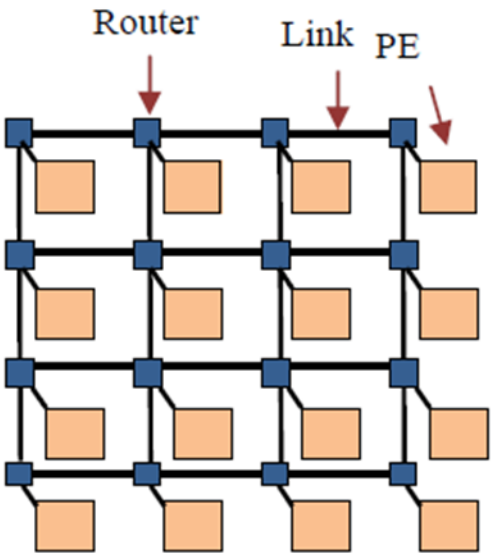

Figure 1.1 Network-on-chip connected to different processing elements

1.2.1 Optical Network on-chip

Optical Network-on-Chip comprises optical elements that could be incorporated into the chip for communication purposes. Figure 1.2 depicts the general optical link comprising the optical signal generation, routing of the signal to the corresponding destination and signal reception. At the generation unit, the electrical signal is altered into the optical signal and at the reception unit, the optical signal is converted back into a corresponding electrical signal. The optical components comprise optical filters, couplers, waveguides, lasers, amplifiers, photodetectors and modulators. Owing to the advancements in Silicon photonics, the integration of optical components on Network on-chip has been possible. Waveguides utilised could be bent, straight or cross-type. Straight waveguides have the most negligible losses whereas insertion losses are induced in bent and cross-type waveguides. Micro ring resonators are deployed for filtering the optical signal for routing purposes. These deliver the required switching and are most prevalent[34].

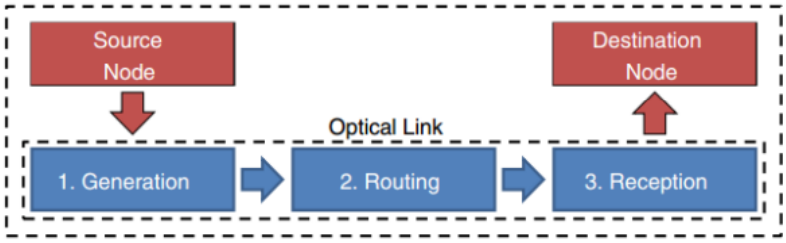

Figure 1.2 General optical interconnect link[34]

1.3. Area analysis

Scalability and design reuse of NOCs is imperative along with other cores and IPs as this has the chance to become a performance bottleneck if NOCs could not be scaled. Power consumption and area overhead are the two vital constraints in scaling the interconnect[11]. Scaling the area should not escalate the power consumption. NOC contains various units like the interface units, buffers, probes,

power-related elements. For widely harnessed devices like smartphones and laptops, the buffer size might get shoot up or fall based on the routing technique and method of switching[12]. A designer can get insight into reuse, modify the design of any element of the NOC and aim for high efficiency and performance. In the VLSI design flow, once an architecture design is complete, the design compiler is run and generates an area report of all the SOCs elements. It is a massive amount of data reporting each element's area, for example, of each NOC, making it difficult to analyse as a whole. Suppose there are b number of buffers in NOCa, p number of power elements, “i” the number of interface elements and “d” number of debugging elements, “o” for the others.

Area(b) = Σ Area of individual buffer in NOCa

Area(p) = Σ Area of individual power element in NOCa

Area(i) = Σ Area of individual interface units in NOCa

Area(d) = Σ Area of individual debug units in NOCa

Area(o) = Σ Area of individual other elements in NOCa

Total area = Area(b) + Area(p) + Area(i) + Area(d) + Area(o)

To overcome the time constraint and manual effort put in by the user who analyses the area, and created a framework to analyse the interconnect area.More about Area is described in the later chapters.

1.4. Network-on-chip - Debug

The intricacy involved in the communication among different cores facilitates the requirement for the debug-monitoring system at the communication network or Network-on-chip. On-chip monitoring is required even after abundant verification of the functionality to ensure the error-free design and avoid bugs reaching the consumer end. Specifically for improved debugging in SOC, transaction monitoring is done at NOC. These debug infrastructures could be complex owing to the manifestation of different clock domains within the NOC. The protocols used in NOC, AXI permitting OOO(Out of Order) execution, reinforces the transaction's complexity. Transactions emerging from a master with an ID may finish later than

the transaction generated most recently with a different ID. Transactions with different IDs can execute in any order; ordering restrictions arise when transactions are of the same ID. Interleaving and out-of-order features provided by the protocols for performance improvements in terms of throughput increase the chances of bugs[15].

1.5.Chip design flow

Accurate and clean Front-end RTL design decides the quality and performance of the chip. Design flow here encompasses the front-end flow dealing with performance and functionality. Flow also must comply with the design constraints to avoid redo of all the steps. [4]

First, the RTL for the design, coded in Verilog/System Verilog, can also support Object-Oriented programmings like C/C++ or System C.[5] The next step involves checking the RTL quality before it proceeds for the back-end design. These checks avoid future errors in synthesis and functional bugs. Some checks include Lint checks, CDC (clock domain crossing) checks, Synthesis checks, CLP(conformal-low power) checks.

1.5.1. Brief about the terms

- **Clock Domain Crossing(CDC)** – Clock domain crossing can happen between synchronous or asynchronous domains when data is transferred between two flip-flops. Data can go into a metastable state that is neither high nor low when data is captured while changing state. Static timing analysis (STA) is used to scrutinise synchronous domain crossings, but for asynchronous domain crossing, flop synchronisers have to be used. Figure 1.2 shows the possibility of a metastable state at the second rising edge of clkb[6].

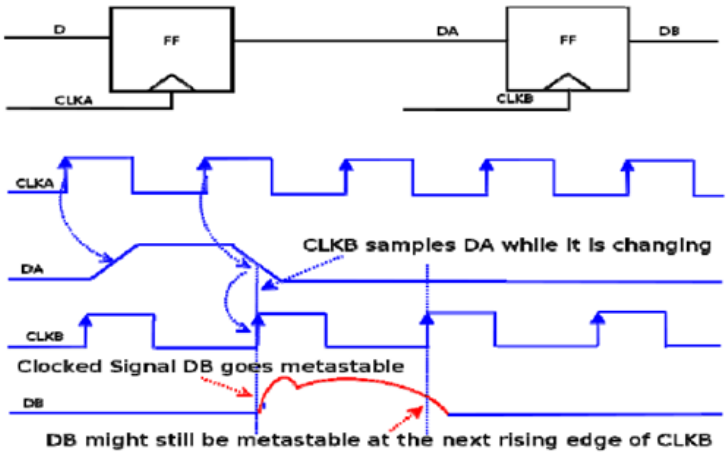

Figure 1.3 Metastability issues faced in asynchronous clock domains

- Lint Checks – Lint checks inspect the aspect of code with the recommended rules. It is basically like the syntax checking and static analysis of code to reduce errors due to functional bugs and synthesis errors.[7][8]
- Unified Power Format – UPF details the design's power intent for low power design[9]. Power supply definitions, isolation cell, memory retentionlevel-shifters, power supply requirements are specified. [10].
- Synthesis –Synthesis converts synthesisable Verilog code to technology-specific gate-level netlist.

These make sure that the RTL is clean and the design has less feasibility to have a bug. Once RTL is clean, it can move on to the back-end design.

1.6 Motivation

Since the automation of various steps involved in the design flow minimises the time consumed in recurring tasks involving manual efforts, mitigates the human-made errors, and escalates the robustness of the RTL delivery process in this thesis, the automation of designflow is done for flow checking and quality assessment. PPA(Power, performance and area) analyses are imperative for design optimisation and re-use. Next, a framework created for area analysis from the reports generated by the designcompiler and also scales to different technology nodes based on various scaling factors provided by the user. To curtail the time and efforts invested in data

collection from debug-dumps, debug automation framework is devised. Finally, we design an ONOC router that is compatible with the CMOS technology for communications.

1.7 Objective

The thesis goal is to incorporate automation to proliferate the robustness of the RTL delivery process, to exploit Data Science to devise frameworks for Area and debug dumps for analyses purposes.

- To automate design flow for RTL quality checks that encompasses updating many constraint files, checking in and checking out them.
- To minimise command-line usage for design flow.
- To capture the area consumed by NOC and its sub-elements from the region-wise CSV files procured from the design-compiler post-synthesis.
- To reduce the efforts in collecting the debuggability register's data and identifying invalid scenarios of the registers from the given Silicon dumps.
- To design an ONOC router and analyse its performance attributes.

1.8 Thesis organisation

The thesis is organized as illustrated below. The second chapter will discuss the vitality of data analysis in today's tech world—the role of Data Science and Automation in the VLSI domain. Then it shifts towards the Area framework creation that has simplified the accumulation of areanumbers for all the elements in the NOC. Next, we discuss the debug automation that has reduced the time taken for collecting the data related to hang-states to almost none. After that, we discuss the design flow automation that has automated the checks like CDC (clock domain crossings), lint checks. After that the design flow of the Network-on-Chip (optical)router that is of 4x4 configuration, non-blocking and minimised crossings is discussed.

Chapter 2 Automation and Data Science for Data Analysis

2.1. Data Science

Nowadays, the phrase Data Science is creating much buzz in the current technology world. The amount of data handled by organisations like Google, Amazon, Netflix is substantial. It is the essence of Data Science that allows data collection, modelling, analysis and visualisation.

Figure 2.1 explains how Data Science is the medial of Machine Learning, Statistics, and Data Analysis[13].Python, open-source libraries like pandas, NumPy are exploited by Data Scientists for playing with a large amount of data.

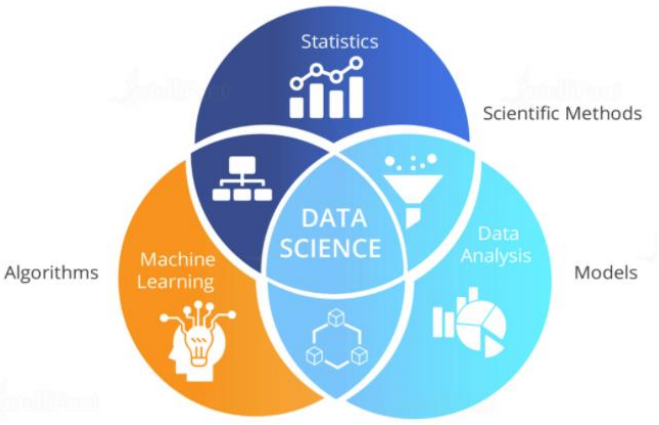

Figure 2.1 The essence of Data Science

2.2 Workflow in Data Science

Data Science workflow is the process extensively used by Analytics for data mining. The primary work goal of this work is understanding the data and modelling a solution to achieve the need. Figure 2.2 shows the steps involved in the workflow and the flow described below.

1. Business Understanding: It involves understanding the project or business objective, keeping in mind different factors like constraints, assumptions, and resources.

2. Data Understanding: Comprehension of the business process and sources involved in the problem, understanding the features of the acquired data, ensuring the data quality. One has to acquire technical knowledge, data, and tool flow.

3. Data Preparation: Ensuring the collected data is clean, formatted, filtered, and complete.

4. Modelling: It involves the selection of apt modelling techniques for the dataset. Once the modelling is complete, test to check the quality and validity.

5. Evaluation: In this phase, results will be generated and evaluated using the model. Based on the model's accuracy, it is decided whether to deploy it or modify it.

6. Deployment: Here, the validated model is deployed into the business needs.

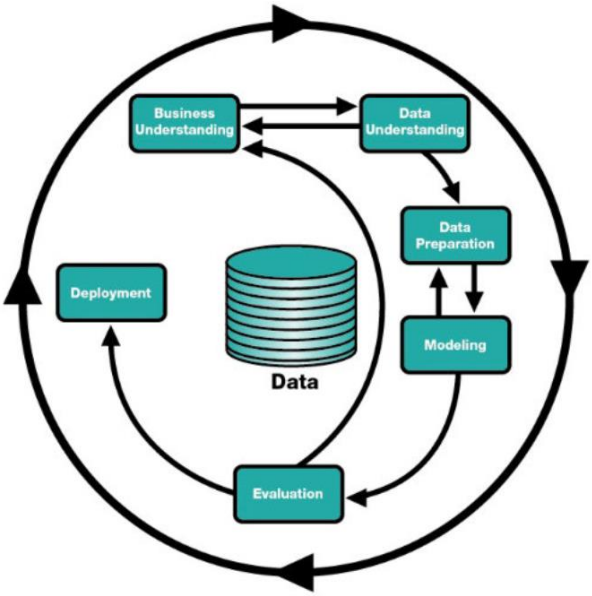

Figure 2.2 Data Science workflow

2.3 Role of Automation in the VLSI industry

Electronic gadgets incorporating the updated features and technology get launched in the market very frequently. The tremendous demand for cutting-edge features by consumers dwindled the lifespan of electronic gadgets. With the upsurge in the design complexity and enormous demand, the hustle on the chip manufacturers is very high for on-time product releases. The testing of the features augmented and the relative hardware improvements consume ample time. Automation streamlines any job; today, it is exploited in almost every field. Automation deployed in the chip design and the performance analysis abates the time taken for the process to complete. Consider the design flow, Area analysis and debug analysis discussed in chapter 1; automation can bring down all the time and efforts increasing the productivity and quality of the outcome. First, the problem is understood and we understand the tools used, the process, the type of data and organise the data. Then we do modelling of it, identifying the scope of automation[28].

2.3.1. Python for automation

Python has a large number of modules that can be exploited. Various libraries enable us to plot graphs, access web data, data analysis, web services helping to create models. Table 2.1 shows various libraries in Python and their uses, as shown in [17].

Library	Uses
Scipy, Numpy	Technical and Scientific computing
Pandas	Data aggregation and aggregation
Scikit-learn, Mlpy	Machine Learning
Scrapy, Beauifulsoup	Web scraping
statsmodels	Statisical Analysis
Keras, Theano, tensorflow	Deep learning
Genism, Ntlk	Text processing
Plotly, seaborn, bokeh, matplotlib	Visualization
networkx	Network visualization and analysis

Table 2.1 Python library and its uses[17]

The exploitation of languages for data science is not limited to Python; others include Perl, Java. Let us discuss a few about various libraries that are primarily used here.

2.3.1.1. Brief about the libraries

Various libraries in Python are deployed in handling data. For example, consider Selenium. This library enables the user to access and download data from web pages. It can be used to access various browsers like Internet Explorer, Google Chrome and Firefox and programmed in languages like C, C++, Python, Perl. Various sub-modules like Web driver allows in accessing and handling data inthe webpage. Graph plotting libraries like Matplotlib aids in data visualisation. XML and HTML webpage data can be effortlessly pulled out using the Beautiful Soup library reducing

days of work. Being most distinguished in ML and Data Science, Pandas is a notable library for data analysis and manipulation. It is exploited in tedious tasks and processes that are repetitive. The OS library is exploited in the creation of directories, navigating among various directories, access of different files in them and omission of directories.

2.3.2. **Perl for automation**

Perl is extensively exploited in the industry due to its unique features. It is employed in IC design as well as in fabrication. It permits linking C/C++ libraries so that they can be incorporated within the Perl script. Similar to Python, Perl is also a commanding programming language that is employed in data extraction, processing, filtering and presentation. Data visualisation and web scraping are also available with Perl. Extraction of essential data from large databases and data files can be streamlined with this language[29][30].

2.4 Background Study

Network on-chip, scalable communication infrastructure has superseded the conventional bus-based architecture imparting higher bandwidth, latency and parallelism[1]. RTL code of the interconnect either in Verilog/System Verilog is very complicated as the alteration has been towards multicore architectures, parallel communication and computing[32][5]. Performing various checks for RTL Quality assessment moderates the errors and bugs in the later processes[7]. Various checks incorporate clock domain crossing, lint checking, low power checks, synthesis checks[6][9][10]. Area efficiency is also as significant as the performance of interconnect. Design compiler delivers the area report post synthesis[33]; these reports are region wise area data that needs filtering. Powerful programming languages like Perl and Python are exploited for data filtering and visualisation purposes[29][30]. Networks that are debug-aware with cross-triggering are proposed [15]. These debug elements capture the data during an invalid state; it could be due to

time outs, decode error or slave errors. An outset of research towards integratingoptical interconnects/Network-on-Chip with conventional NOC is attributable to the higher bandwidth, latency and lower power consumption in optical interconnect. The

delay, static power consumed also scales up with scaling down the transistors[22][25]. So, many architectures are proposed that can handle the energy and performance requirements.

2.5 Automation and Framework creation

As described, Python and Perl are exploited for automation. A framework for area numbers extraction from data files dumped by the design compiler is developed. This framework gives visualised analyses and scales to various technology nodes exploiting Perl. Automated debug data collection with Python framework utilises web scraping to extract the data from HTML and filter the required data. This has dwindled many hours of effort to few hours. Design flow involves many tool commands, command-line usage, check-in and check-out of various files, accessing files and updating them. An automation framework has eliminated the problems. Using Python the coefficients for Network on-chip router design are visualised and computed. In the succeeding chapters, the framework created for area analyses, debug automation framework, automation of design flow and Network on-chip architecture (optical) will be discussed.

Chapter 3 Area Analysis for NOC

3.1. Floorplan of the chip

Network-on-chip is a communication network infrastructure used widely in a SOC. Figure 3.1 shows the floorplan of the chip at a higher level of abstraction. The regions in the chip are termed as Center, East, West, North, South for the soft macros. They primarily include hard and soft macros, which couldincorporate different IPs, processor cores, memories.Since interconnects connect various cores, they are spread all across the chip. So for a particular interconnect, its elements could be spread across various regions in the chip.

3.1.1 Macros

Macros are Intellectual Properties(IPs); these are designed to be picked up and placed by the designer.

These are of two types:

1. Hard Macros

These are peculiar to IP manufacturing technology optimised for area, powerand timing. RTL is non-modifiable; performance parametersare optimised andforeseeable, either in performance, area, power or timing. For example, memory.

2. Soft Macros

   These macros are modifiable, i.e., RTL can be changed by the designer and is synthesisable, leading to anomalous performance parameters in either performance, area, power or timing.Being portable, their IP protection is endangered.

   Few of the interconnects are soft macros.

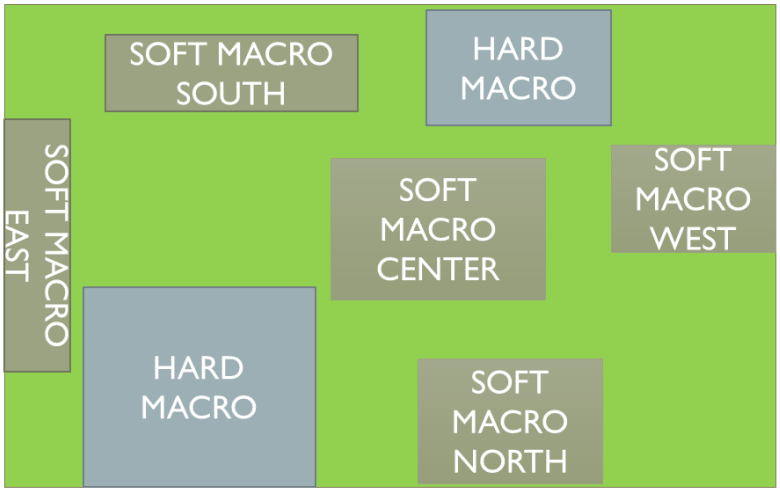

Figure 3.1 Floor Plan of the chip

3.2. Abstraction of elements in NOC for Area Analysis

NOC contains various sub-components; At lower-level abstraction, these include buffers for storing and forwarding packets for meeting the timing requirements. Power related elements like retention flop, gating elements. Interface units convert standard protocols like AMBA AXI to data packets as shown in figure 3.2.

Other elements include debug elements for debugging purposes that are present in the NOC near the interfaces. Figure 3.3 shows elements of the NOCs spread across thesoft macro-regions. Various NOCs are present, designed for meeting the performance requirements in bandwidth, latency, and throughput. All the NOCs, along with their elements, are analysed for the area. The dots represent various elements of the NOCs, black dots for buffers, orange for power-related elements, silver dots for interface and brown dots for debug related elements.

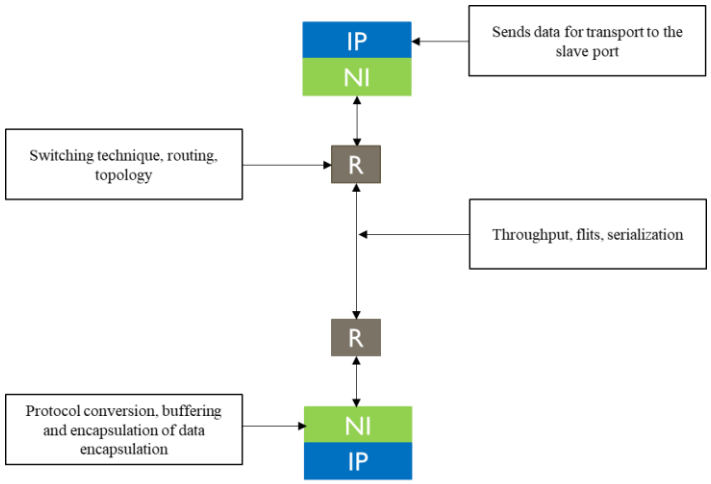

Figure 3.2 Model of NOC

3.3. Area CSV file generation

Design can be considered at various levels. SOC team synthesises the design at SOC level rather than at core or IP level. Multiple soft macros are grouped and synthesised at the regional level; soft macros comprise NOCs and other elements. The regional level synthesis produces region-wise area reports.

Area report for each region, i.e., South, East, North, East and Center regions area reports are dumped by the Design Compiler. The next step post-RTL design is the synthesis.

It includes the steps[14]

1. Analysis
2. Elaboration
3. Compilation

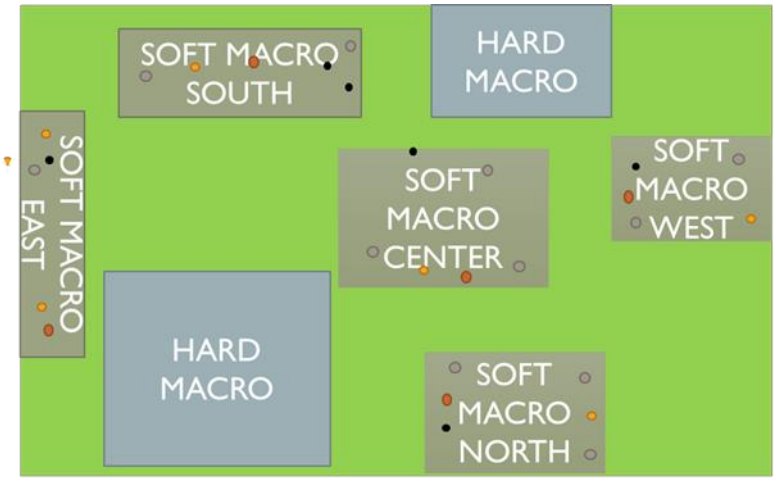

Figure 3.3 Various elements of NOC

Figure 3.4 shows the flow for area report generation for the design; the report gives the design's total area and area break-down modules. The report states macros, buffers, total combinational, non-combinational areas, and interconnects areas. Area CSV reports for a design get generated region-wise.

3.4. Problem – Statement

As the design compiler run at the SOC level generates area reports of all the region's elements, it became a cumbersome task for the designer to analyse the interconnect area of one NOC from different region CSV files with randomly distributed area numbers of different elements. Here three different NOC areas, as explained in section 1.2 are taken, i.e., NOCa, NOCb, NOCc

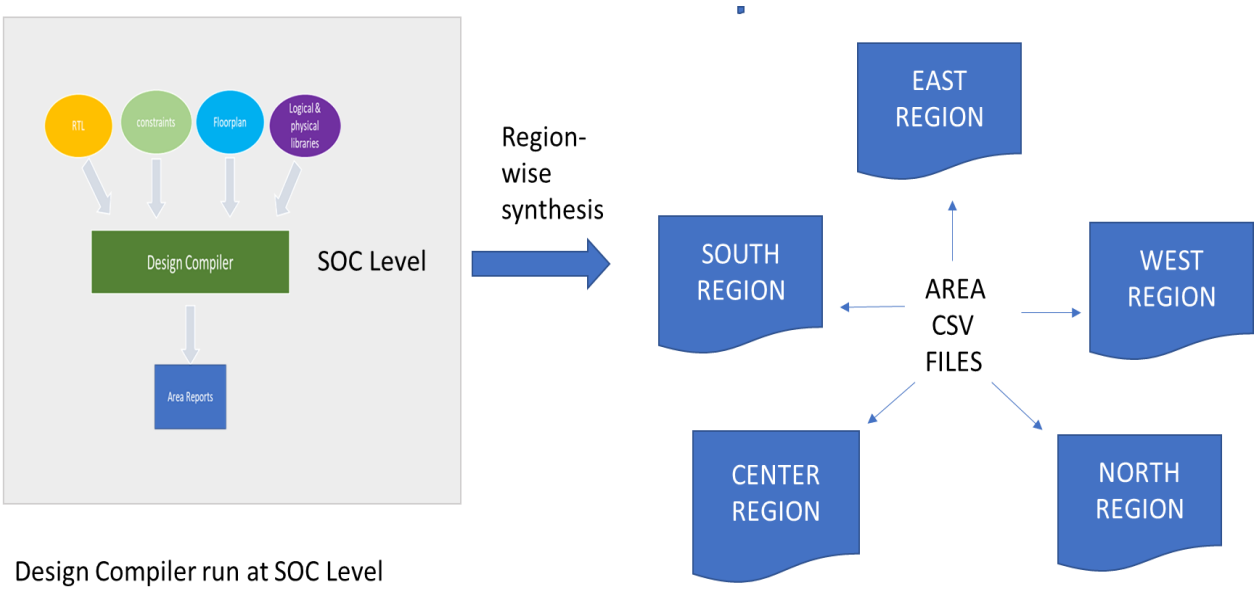

Figure 3.4 Flow for area report generation

3.4.1. Existing Flow

Figure 3.5 shows the existing flow for area number analysis where the CSV files dumped by the design compiler, reading the files and extracting the areas of each of the elements, i.e. for example, buffers. Each of the buffers of a particular NOC is identified from all the files for analysing the area numbers. Reiterate the same process for all other elements.

The above flow shows that it is a cumbersome task for any designer to spend so much time organising data from the area reports already generated by the design compiler. This problem shows potential for the exploitation of Automation and Data Science for data wrangling and data visualisation, making it facile to understand.

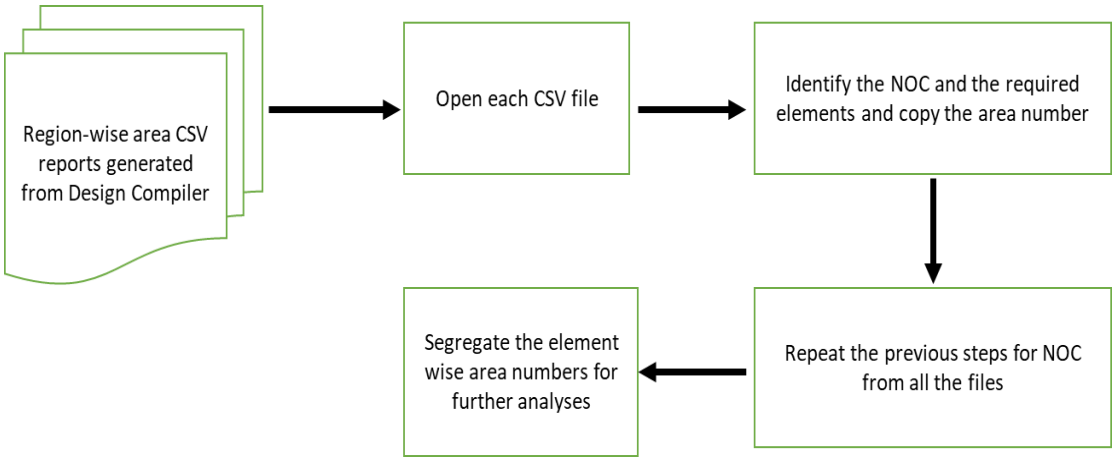

Figure 3.5 Existing flow for Area Analysis

3.5. Framework creation for Area Analysis Flow

It has a programmed framework using Perl and tcsh scripting that enables data visualisation and data wrangling. Figure 3.6 shows the flowchart of the framework. The features of the script are as follows

3.5.1. Configuration file-based execution

Whenever the script created is run, it reads the settings from the configuration file. The following switches are present in the file,

1. Project Path – Give the path of the Area CSV files of SOC in this setting.

2. Tech node – Give technology node of the design in this setting, e.g. 15nm.

3. Scaling nodes – Give the other technology nodes for getting scaled areas.

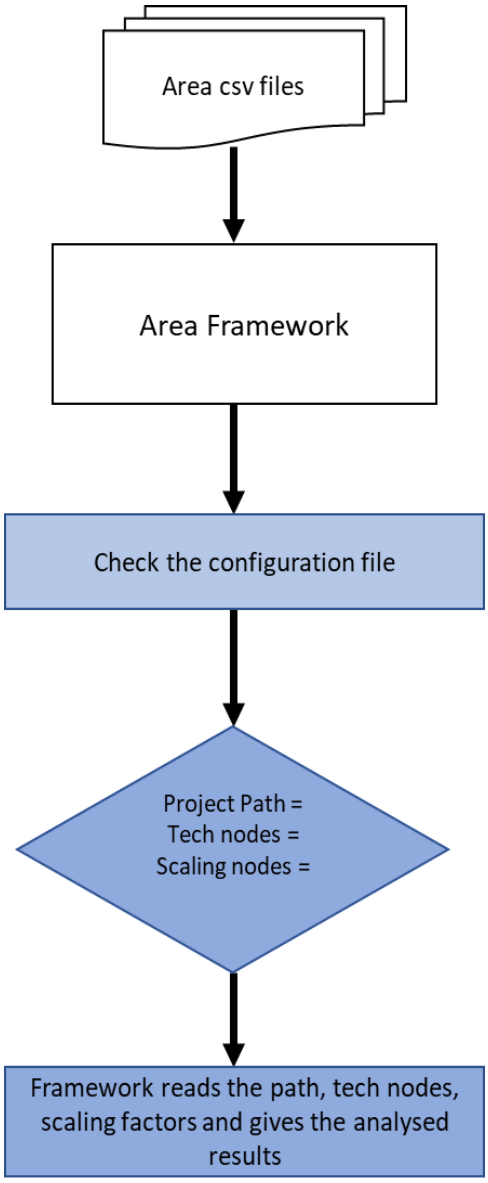

Figure 3.6 Flow chart showing Area Framework using Perl programming

Getting scaled area is user-definable. User can add scaling factors in the script.

3.5.2. Steps to use the framework

In the script's directory, open the .cfg file, i.e. the configuration settings file and Enter the settings as required. Close the file. Source the .sh file that reads the

Configuration file settings and gives the settings read as input to the Perl script. The script identifies all the interfaces, power, debug, buffer related elements, computes the total area occupied by these elements individually for each Network-on-chip and visualises the data.

Chapter 4 Automation of Debug Analysis

4.1. Introduction

Once the chip design is accomplished, testing the design is vital to check for flaws. As the number of cores and the communication complexity increases, so increases the complexity of NOC architecture; hence there is a high chance of the SOC going into an undetermined state. The chip has inbuilt debug features that will capture the erroneous states, enabling us to find the cause for Silicon going to hang state. These include trace data collection, triggering units and cross trigger interfaces [15].

In this chapter, debug for SOC is discussed, followed by the existing procedure for collecting the debug data, i.e. the debug register's hang states or undetermined state and then the exploitation of data science for framework creation for identifying the hang status of registers from the dumps.

4.2. Debug infrastructure overview in NOC

As discussed in the above section, the below figure 4.1 shows a debug aware NOC architecture. It has various cores like CPU, GPU, MODEM and PCI connected to the upstream side of the interconnect, whereas DDR and local memory are connected to the downstream side of the NOC.

The choice of placement of debug probes is designer specific; one can place the probes near the NIU for auditing transactions issued by the upstream core elements. The central debug unit will trigger the debug resisters to dump the data to the externally connected AMBA Trace bus for further analysis.

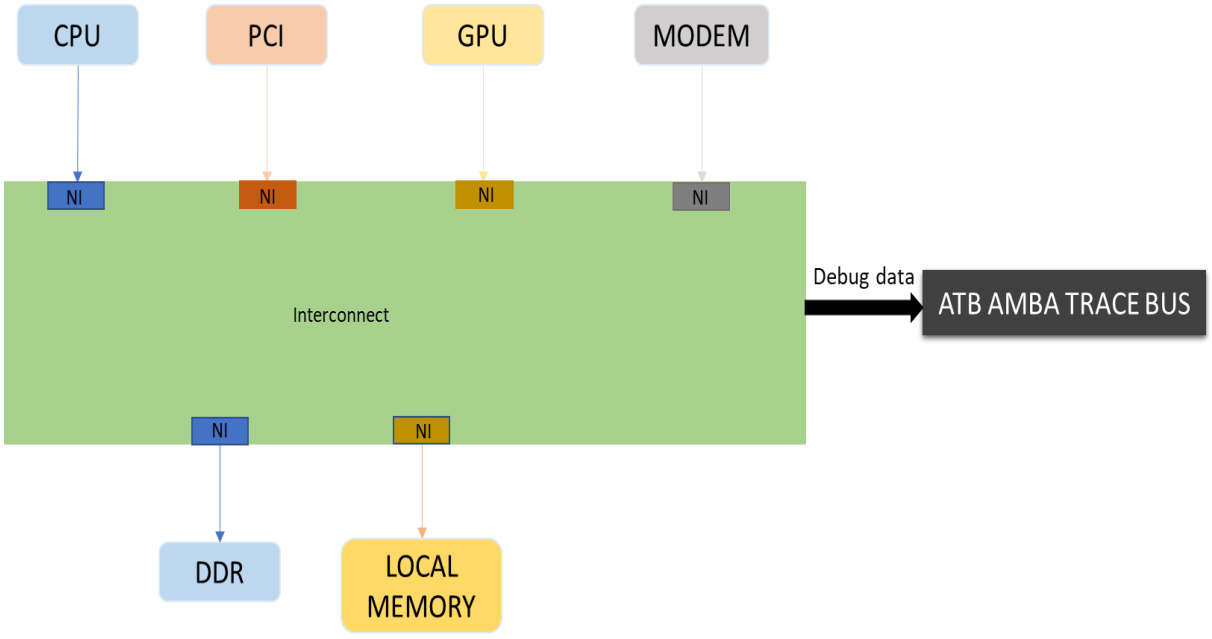

Figure 4.1 A debug–aware NOC architecture

4.3. Hang state Analysis

Hangs occur when a transaction between master and slave takes more time than usual. When a transaction takes a longer time than usual, a time-out counter present at the slave side gets timed out and generates an interrupt to notify that a hang has occurred.

4.3.1.Placing the debug registers

Network-on-Chip includes various registers for capturing the status of the ongoing traffic behaviour. These registers are scattered across multiple regions in the NOC, either proximate to the masters or slaves. Among the registers that encapsulate the traffic status, few of them are for debuggability. These registers for debug purposes are not functionally active until they get triggered. To comprehend an invalid scenario that eventuates in the network in a particular path, multiple registers

including some dedicated status registers for time outs, decode errors are scrutinized to understand it.The placement of debug-related elements in the NOC is vital because we cannot place the elements all across the chip. It can hardly consume 5 to 10 per cent of the average area of the NOC.Hence the placement of debug elements is also a crucial task in NOC design. Once the first few prototypes get fabricated, we run multiple-use cases and verify thesoftware settings and also check if the features added to the chip are satisfied. For example, Is the chip compatible with 5G technology? Does it meet the performance requirements?

During the performance check, we could encounter hangs in the system. There are several provisions like hang-reporting that creates an interrupt that states a hang has occurred. Then we need to provide a system reset to get back to a normal state from hang-state. The debug registers provide the hang information.

4.3.2.Causes of hang in NOC

There could be various reasons for hang in the system.

Non-responding targets:

Target may be non-responsive due to clock gating, i.e. clock to the target might be in off-state. There might be power collapse due to power gating on the target side, or the target might be in a bad state.

In the case of hangs, the response of the transaction by the slave could be one of those as mentioned in [16]

1. Slave Error –

   A slave error is a response to the transaction by the slave when there is a timeout condition in the slave, transaction of unsupported transfer size or if access to writing has endeavoured to read-only location[16].

2. Decode Error –

   Decode error sent as a response to when the destination address of a transaction

BRESP[1:0] RRESP[1:0]	Response	Meaning
b00	OKAY	Successful normal access
b01	EXOKAY	Successful exclusive access
b02	SLVERR	Sends error response to the master post access by the master
b03	DECERR	Decode error denotes no presence of slave with the specified address

Table 4.1Table showing SLVERR and DECERR responses in AXI protocol used for debugging purposes.

does not map to any slave downstream of the NOC. Default slave sends this response when this happens as it may end up searching for the slave with the address leading to stalling of further transactions in the NOC. Logging of all the decode errors responses by the default slave assists in debugging[16].

4.4. Existing post-debug Analysis procedure

This section describes the existing procedure to analyse silicon hang issues for debugging. All the silicon debug dumps dumped by the NOC units are available at an HTML link. The NOC team has the list of debug registers whose read-outs have to be analysed to identify which core is involved in the hang issue and the cause for it. The flowchart in figure 4.2 describes the conventional procedure followed.

4.4.1. Detailed steps

Here are the steps in the conventional hang cause detection procedure.

1. From the long list of read-outs present in the Html link, search for each register specified in the read-out register list.

2. So if there are 200 registers (can be more) search and copy readout values of all the registers. It is a very tedious job for the user to collect all the data.

3. Next, from the captured data analyse and identify which registers are in bad states.

4. Identify which data fields in the register represent the bad state.

5. Analyse all the registers identified for analysis and identify the cause of hang and report it to the software team for further analysis.

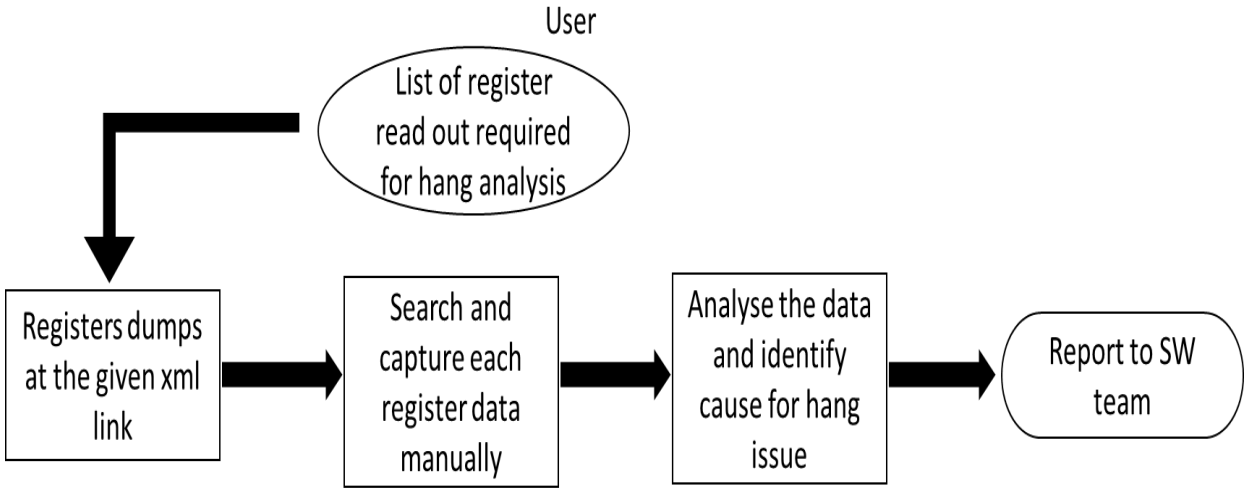

Figure 4.2 Flow chart showing existing procedure

4.5. Debug Framework

We created a framework for making the analysis process non- tedious, which eliminates all the above steps.It eliminates the time consumed by using the above discussed conventional procedure.

4.5.1. Exploiting Python for automation

We used the Python data framework for reducing the efforts of the user by automating all the steps discussed in section 4.4.1. It uses several Python libraries like requests, selenium, Beautiful soup, re to capture the data from the given Html link by reading the page's HTML code. It parses the data from the required tabs and saves it in text files in the input directory allowing the user to read them if necessary. The decoding logic identifies the wrong states.

4.5.2. Features of the debug automation framework

We have entirely automated all the steps followed in the conventional method and reduced all the efforts in the debugging process. We have included a configuration file based execution. We discuss below the parameter settings added in the configuration file

1. url_link :

   This setting takes the HTML or XML link as input. Generally, the software team provides the links for getting access to dumps of debug related registers.

2. Chrome_driver_path:

   This setting takes the chrome driver executable path as input. Specify the correct path for the errorless working of the framework.

3. Input_file_path:

   Specify the path to the input list of debug registers.

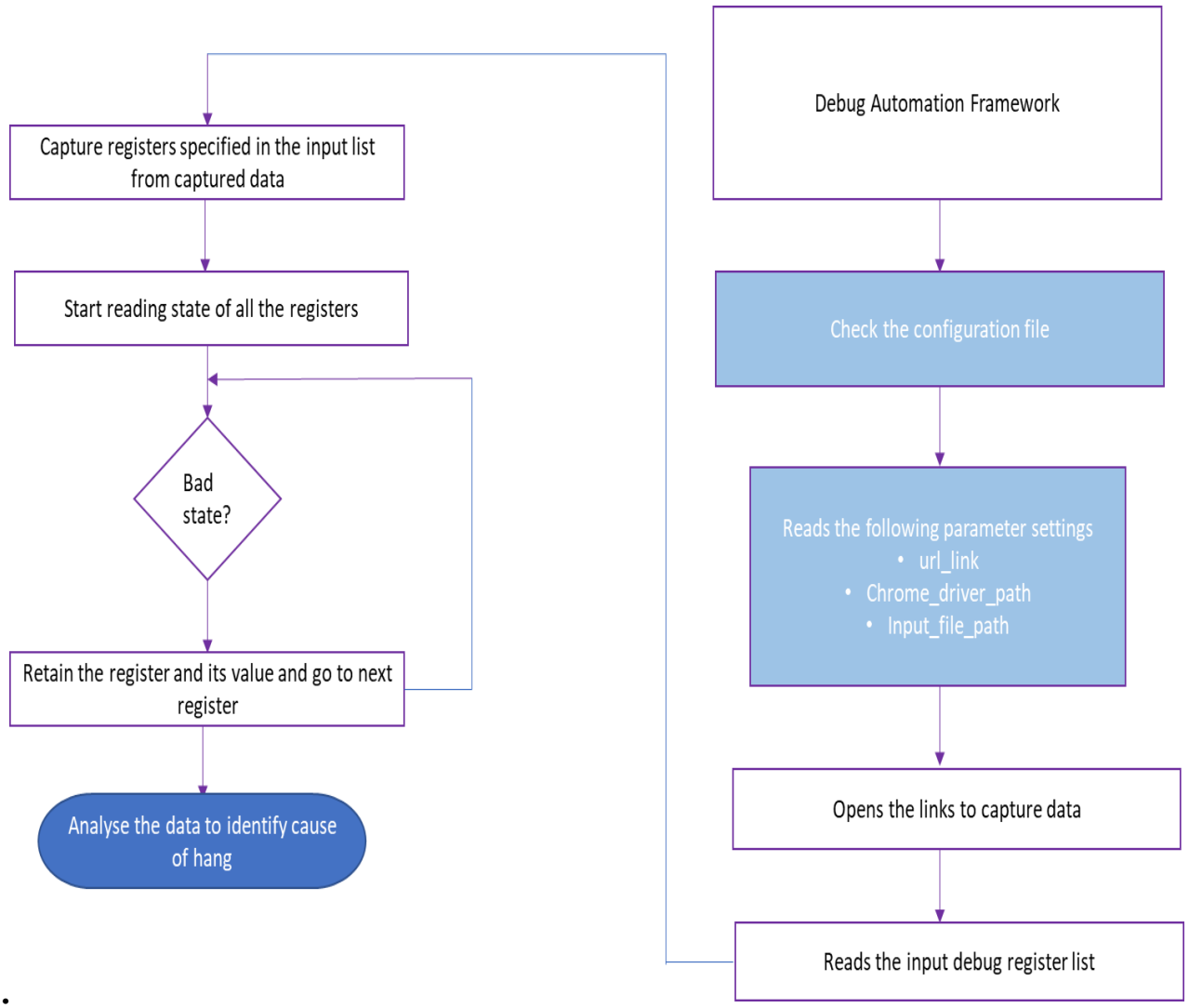

Figure 4.3 Automated framework for debug-hang analysis

Flow-chart above in figure 4.3 briefly describes the exploitation of the debug framework. Here we will elaborate on the steps used for debug-analysis in the net paragraph.

4.5.3. Accessing HTML code of webpage and generation of debug output

Create an input directory in the current working directory and put the input file list in it. Open the configuration file and enter the settings. Ensure that the correct path is entered to avoid the "WebDriverException" and termination of the run. Run the script; it opens the chrome browser and reads the HTML code of the provided webpage. We developed a decoding logic for capturing the required data. Once the data capturing data is done it is stored in a .txt file for reading later. Now all input debug register names are compared against the captured data and if there is a match in the register names, next it checks the state of the register. If the register's state is found to be ina bad or unacceptable statethe framework captures the name of the register and the values. It is left to the user to analyse the output file in the output directory to detect the cause of hang in the NOC.

4.5.4. Comparison between conventional and automated methods

- The time and efforts invested by the user in the conventional process are abated merely to furnishing the config file and running the framework.
- Easier identification of silicon hangs.

Chapter 5 Automation of Design Flow

5.1. Design Flow

After the NOC design, the RTL source code in Verilog/System Verilog must be verified to ensure that the design is clean, i.e., free from syntax errors, unsynthesisable constructs, ahead to avoid the long back end stages in the ASIC design cycles. It also minimises the functional bugs and eliminates the errors that occur during the synthesis. The design Flow described here verifies the front-end design and all the design steps involved in the design.

5.2. Existing Architecture

In section 1.3, we have given a brief introduction to design flow and section 1.3.1 described various checks verified in the front-end design for clean RTL design. In this section, we will describe the existing architecture for design flow.

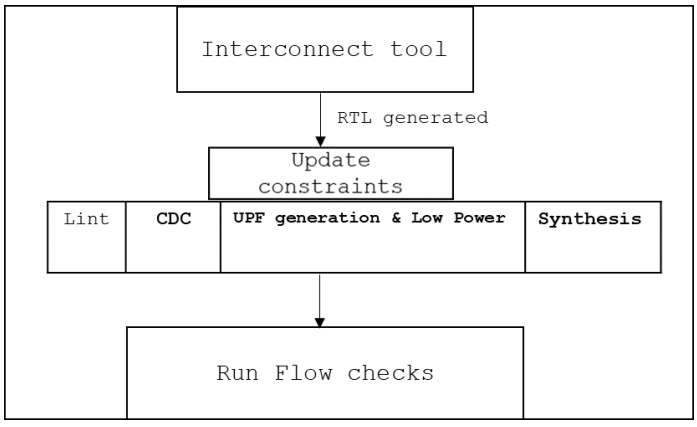

Figure 5.1 Existing design flow architecture

Once the design is complete, entirely inclusive of all changes we wanted to make to the source code be it the addition of new ports or clocks, we need to update all the design constraints manually for each of the flow check in their respective directories to ensure that it does not fail during the checks. Checking comprises running various tool-specific commands and accessing the required files and updating them. The check-ins and check-outs of all the required files should be taken care of before running flow checks.

5.2.1. Algorithm for the conventional mode of design flow

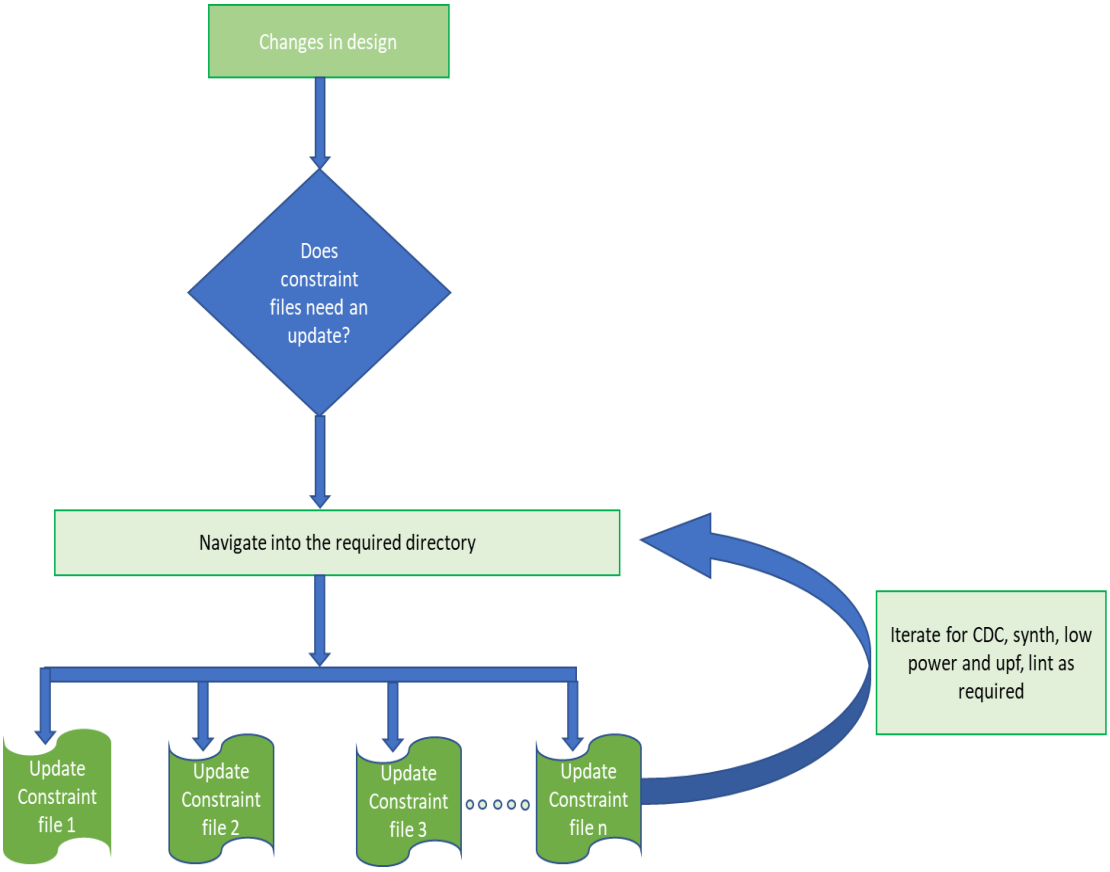

Figure 5.2 Flowchart showing iterative process of constraint file updating

In this section, we discuss the manual efforts incorporated by the designer to verify the front-end design flow. As discussed above post design, we run the interconnect tool, and the tool dumps the constraint files in the respective directories as specified by the designer. The user needs to update all the design constraints as required. Since today's NOC is a complex communication network, it involves many components; hence we can expect the number of constraints files. It employs much time and efforts to update each of the constraint files in the directories of the flow checks. Figure 5.2 shows the iterative process of constraint file update.

The next step followed by updating the constraint file is running the flow checks. Run all the commands for each of lint, CDC, UPF, synthesis. There is almost a total of 100 commands run on the command line to run the flow checks. Once all the flow checks are done the user can check each error log files in the output directory for each run. The user has to identify which run has failed and open the output log file to determine why a check has failed. Some of the reasons could be due to mismatch in port names or addresses, missing files. The user has to debug the causes of failure, fix it and repeat the above-discussed steps. Figure 5.3 shows the conventional method as discussed.

5.3. Automation of design flow

As we have identified the scope for automation owing to the lengthy steps involved in the command line usage, updating constraint files and the time taken to accomplish the complete task is also substantial and should be taken into account. Now we discuss the automation of design flow. Figure 5.4 shows the improved architecture; it has a framework around it that eliminates the hurdles discussed above. Figure 5.5 shows the flow chart for the improved architecture, which has proven to be effective.

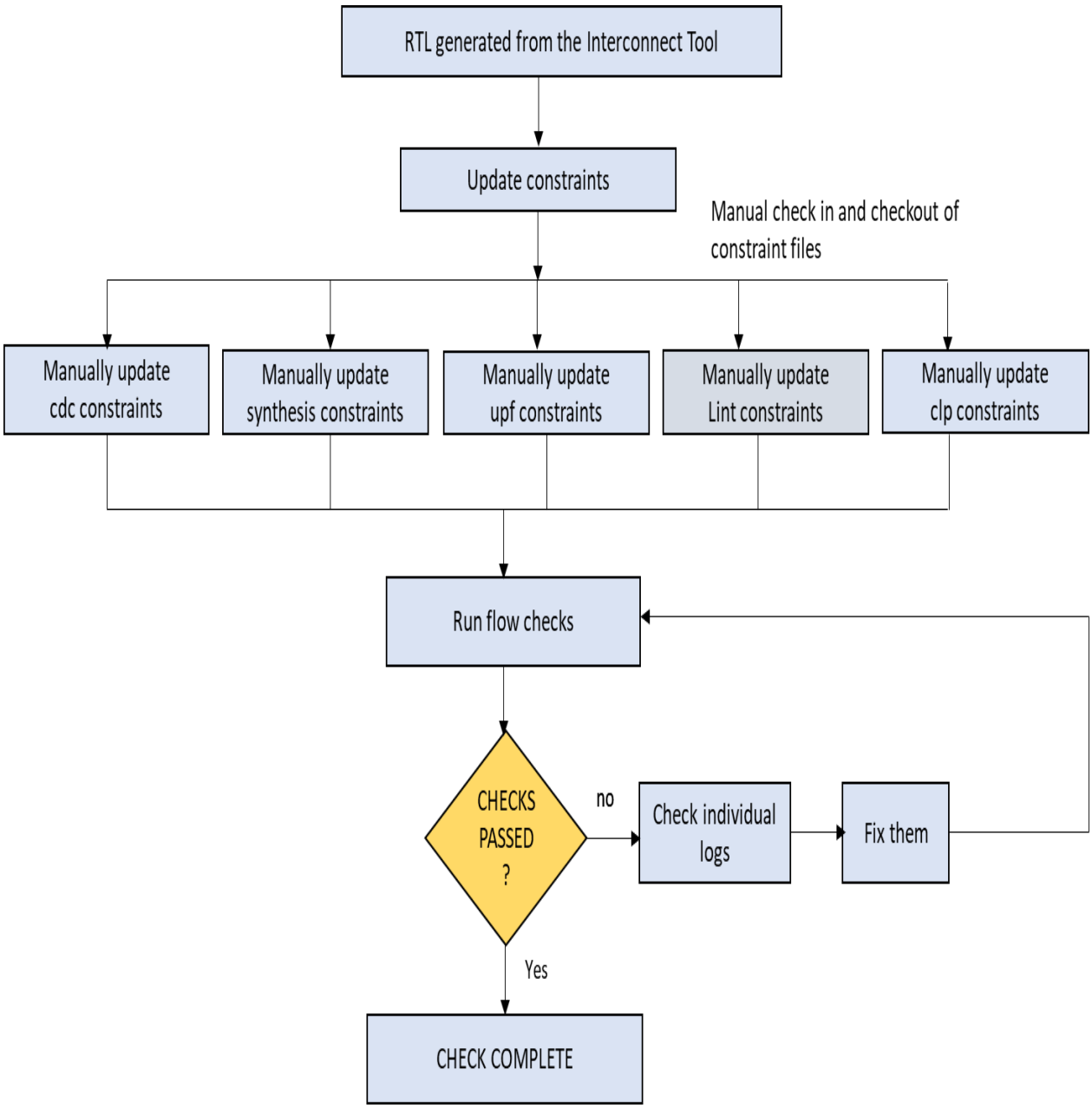

Figure 5.3 Flow chart showing steps followed in the conventional method

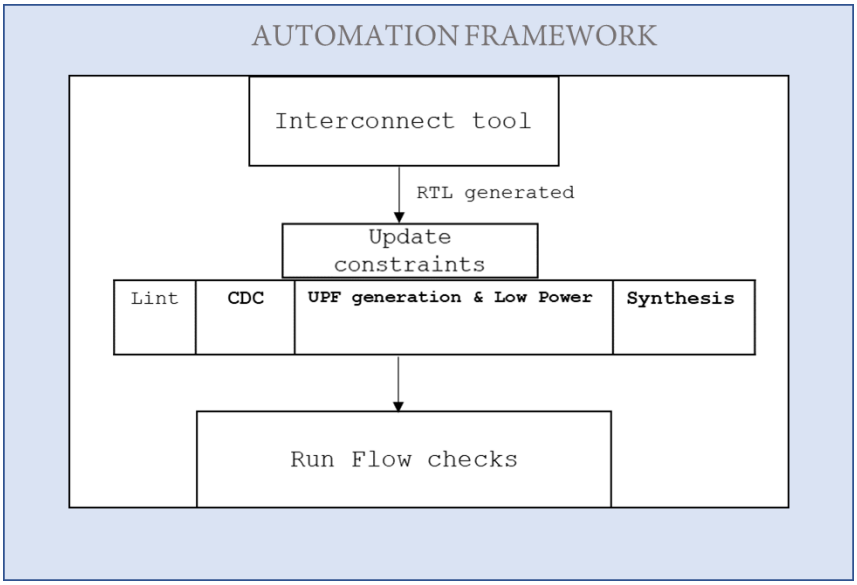

Figure 5.4 Improved Design Flow Architecture

5.3.1 Features of the framework

The framework coalesces all the commands that have the scope of being automated to simplify the process.

As a part of automation, we have added configuration file-based execution; a configuration file is a file that configures the settings or initial parameters for the program. The config file has the following settings added.

1. Flow:

   Flow setting can be either 0 or 1. The user can set it to 0 if he does not want to run the flow checks and one if he wants to run the checks.

2. Constraint_update:

   This field requires the constraints that need updating, for example, lint, CDC filled each parameter separated by a comma. Comma separation is mandatory as

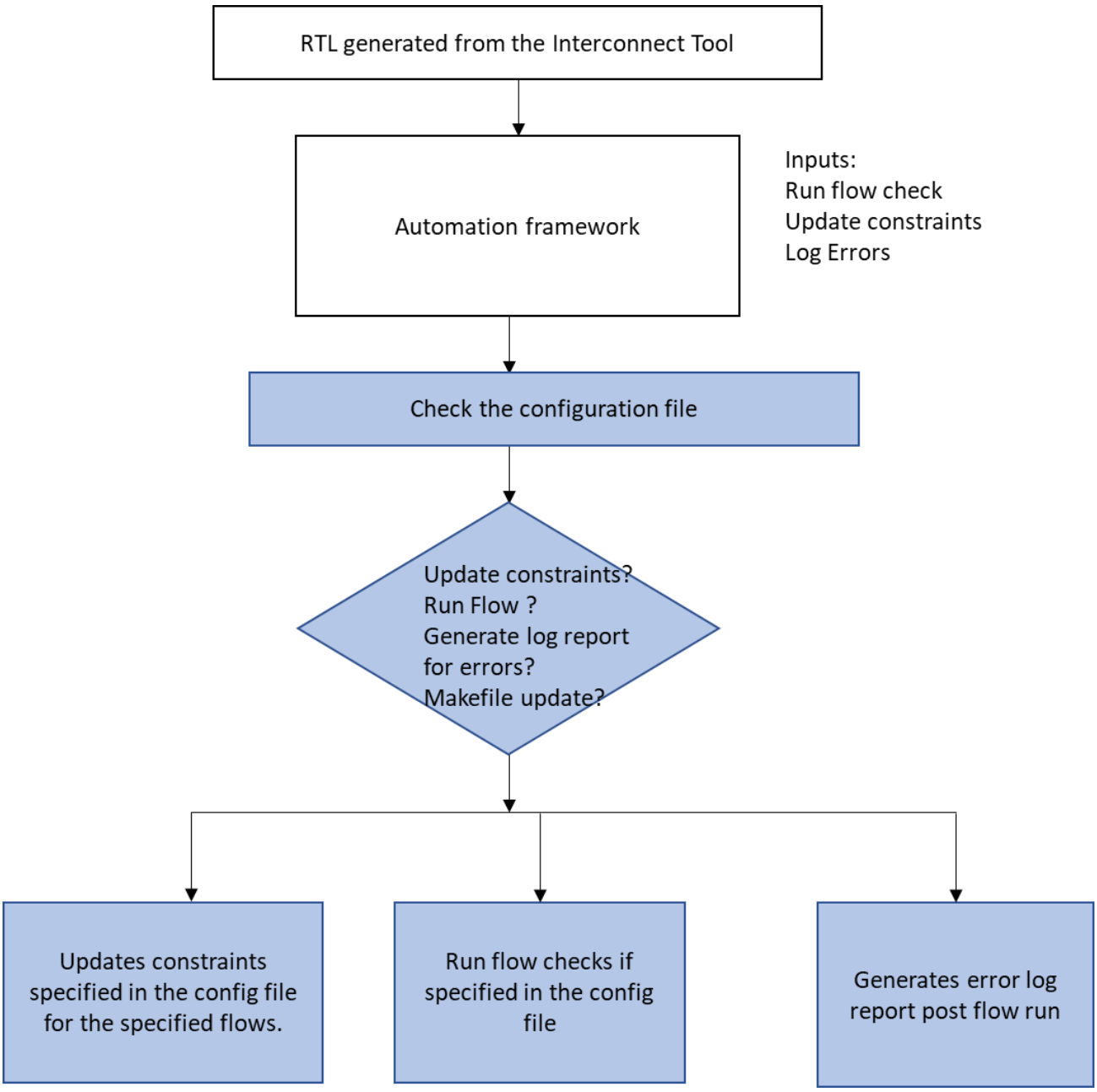

Figure 5.5 Design flow with automation framework

the framework identifies the settings for constraint individually if separated by commas.

3. Error log:
   This setting will allow the user to print all the errors from each of the log directories in one place post-flow run.

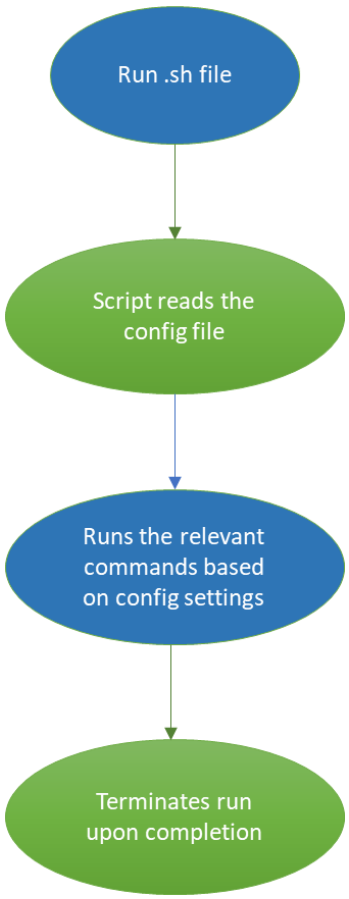

Figure 5.6 Config file-based script execution

Post running the script, it executes all the relevant commands for all the runs. These commands include dumping of constraint files by the interconnect tool by checking constraint_update configuration parameter, post update, it checks the flow switch and runs the checks and generates error reports. Configuration file-based execution is shown in figure 5.6.

5.3.2 Comparison between automated and conventional methods

This framework proved to be advantageous for the users since

- Time consumed in updating the constraint file is minimised.
- Reduced chances of human-made mistakes.
- We reduced command line usage to almost 10 per cent.
- The script takes care of check-ins and check-outs of files, eliminating the failures due to check-ins and checkouts.
- It has reduced the time and effort in error debugging.

Chapter 6 Analysis and Design Flow of ONOC architecture

6.1. Introduction

Assimilation of Optical interconnects in intra-chip communications for multi-core architectures comprising numerous cores, complex computing and communication schemes conform with the immense urge for performance requirements. There is extensive research in this domain since these can meet the demand for low power requirements, high latency and high bandwidth. Micro-ring resonator-based optical routers are prevalently exploited for ONOC architecture design. We will design the micro ring resonator adopting an SOI (Silicon on insulator) waveguide with a layer of Indium-Tin Oxide deposited on it. Indium – Tin Oxide facilitates the tuning of the ring up to ~2nm, eliminating thermal tuning of the resonator that not only costs much power and area overhead but also alters the properties of Silicon due to its high thermo-optic coefficient.

Silicon photonics facilitates the integration of optical components with CMOS as it is compatible with CMOS fabrication technology fulfilling the performance attributes; low power consumption and high bandwidth. We converse the electrically tunable MRR, design flow and performance of the proposed router in this chapter.

6.2. Design of ONOC Architecture

6.2.1. Router in Network-on-Chip

A router transmits the incoming message to the destination. Figure 6.1demonstrates conventional network-on-chip router architecture. The local output and input ports are connected to the core that is local to it. The remaining four ports direct the input signal from any other ports to the destination based on the control and routing logic[24].

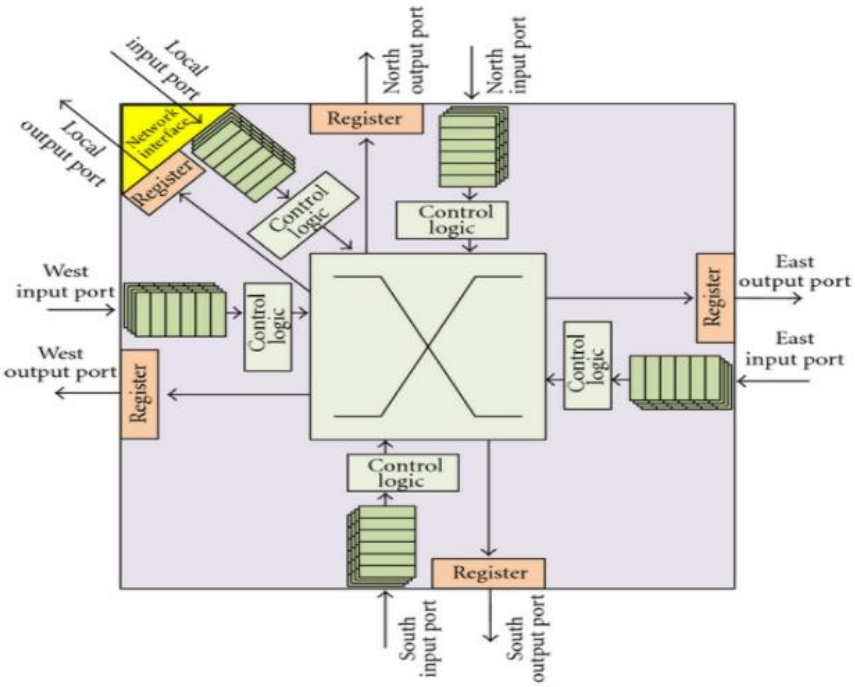

Figure 6.1 Network-on-chip router architecture[24]

The network interface unit transforms the incoming data packets into flits. The flit comprises header flit mentioning its destination, body flit, and tail flit.

6.2.2. Wavelength-routed Network-on-Chip

Wavelength routed optical networks involves deploying optical elements for overcoming the performance bottlenecks in electrical interconnect. Different wavelengths are allowed to transmit in the waveguide, improving the bandwidth and of the system. Some of the wavelength-routed architectures are discussed in this section.

6.2.2.1.Parallel and cross-MRR switches

MRR switches are broadly adopted in optical router design. They perchance are active or passive. Two types of MRRs employed in optical routers and NOCs are parallel and cross MRR switches[4]. Figure 6.2 shows both configurations used in optical architectures. During the off-state, i.e. the wavelength of the signal is out of

resonance, the input passes uninterruptedly to the through port; during the ON state, the signal is transmitted to the drop port. Ousting cross-switches with only parallel

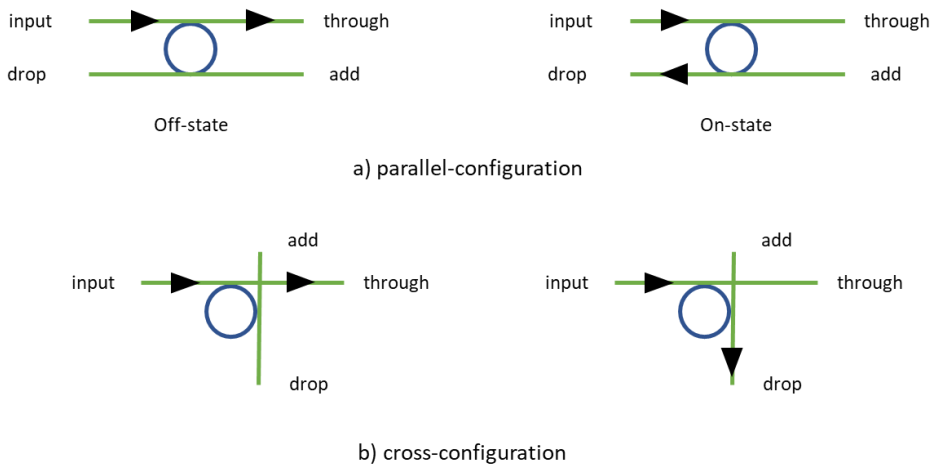

Figure 6.2 Cross and parallel MRR configuration

switches have diminished the insertion losses[4]. Wavelength division multiplexing employed in optical NOC enhances the bandwidth owing to signal transmission of distinct wavelengths on the same waveguide. The non-blocking, passive GWOR with nominal power loss and acceptable bandwidth is shown in figure 6.3 [25]. It employs 8 MRRs and two distinct wavelengths; all are in cross switch configuration.

The routing table is listed below. The signal from a source is inhibited from reaching its destination. I0 to O3, I3 to O0, I2 to O1 and I1 to O2, the signals traversing through these paths are uninterrupted by the ring resonators. Expect the sources and destinations in the straight path; all the signals arerouted by the ring resonator when the signals pass in their path. The designed optical router that is non-blocking employs parallel resonators is discussed in the next chapter.

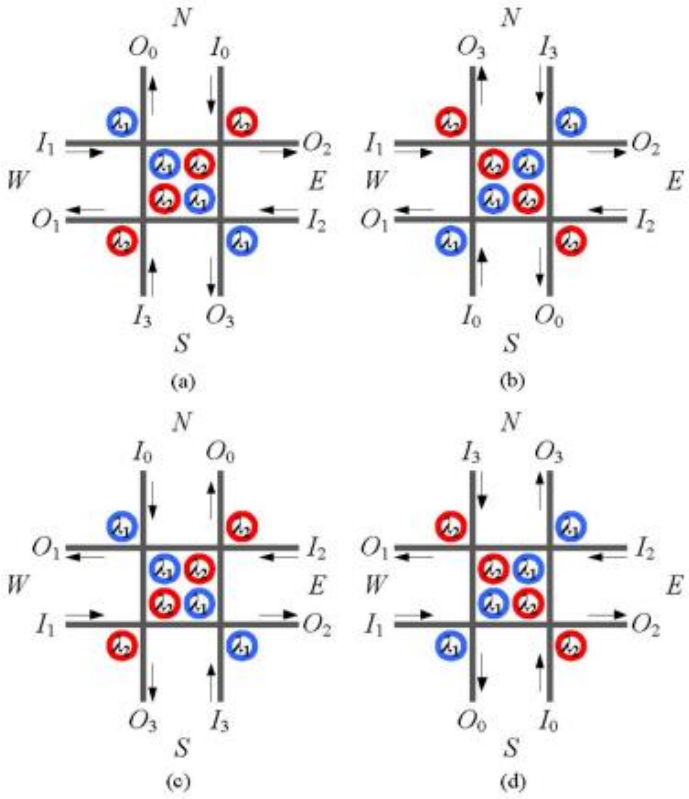

Figure 6.3 Generic Wavelength Optical Routed Architecture a)Type I b)Type II c) Type III d)Type IV[25]

	O_0	O_1	O_2	O_3
I_0	-	λ_1	λ_2	λ_3
I_1	λ_1	-	λ_3	λ_2
I_2	λ_2	λ_3	-	λ_1
I_3	λ_3	λ_2	λ_1	-

Table 6.1 Routing table for 4x4 GWOR

6.2.2.2.Power loss analysis in O-NOC

The insertion loss is maximal in the case of cross switch configuration. Insertion loss

can be surmounted by exploiting the add-drop resonator in parallel configuration[26]. Other losses in the router could be bending, propagation, losses due to interference and cross talk degrading the performance of the interconnect. Cross-talk noise ensues when wavelength division multiplexing is employed. It can be interchannel crosstalk between different wavelengths or intrachannel crosstalk between the same wavelength. Cross talk noise crumbles the SNR (signal to noise ratio), resulting in excess input laser power requirements[2].

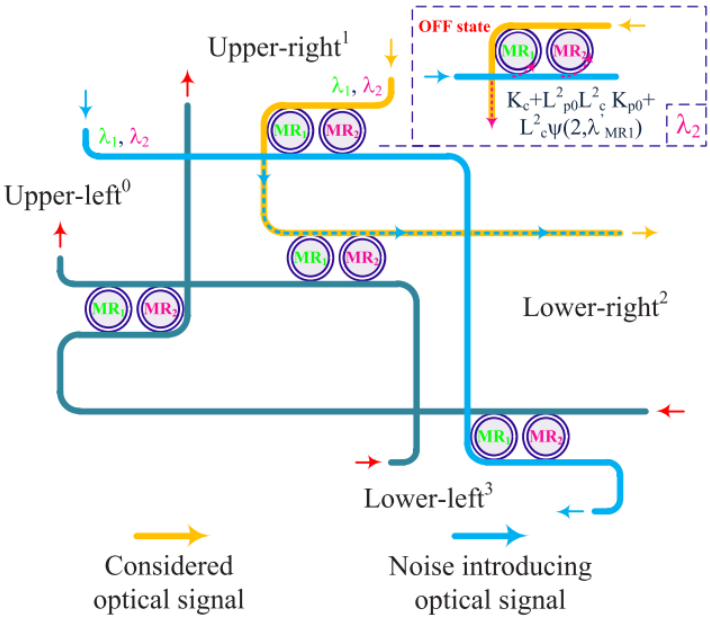

Figure 6.4 Noise due to interference and cross-talk, an example[22]

From figure 6.4, the signal traversing from the upper-right to the lower right and the signal from the upper left destined to the lower-left interfere. This interference is by the two MRRs on the upper right side whose resonance wavelengths match the wavelength of the signals travelling from the upper left. The cross switch at the upper right leads the signal from the upper-left (blue in colour) to the lower-right inducing

cross-talk noise. A 4x4 router with lesser cross-talk, non-blocking is proposed in chapter 7. The redundant MRRs sustains the SNR of the signal; lessens the cross-talk and interference. The optical router architecture can be extended to NxN configuration.

6.3. Design flow of MRR

The design of the ring resonator is discussed in this section. A microring resonator employed as an optical switch in optical NOC architectures widely uses the add-drop configuration to route the incoming signal to the required destination. Here we design an MRR calculating the cross-coupling(κ) and through-coupling(t) coefficients using the technique discussed in[18]. As shown in figure 6.5, the add-drop configuration of MRR is harnessed for filtering the input signal at resonance and transmits it tothe drop port; hence these can be exploited for routingsignals.Generally, when a wave transmits in the ring and establishes atotal phase shift of 2π or integral multiples of 2π, the ring cavity is in resonance and governed by equation (1).

Where m is the mode, R is the radius of the ring, neff is the effective refractive index of the waveguide, and λ is the wavelength.

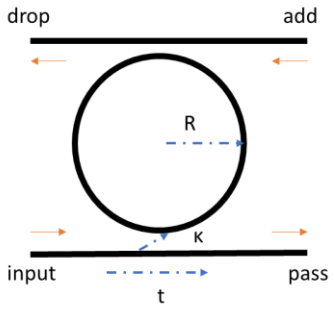

Figure 6.5 Add-drop ring resonator configuration

$m\lambda = 2\,\pi\,neff * R$ (1)

We began with the radius of the ring radius of 10,000nm, the width of the waveguide is 450nm and a resonant wavelength of 1550nm. The effective index of the waveguide is 2.36[18]. The cross-coupling coefficient is computed from equation 2. $a_E, a_O, \gamma_O, \gamma_E$ are the fitting parameters, the flexure function of the coupling region is B(x),d is the least gap amid the waveguide and the ring,x_E , $x_O$are the gaps for even and odd modes obtained by[19]

$$\kappa = \left(\sin\frac{\pi}{\lambda}\right)\left[\frac{a_E}{\gamma_E}e^{-}d\,B(x_E) + \frac{a_O}{\gamma_O}e^{-}\gamma_O d\,B(x_O)\right] \quad (2)$$

The flexure function is given by[19]

$B(x) = \sqrt{2\,\pi\,x}$ (3)

$$B(x_E) = \gamma_E\left(R + \frac{w}{2}\right) \quad (4)$$

$$B(x_O) = \gamma_O\left(R + \frac{w}{2}\right) \quad (5)$$

where x is the gap,R is the radius of the ring, w is the waveguide width. The flexure function establishes the correspondence between the varying gap from the waveguide and the ring's curvature as shown in figure 6.6.

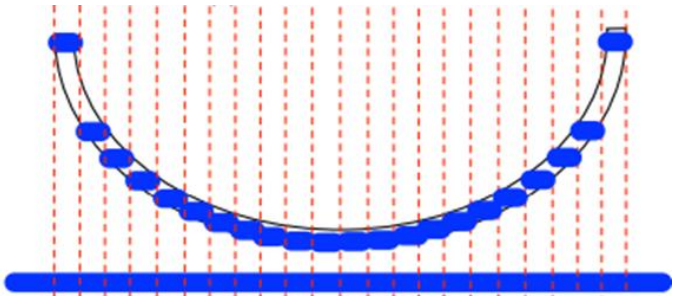

Figure 6.6 Non-uniform coupling between ring & straight waveguide[19]

The best-fit parameters for $a_E, a_O, \gamma_O, \gamma_E$,obtained by the method of non-linear curve fittingin Python. The obtained values are $a_E = 0.225, a_O = 0.010, \gamma_O = 0.011, \gamma_E = 0.002$.Using these parameters and d = 150nm, the value of κ = 0.2924 is computed from equation 2.

6.3.1.Computing the characteristics of the MRR

The intensity of the signal broadcasted through the ring can be characterised by the mathematical relationship described in [20]. The transmission of all-pass configuration is furnished as

$$T_n = \frac{a^2 - 2at\cos\emptyset + t^2}{1 - 2at\cos\emptyset + (at)^2} \qquad (6)$$

For the add-drop configuration considered here, the transmission via the drop port and the pass-port is demonstrated in the below equations.

$$T_p = \frac{t_2^2 - 2t_1t_2a + t_1^2}{1 - 2t_1t_2a\cos\emptyset + (t_1t_2a)^2} \qquad (7)$$

$$T_d = \frac{(1 - t_1^2)(1 - t_2^2)a}{1 - 2t_1t_2a\cos\emptyset + (t_1t_2a)^2} \qquad (8)$$

Here, the through coefficients $t_1 \, and \, t_2$ are considered to be of identical value. It concludes that the parameters opted will provide acceptable transmission characteristics.

6.4. ITO – based MRR

Indium-Tin-Oxide, a transparent and conducting element, furnishes the confinement of light at the ENZ(Epsilon-near-zero) region. Electrical or optical excitation of the

material persuades an alteration in carriers (accumulating or depleting) enables the tuning of the real part of permittivity. A thin layer of Indium Tin oxide ITO (20nm) is deposited on the top of Silicon dioxide to act as the gate electrode enabling the electrical tuning of the resonator. When a bias is applied to the ring, carriers accumulate at the ITO- SiO2 and SiO2-Si interfaces owing to the plasma-dispersion effect. According to the Drude model[27]

$$\varepsilon_r = \varepsilon_\infty - \frac{\omega_p^2}{[\omega(\omega+i\tau)]} \quad (4)$$

The high-frequency permittivity is *symbolized by* ε_∞ , relative permittivity by ε_r, the plasma dispersion frequency by ω_p and collision frequency by τ

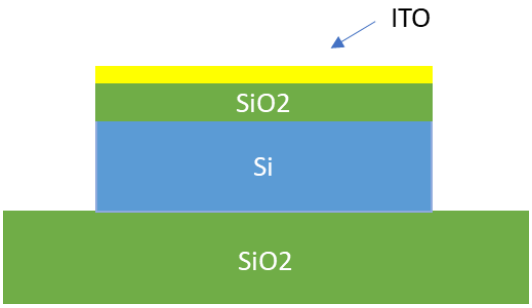

Figure 6.7 Figure showing ITO as the gate electrode for MRR used for electrical tuning

The carrier change induces a change in permittivity, which induces further change in the refractive index, tuning its wavelength.

$$\omega_p^2 = \frac{N_c q^2}{\varepsilon_0 m^*} \quad (5)$$

N_c symbolises the carrier density, m^* symbolises the effective mass and q, the electrical charge. The below relationship furnishes the alteration in refractive index with the modification of the effective refractive index due to carrier change

$$\Delta\lambda = \Delta n_{eff} \frac{\lambda}{n_{eff}} \quad (6)$$

Hence, by effectively modulating the carrier density by employing, bias the resonator can be tuned.

Chapter 7 Results and discussion

In all the preceding chapters, the design flow for RTL delivery, the existing process for capturing area numbers of Network-on-chip and its elements, the procedure for getting the debug registers and identification of invalid scenarios in the register dumps and the scope for automation framework creation was discussed. In this chapter, we analyse the improvements achieved in terms of reduction in manual efforts and time-taken in command-line usage, file access, debug register data dumps capture, region-wise area CSV file reading for identifying and getting the area numbers for different interconnects as required. This chapter also describes the design flow of O-NOC router architecture and its analysis.

7.1.Area Framework

As described in chapter 3, the area CSV files from the post-synthesis are region wise that contains area data of many elements along with NoC. Here we discuss the improvements attained through the framework.

7.1.1. Configuration settings file

A configuration file is an input file that configures the process parameters.

```
######Configuration file for area#########################
# Project = path to the area CSV reports                  #
# Tech_node = Specify the technology node for the  project #
# Scaling_nodes = Specify the nodes to scale              #
#                                                         #
# *Specify the scaling factors in the script if required*  #
###########################################################


Project=
Tech_node=
Scaling_nodes=
```

Figure 7.1 Area configuration file

Figure 7.1 shows the area configuration file,the path to the CSV files entered in the path. Tech_node parameter takes the current technology node. Scaling_nodes parameter takes comma-separated technology nodes numbers.

7.1.2.Data Visualisation

The framework visualises the total area numbers of sub-elements for analysis eliminating the tedious work of scanning all the files and noting each area of each of the buffers, debug probes, interface units and adding them to get the total area. Figure 7.2 shows the visualised total area for analysis.

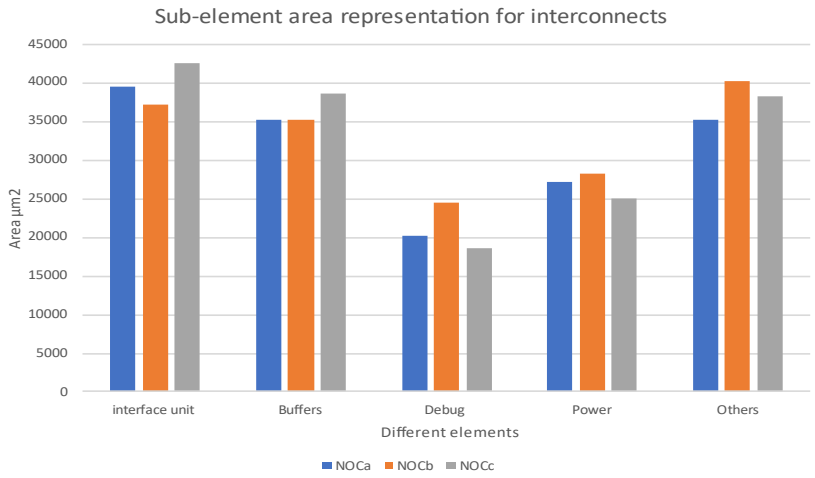

Figure 7.2 Area of sub-elements of different NoCs

The figure 7.3 shows the pie-charts for the percentage of area occupied by NoCa, NoCb and NoCc.

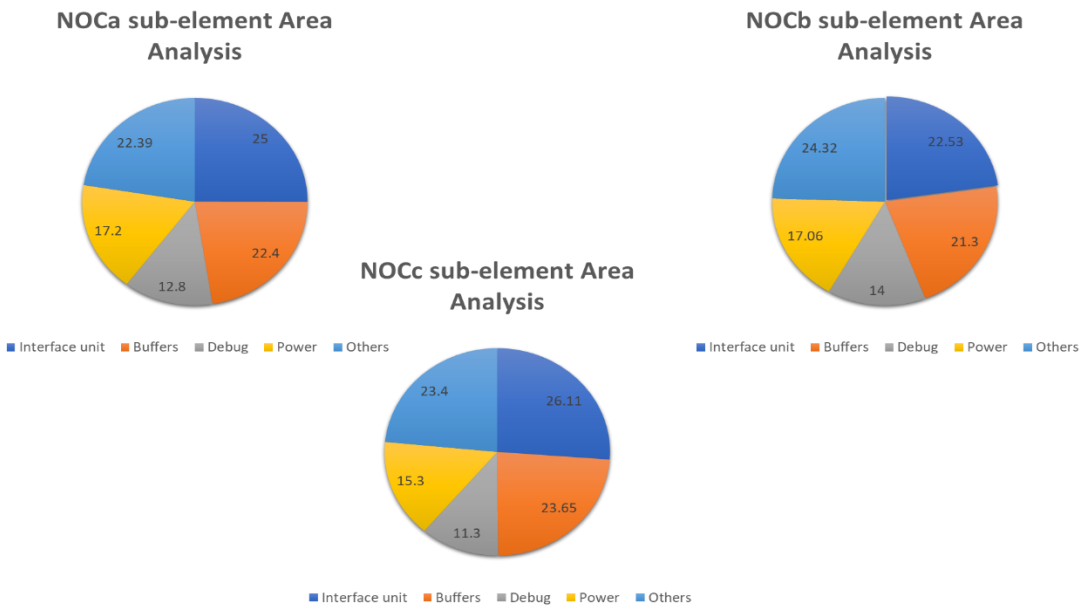

Figure 7.3 Pie-chart representation of the percentage of area occupied by sub-elements in Network-on-chip

7.1.3. Scaling of area

The framework scales the area of each sub-element to different technology nodes. Figure 7.4 shows the scaled total area of the three NoCs. The framework can also be supplemented with other technology nodes.

7.1.4. Improved vs Existing process

The existing process involved much tedious task of reading each CSV file, the framework proved to be potent.

- Capture each sub-elements total area and records the total number of each sub-element and their area contribution.

- Visualises the area numbers for a better understanding of trends in different technology nodes and among different interconnects.

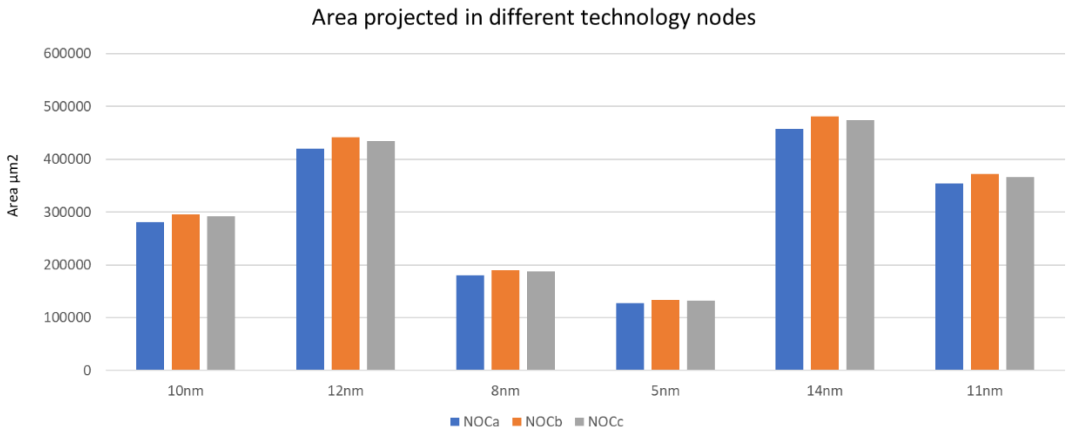

Figure 7.4 Scaled total area of NoCa, NoCb and NoCc in different technology nodes

7.2. Debug automation framework

The automated debug framework discussed in chapter 4 has efficiently eliminated the reduced capturing and decoding of the invalid states for hang analysis.

7.2.1. Configuration settings file

A configuration file has been included as shown in the figure 7.5; url_link takes the HTML webpage link as input, the chrome driver path for automated opening of the Web browser and capturing the required dump data. The input file path for the list of Network-on chip debug registers.

```
##################config file for debug analysis#############################
# Please provide the settings as described below                            #
#                                                                           #
#url_link               --- specify the xml or html link provided by the software team    #
#chrome_driver_path  --- specify the path to the chromedriver.exe          #
#input_file_path        --- specify the path to the input debug register file          #
#############################################################################

url_link =
chrome_driver_path=
input_file_path =
```

Figure 7.5 Design flow configuration file

7.2.2. Framework over the conventional procedure

The framework created using Python. Different modules like Selenium, requests and Beautiful soup. It parses the web page and filters the debug data. It converts the HTML code into text files and compares it with the input register list. The data is further filtered and decoded. Figure 7.6 shows the snippet of the code for the framework.

- It is 1000+ lines of code and captures only those debug registers data that are in an invalid state.
- Easier identification of the cause of Silicon hangs.
- The significance of port names has to be decoded by the designer.

Table 7.1 shows the comparison of the existing process of debug-data capture for analysis with the framework. The state of approximately 220 registers from the input listhas to be checked against the dumped data and captured to analyse one dump. The number of dumps may be greater than one for a chip when checked multiple times. Time taken to capture and decode been cut down from greater than 12 hours to less than 6 hours.

```
from selenium import webdriver
from selenium.webdriver.common.by import By
from selenium.webdriver.support.ui import WebDriverWait
from selenium.webdriver.support import expected_conditions as EC
import re
import xlsxwriter
import requests
from bs4 import BeautifulSoup
import sys
import os

workdir = os.getcwd()
config_file = open('debug config file','r')
config_data =[]
xml_url = ""
driver_xml_path = ""
loc = ""

for line in config_file:
    if(re.search("url_link",line)):
        x = line.split("=")
        xml_url = x[1]
    elif(re.search("chrome_driver_path",line)):
        x = line.split("=")
        driver_xml_path = x[1].strip()
    elif(re.search("input_file_path", line)):
        x = line.split("=")
        loc = x[1].strip()

input_dir = workdir + "\input"
os.chdir(workdir+"\input")
#### access the given input html link ####
driver_xml_path.get(xml_url)
element = WebDriverWait(driver_xml_path, 30).until(EC.presence_of_element_loc
source =  driver_xml_path.page_source
soup = BeautifulSoup(source,"html.parser")
tabs_name = soup.find_all('span',class_="html-attribute-value")
### open the webpage from the given xml link ##########
for i in range(0,len(tabs_name)):
    z = re.search("[redacted]*html",str(tabs_name[i].text))
    if z:
```

Figure 7.6 Snippet of code

Feature	**Without framework**	**With framework**
Debug registers data to capture	220	automated
Time taken to decode(for one dump)	>12hrs	<6hrs

Table 7.1 Comparison with and without framework

7.3. Automated Design Flow

7.3.1. Configuration settings file

As a part of automating the design flow, a configuration file, as shown in figure 7.7 is added; the automation script interprets all the settings and runs the commands accordingly.

```
#### Configuration file ############################
## flow = 0 or 1 runs flows checks                 ##
## constraint update = cdc,lint,upf,synthesis,clp  ##
## Error log = yes or no                           ##
##
####################################################


flow = 1
constraint update = cdc,lint,synthesis
Error log = yes
Makefile_update = no
```

Figure 7.7 Design flow configuration file

7.3.2. Execution time analysis

The comparison for the time incorporated for flow run, updating constraint file, and total run time for the existing flow and the improved architecture is shown in table7.2 and figure 7.8. Post design changes the constraint files update take about one hour to update manually is reduced to 5 minutes. Flow run time is tool-specific. Hence it is constant. So, the total run time for the design flow is reduced by approximately 68%. The figure 7.9 shows the sample display of the result at the run completion. The user has to check the error log files, decode and fix the bugs as required.

FEATURE	EXISTING FLOW	IMPROVED ARCHITECTURE
Flow Run Time(Tool specific)	20mins	20mins
Constraint update	1 hr	5 mins
Total run time	1hr + 20mins	20mins+5mins

Table 7.2 Design flow configuration file

Figure 7.8 Comparison of average execution time in existing and improved architecture

cdc	Lint	Upf	Synthesis	clp
F	P	F	P	F

P:PASS F:FAIL E:EXEMPTED

Figure 7.9 Sample flow result

7.3.3. Reduced number of file access and command line usage

The automation framework lessens the number of constraint files to updatemanually.

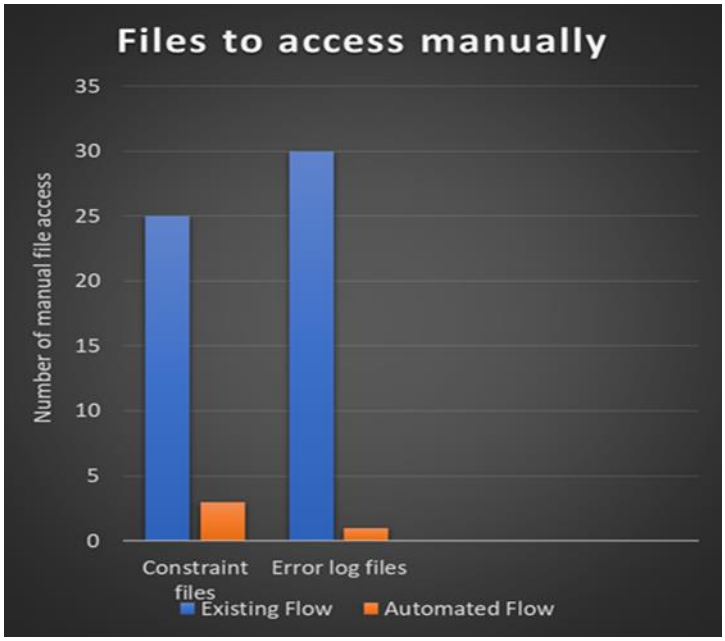

Figure 7.10 Number of file access in existing and automated flow

The constraint update setting in the config file enables the automatic update of constraint files of specified parameters. Also, post flow run completion if the output displays any failures, different log files must be checked for debugging. A post-run

log file has been added that captures errors from all the file (approximately 30) and stores it in the run directory. Figure 7.10 shows the number of file access for constraint file update and log files.

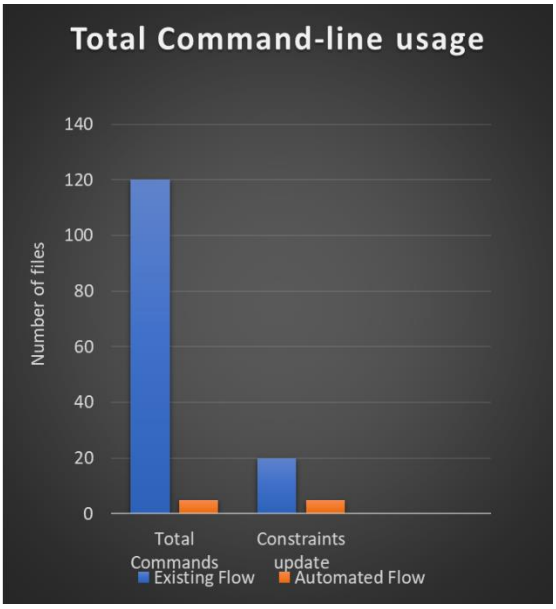

Figure 7.11Command-line usage comparison in existing and automated flow

Figure 7.11 depicts the command-line usage for the flow run; in total, there are about 120 commands till the termination of the run. Automation has reduced the number of commands to be input by the user. Although the frameworkupdates the constraint files, there might be some files that require a manual update.

7.3.4. Improved design flow vs Existing flow

The improvements added through the automation framework has provided the following

- Easy debugging
- Reduced command-line usage for constraint and flow execution
- Improved robustness

7.4. Design Flow ofO-NoC router architecture

7.4.1. 4x4 Network on-chip router

The proposed router shown in figure 7.12 is a 4x4 non-blocking router. It encompasses 8 MRRs and six waveguide crossings. The input and output ports are assigned from $I_0 - O_0$ to $I_4 - O_4$ in the anti-clockwise direction. This router employs only parallel resonators with no cross-MRRs scaling down the insertion losses. Since this router uses two different wavelengths recursively to enable parallel communication, it enables the communication between any pair of input-output ports concurrently, provided the output port is free. We can see in figure 7.12 that some of the MRRs are redundant; for example, MRR with λ2 between I1 -> O0 and I2 -> O3 are redundant but these are not evicted to avoid incoherent cross-talk and interference.

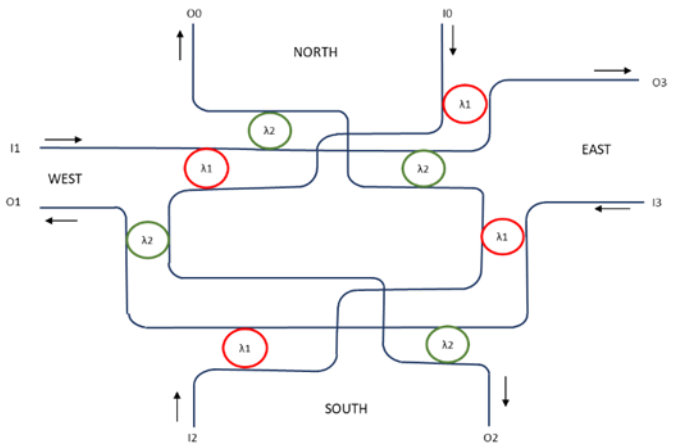

Figure 7.12 4x4 parallel O-NoC router

The routing table for the 4x4 router is shown below in table 7.3. The signal to an output port other three input ports are of different wavelengths. There are no sharp turns employed to lessen the bending losses.

	O0	O1	O2	O3
I0	--	λ2	λ3	λ1
I1	λ2	--	λ1	λ3
I2	λ3	λ1	--	λ2
I3	λ1	λ3	λ2	--

Table 7.3 Routing table for 4x4 O-NoC router

This router is extended to 5x5 architecture as shown in figure 7.13 employing 16

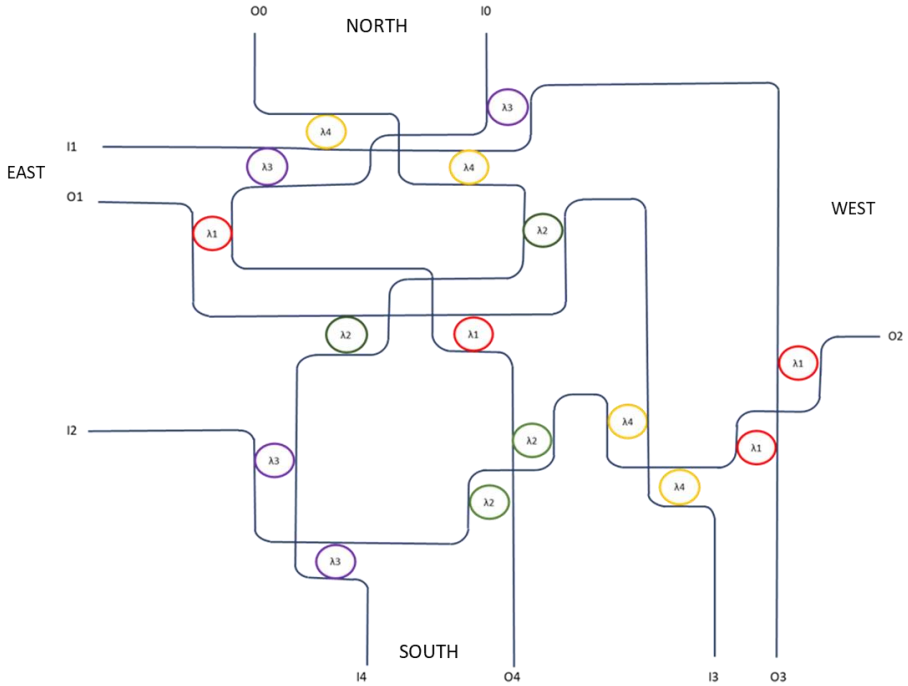

Figure 7.13 5x5 ONOC router

.

MRRs in a parallel configuration, and the routing table is table 7.4. Four different wavelengths are employed, it can be perceived that as wavelength division

	O0	O1	O2	O3	O4
I0	--	λ1	λ2	λ3	λ4
I1	λ4	--	λ1	λ2	λ3
I2	λ3	λ4	--	λ1	λ2
I3	λ2	λ3	λ4	--	λ1
I4	λ1	λ2	λ3	λ4	--

Table 7.4 Routing table for 5x5 ONOC router

multiplexing is employed, all input ports exploit distinct wavelengths to a particular destination port avoiding confliction.

7.4.2. Computing design parameters for MRR

The cross-coupling coefficient for the ring resonator in parallel configuration is shown in figure 7.14. As depicted from the figure, the strength of the cross-coupling coefficient decreases as the gap increases. Note that the cross-coupling coefficient considers the curvature of the ring. The following parameters tabulated in table 7.5 are considered while computing the best-fit parameters by curve fitting using Python.

$$\kappa = \left(\sin\frac{\pi}{\lambda}\right)\left[\frac{a_E}{\gamma_E}\, e^{-}\gamma_E d\, B(x_E) + \frac{a_O}{\gamma_O}\, e^{-}\gamma_O d\, B(x_O)\right] \qquad (1)$$

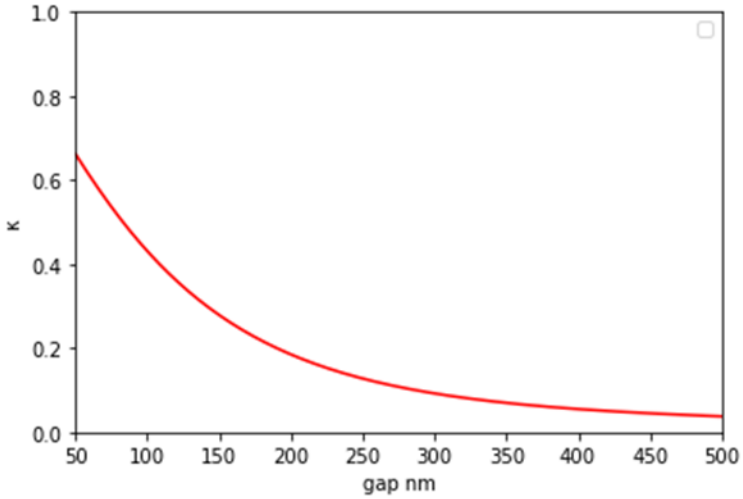

Figure 7.14 Variation of cross-coupling coefficient with the gap

Parameter	Value
Radius	10nm
Waveguide width	450nm
Wavelength	1550nm
neff	2.36

Table 7.5 Parameters considered for κ calculation

The best-fit parameters and the coupling coefficients computed through the equation are as shown in table 7.6.Figure 7.14 illustrates the variation of the coupling coefficient with the gap; it is evident that there endures a strong coupling amidst the ring and the waveguide at lower gaps. Considering a lossless resonator $\kappa^2 + t^2 = 1$, the through-coefficient is gauged to be 0.9563.The losses in the resonator determine the performance of the ring. Using the power-law fit[2], the loss α[dB/cm] is 4.5653[dB/cm].The transmission via the add and drop ports of the ring established in Figure 7.15 concludes that the parameters opted will provide acceptable transmission

characteristics. The quality factor is found to be 12548.25.The blue coloured transmission curve represents the output of the all-pass resonator, green for the drop-

Parameter	Value
a_E	0.225
a_O	0.010
γ_O	0.011
γ_E	0.002
d	150nm
κ	0.2924
t	0.9563
α[dB/cm]	4.5653[dB/cm]

Table 7.6 Estimated parameters of the MRR through non-linear curve-fitting using Python

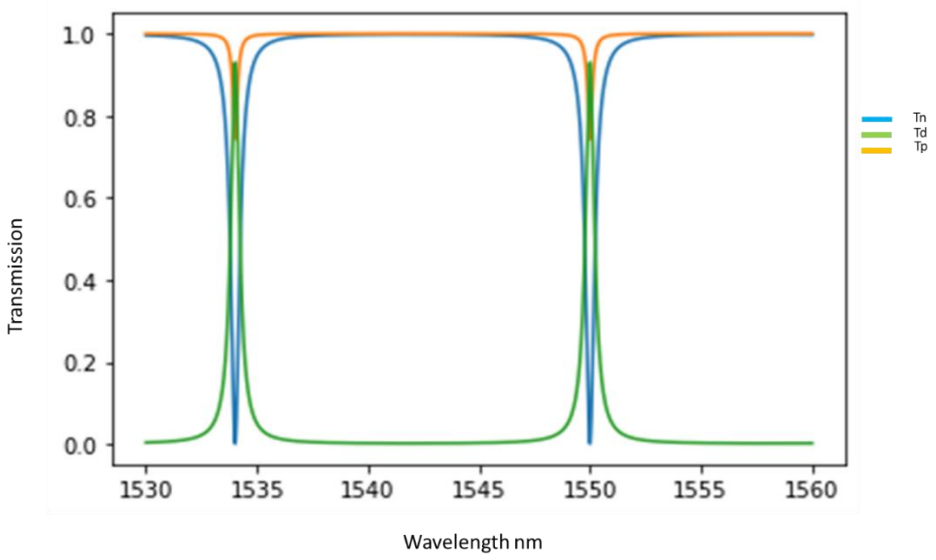

Figure 7.15 Transmission through the Tn (all-pass), Td(drop-port), Tp(pass-port) of the ring resonator

port and orange for the add-port of the MRR.

7.4.3 Tunability of MRR

As explained in chapter 6, a thin layer of Indium Tin oxide ITO (20nm) is deposited on the top of Silicon dioxide to act as the gate electrode enabling the electrical tuning

Figure 7.16 Indium-Tin-Oxide as the gate electrode for MRR used for electrical tuning of the resonator.

Chapter 8 - Conclusions and Future Scopes

8.1. Conclusion

As demonstrated in the previous chapters, the problems regarding handling a massive amount of data, finding, filtering and decoding the status of debug registers of the Network-on-Chip and design flow that includes multiple toolruns, excessive command-line usage and manual constraint file update. This thesis addresses the resolution by exploiting automation and framework creation eliminating the complex tasks incorporated in a process incorporating Python, Perl, Bash. Also, using Python we have estimated the coupling coefficients of the resonator in the design of the Optical Network on chip router. We see that employing these resolutions have reduced the efforts and time put by the user.

- Data visualisation for area analysis gives a picture of the area of the sub-elements present in different interconnects from the region-wise area CSV files.
- Scaled areas furnished by the framework from the given technology nodes is an added advantage that projects the trends of area variations.
- By adding the configuration files, an interface to configure the required settings is provided.
- Debug automation framework has abolished the effort to filter around 220 registers from the given debug dump data, check for hang states and aggregate the data to find the reason for hang. Debug framework takes the URL link of Silicon dumps as input and captures the required data. Hang data analysis could be repetitive depending on the number of dumps given in the debugging phase for a chip.

- The design flow automation that incorporates various flow checks has been automated to run the flow checks. The total time taken is reduced; however, the tool run time is constant
- Constraint file updates involving many files to update from various directories are also eradicated. Creating element was also automated.
- The command-line usage is reduced. Error logging enabled in the design flow automation reduced the effort and time in debugging the errors.
- Design flow of Network-on-Chip optical router in the 4x4 configuration adopting MRRs in parallel lessens the insertion losses. The design described diminishes the interference and cross talk noise. The ring resonators employed are tunable electrically.

8.2. Future Scope

- The automation framework can be extended to provide area analysis among different projects graphically, including scaling provision.
- Improving the debug analysis framework to not only capture the hang state-related data but also to decode the significance of bits in it.
- To optimise the Network on-chip router (optical) to reduce the number of ring resonators to bring down the power and area consumption

REFERENCES

1. D. Mahajan, S. Patil, W. V. Shashikant, M. Dangayach, P. V. Bhanu and J. Soumya, "Design Automation of Network-on-Chip Prototype on FPGA," *2019 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER)*, Manipal, India, 2019, pp. 1-4, doi: 10.1109/DISCOVER47552.2019.9008005.

2. W. J. Dally and B. Towles, "Route packets, not wires: on-chip interconnection networks," *Proceedings of the 38th Design Automation Conference (IEEE Cat. No.01CH37232)*, Las Vegas, NV, USA, 2001, pp. 684-689, doi: 10.1109/DAC.2001.156225.

3. S. Kumar *et al*., "A network on chip architecture and design methodology," *Proceedings IEEE Computer Society Annual Symposium on VLSI. New Paradigms for VLSI Systems Design. ISVLSI 2002*, Pittsburgh, PA, USA, 2002, pp. 117-124, doi: 10.1109/ISVLSI.2002.1016885.

4. A. Kumar and P. R. Panda, "Front-End Design Flows for Systems on Chip: An Embedded Tutorial," *2010 23rd International Conference on VLSI Design*, Bangalore, India, 2010, pp. 417-422, doi: 10.1109/VLSI.Design.2010.70.

5. T.R. Mück, A.A. Fröhlich,Aspect-oriented RTL HW design using SystemC,Microprocessors and Microsystems,Volume 38, Issue 2,2014,Pages 113-123,ISSN 0141-9331, https://doi.org/10.1016/j.micpro.2013.12.002.

6. Shubhyant Chaturvedi. 2012. Static analysis of asynchronous clock domain crossings. In Proceedings of the Conference on Design, Automation and Test in Europe (DATE '12).EDA Consortium, San Jose, CA, USA, 1122–1125.

7. A. Yadav, P. Jindal and D. Basappa, "Study and Analysis of RTL Verification Tool," *2020 IEEE Students Conference on Engineering & Systems (SCES)*, Prayagraj, India, 2020, pp. 1-6, doi: 10.1109/SCES50439.2020.9236747

8. L. Leinweber, B. Singh and C. Papachristou, "Expert System Simulation of Hardware," 2013 IEEE 25th International Conference on Tools with Artificial Intelligence, Herndon, VA, USA, 2013, pp. 207-212, doi: 10.1109/ICTAI.2013.40.

9. E. Garat, D. Coriat, E. Beigné and L. Stefanazzi, "Unified Power Format (UPF) methodology in a vendor independent flow," *2015 25th International Workshop on Power and Timing Modeling, Optimization and Simulation (PATMOS)*, Salvador, Brazil, 2015, pp. 82-88, doi: 10.1109/PATMOS.2015.7347591.

10. F. Bembaron, R. Mukherjee, S. Kakkar and A. Srivastava, “Low power verification methodology using UPF”, Design & Verification Conference & Exibition (DVCon 2009), San Jose CA, pp. 228-233, in press

11. S. Abadal, M. Iannazzo, M. Nemirovsky, A. Cabellos-Aparicio, H. Lee and E. Alarcón, "On the Area and Energy Scalability of Wireless Network-on-Chip: A Model-Based Benchmarked Design Space Exploration," in IEEE/ACM Transactions on Networking, vol. 23, no. 5, pp. 1501-1513, Oct. 2015, doi: 10.1109/TNET.2014.2332271.

12. S. Sahu and H. M. Kittur, "Area and power-efficient network on-chip router architecture," 2013 IEEE Conference on Information & Communication Technologies, Thuckalay, India, 2013, pp. 855-859, doi: 10.1109/CICT.2013.6558214.

13. S. Kumar, N. Dhanda and A. Pandey, "Data Science — Cosmic Infoset Mining, Modeling and Visualization," 2018 International Conference on Computational and Characterization Techniques in Engineering & Sciences (CCTES), Lucknow, India, 2018, pp. 1-4, doi: 10.1109/CCTES.2018.8674138.

14. Design compiler graphical,https://www.synopsys.com/implementation-and-signoff/rtl-synthesis-test/design-compiler-graphical.html.

15. M. H. Neishaburi and Z. Zilic, "Debug Aware AXI-based Network Interface," 2011 IEEE International Symposium on Defect and Fault Tolerance in VLSI and Nanotechnology Systems, Vancouver, BC, Canada, 2011, pp. 399-407, doi: 10.1109/DFT.2011.34.

16. AMBA AXI Protocol Specification, Version 2.0, 19 March 2004.

17. Butwall, Mani et al. “Python in Field of Data Science: A Review.” *International Journal of Computer Applications* 178 (2019): 20-24.

18. S. Dwivedi, T. Van Vaerenbergh, A. Ruocco, T. Spuesens, P. Bienstman, P. Dumon, and W. Bogaerts, "Measurements of Effective Refractive Index of SOI Waveguides using Interferometers," in *Advanced Photonics 2015*, OSA Technical Digest (online) (Optical Society of America, 2015), paper IM2A.6.

19. M. Bahadori *et al*., "Design Space Exploration of Microring Resonators in Silicon Photonic Interconnects: Impact of the Ring Curvature," in *Journal of Lightwave Technology*, vol. 36, no. 13, pp. 2767-2782, 1 July1, 2018, doi: 10.1109/JLT.2018.2821359.

20. Bogaerts, W., De Heyn, P., Van Vaerenbergh, T., De Vos, K., Kumar Selvaraja, S., Claes, T., Dumon, P., Bienstman, P., Van Thourhout, D. and Baets, R. (2012), Silicon microring resonators. Laser & Photon. Rev., 6: 47-73. https://doi.org/10.1002/lpor.201100017L. 4.

21. Zhang et al., "On reducing insertion loss in wavelength-routed optical network-on-chip architecture," in IEEE/OSA Journal of Optical Communications and Networking, vol. 6, no. 10, pp. 879-889, Oct. 2014, doi: 10.1364/JOCN.6.000879.

22. M. Nikdast et al., "Crosstalk Noise in WDM-Based Optical Networks-on-Chip: A Formal Study and Comparison," in IEEE Transactions on Very Large Scale Integration (VLSI) Systems, vol. 23, no. 11, pp. 2552-2565, Nov. 2015, doi: 10.1109/TVLSI.2014.2370892.

23. Eyal Feigenbaum, Kenneth Diest, and Harry A. AtwaterUnity-Order Index Change in Transparent Conducting Oxides at Visible Frequencies,*Nano Letters* 2010 *10* (6), 2111-2116,DOI: 10.1021/nl1006307.

24. Wen-Chung Tsai, Ying-Cherng Lan, Yu-Hen Hu, and Sao-Jie Chen. 2012. Networks on chips: structure and design methodologies. JECE 2012, Article 2 (January 2012), 1 pages. DOI:https://doi.org/10.1155/2012/509465.

25. X. Tan, M. Yang, L. Zhang, Y. Jiang and J. Yang, "A Generic Optical Router Design for Photonic Network-on-Chips," in *Journal of Lightwave Technology*, vol. 30, no. 3, pp. 368-376, Feb.1, 2012, doi: 10.1109/JLT.2011.2178019.

26. L. Zhang, Y. Man, X. Tan, M. Yang and Y. Jiang, "Wavelength routed optical network-on-chip architecture with lower insertion loss," *2014 Optical Interconnects Conference*, 2014, pp. 65-66, doi: 10.1109/OIC.2014.6886073.

27. Zaki, A.O., Kirah, K. & Swillam, M.A. Hybrid plasmonic electro-optical modulator. Appl. Phys. A 122, 473 (2016). https://doi.org/10.1007/s00339-016-9843-y.

28. A. Patil, M. Proeller, A. Kshirasagar and A. Nahar, "A Framework for Automated Testing of RTL Designs," *2015 International Conference on Computing Communication Control and Automation*, 2015, pp. 989-991, doi: 10.1109/ICCUBEA.2015.195.

29. G. A. Allan, "Targeted layout modifications for semiconductor yield/reliability enhancement," in *IEEE Transactions on Semiconductor Manufacturing*, vol. 17, no. 4, pp. 573-581, Nov. 2004, doi: 10.1109/TSM.2004.835727.

30. S. Neeli, K. Govindasamy, B. M. Wilamowski and A. Malinowski, "Automated Data Mining from Web Servers Using Perl Script," 2008 International Conference on Intelligent Engineering Systems, 2008, pp. 191-196, doi: 10.1109/INES.2008.4481293.

31. Nikoukar, Ali & Sadegh Amiri, Iraj & Alavi, S. & Shahidinejad, Ali & Anwar, Toni & Supa'at, Abu & Idrus, Sevia & Teng, Lo. (2014). Theoretical and Simulation Analysis of The Add/Drop Filter Ring Resonator Based on the Z-transform Method Theory. Quantum Matter.

32. D. Mahajan, S. Patil, W. V. Shashikant, M. Dangayach, P. V. Bhanu and J. Soumya, "Design Automation of Network-on-Chip Prototype on FPGA," 2019 IEEE International Conference on Distributed Computing, VLSI, Electrical Circuits and Robotics (DISCOVER), 2019, pp. 1-4, doi: 10.1109/DISCOVER47552.2019.9008005.

33. C. Chen and S. D. Cotofana, "Towards an Effective Utilization of Partially Defected Interconnections in 2D Mesh NoCs," 2014 IEEE Computer Society Annual Symposium on VLSI, 2014, pp. 492-497, doi: 10.1109/ISVLSI.2014.70.

34. Bergman, Keren, et al. Photonic Network-on-Chip Design. Springer, 2014.