University of Cincinnati

Date: 5/20/2013

I, Jiayong Liu , hereby submit this original work as part of the requirements for the degree of Master of Science in Computer Engineering.

It is entitled:

Variance Validation for Post-Silicon Debugging in Network on Chip

Student's name: **Jiayong Liu**

This work and its defense approved by:

Committee chair: Wen Ben Jone, Ph.D.

Committee member: Carla Purdy, Ph.D.

Committee member: Ranganadha Vemuri, Ph.D.

3546

Variance Validation for Post-Silicon Debugging in Network on Chip

A thesis submitted to the

Graduate School
of the University of Cincinnati

in partial fulfillment of the
requirements for the degree of

MASTER OF SCIENCE

in the School of Electronics & Computing System
of the College of Engineering & Applied Science

by

Jiayong Liu
B.E. (Electronic Science and Technology), Shanghai Jiao Tong University, July 2007

Thesis Advisor and Committee Chair: Dr. Wen-Ben Jone

Abstract

Since more complex components are integrated into a single chip and scale of chip goes far beyond, pre-silicon verification is getting hard to achieve full coverage at chip level and catch design bugs under physical conditions. As a complementary checking step, post-silicon validation has demonstrated its importance in verifying chip functionality. But still, the task of post-silicon validation is extremely difficult, especially for complex designs such as network on chip. In this thesis, a novel on-chip validation method based on the concept of variance validation is proposed to facilitate the process of post-silicon validation targeting the network on chip platform. Cores are paired and tagged as a functional core and a validating core in each pair. Variances are created for programs executed in each pair of cores. By comparing the outputs from each pair of cores, the method enables at-speed on-chip validation for core-based architectures and helps detect functional bugs after chip manufacturing. With little effort, the method can be extended to support reliable system design. Experiments are carried out and effectiveness of the proposed variance validation method is demonstrated by simulation results.

To my Dear Parents

Acknowledgment

First, I would like to thank my advisor, Dr. Wen-Ben Jone for his generous mentoring. Without his academic advice, constructive feedback and continuous encouragement, I could not think of completing my thesis work. He shows the capability of talking in the art while still be focused on the direction of research. Thanks to his comments, I continue developing new ideas towards the goal of thesis and reviewing what can be a better choice. And extremely thanks for meeting with me in a tight schedule when he is busy and talking in the air when we cannot meet face to face.

Second, I would like to thank my committee members, Dr. Ranga Vemuri and Dr. Carla Purdy. Thanks for taking time reviewing my thesis work and your valuable inputs. Also great thanks to all the professors in the College of Engineering and Applied Science. I did learn a lot from your lectures and courses, together with the course projects.

Third, I would like to thank all my friends in University of Cincinnati. I enjoy the time we spent together and the help your guys offer.

Last, I would like to thank my dear parents. I appreciate their support

and love, especially in the hard times. It is an honor to be taken care of in years of study.

Thank you all.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

Post-Silicon validation usually includes all the validation efforts after the first tape-out of chips and before ramping up to high-volume manufacturing. It is intended to catch all possible escaped bugs from verification efforts before chip manufacturing and prevent the potential disastrous consequences of high-volume production and shipment of defective products [15].

As shown in Figure 1.1, a design is first created from the specification. After rounds of design and verification processes, the first tape-out of the design is created. Starting from the first tape-out, post-silicon validation is carried out to make sure each silicon matches the original specification. Any mismatch between a silicon and its specification will cause a redesign to fix the bug. The round of tape-out and redesign is usually called a new silicon spin.

As indicated in [16], from the report of 2004/2002 IC/ASIC Functional

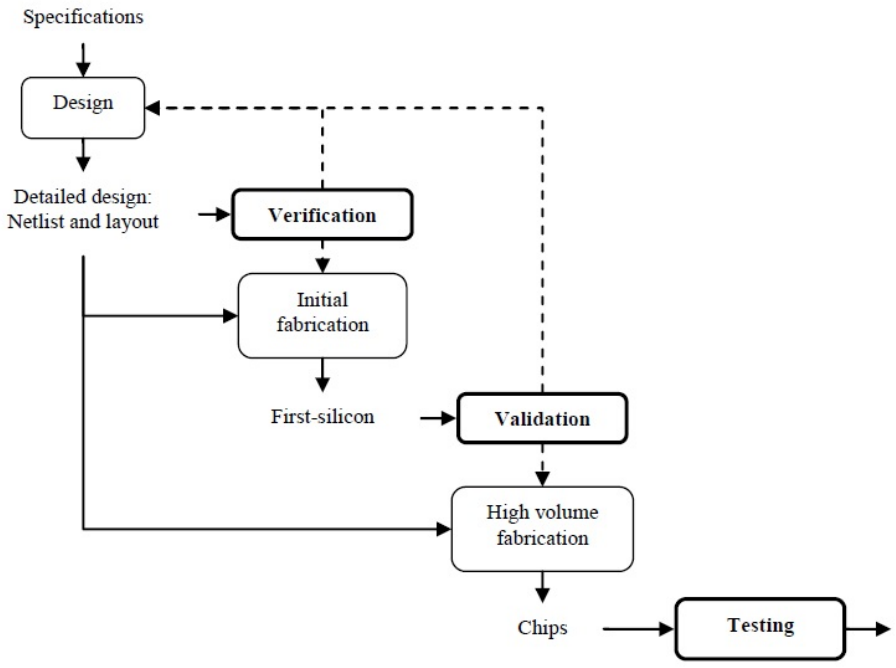

Figure 1.1: A typical design flow including verification and validation [1].

Verification Study by Collett International Research, it is claimed that half of all chip developments require a re-spin, three quarters due to functional bugs. Such a re-spin is expensive since it requires diagnosis to identify the root cause, redesign, creation of a new set of masks, and re-fabrication [1].

Pre-silicon verification covers most of the design bugs but it is still not enough due to limited testcases. For the block level design, pre-silicon verification does well in verifying the functionality. But at the full-chip level, it is hard to achieve full coverage in tolerable time slot. Also due to lack of noise mode, there is no simple way to verify functionality under physical impacts. As a complementary, post-silicon validation is used to guarantee

chip functionality in a real silicon and design bugs escaped from pre-silicon verification are expected to be caught in the post-silicon validation.

Post-silicon validation has become increasingly difficult in several aspects. As the complexity and density of VLSI circuits go far beyond imagination, the proportion of post-silicon validation resources as a percentage of design resources has gone up significantly over the last decade [17], and the role of post-silicon validation has become dramatically important.

After the initial silicon tape-out is ready, post-silicon validation is carried out to detect bugs in real silicon chips. In the process of post-silicon validation, there is little insight into the silicon states due to limited observability and the only interface is package pins. Internal states can be shifted out through scan chains, but there is still much less information when compared to pre-silicon verification. Also the bandwidth of data transmission dominates the progress of post-silicon validation.

Due to short post-silicon validation cycles, it is important that these bugs can be detected rapidly. Exhaustive test patterns and enumerative exercising are neither plausible nor affordable. Hence an efficient and fast method for post-silicon validation is expected to meet the tight schedule. Details on location and cause of bugs are also required for a fast redesign fix.

Besides the coverage gained from pre-silicon verification, post-silicon validation is given the hope to find bugs that occur in the circumstances of uncommon usage. Simulation speed is quite slow and it is almost impossible to run a full-coverage pre-silicon verification. But tests running at speed

in silicons, potentially allow much more test patterns and better coverage. Post-silicon validation is expected to guarantee the success of the following high-volume production.

According to [17], most of traditional post-silicon debug techniques still rely on the use of increasingly expensive logic analyzers for observing external interfaces of the Device Under Test (DUT). This adds up to the validation cost. The more time is spent on post-silicon validation, the more expensive additional cost is.

With shrinking VLSI technologies, the density and complexity of VLSI circuits has been greatly increased recently, and there are several limitations for traditional design methodology. For example, design productivity does not scale well with technologies, and power consumption on global clocking makes it intolerable for heat dissipation. Long wires cause excessive delay that can be more than the number of tolerable clock cycles. A new design method, network on chip (NOC), seems to be a prominent solution for the above problems.

The NOC architecture has its natural advantage of scaling as it integrates multiple cores into the same chip and connects them via a network. In the NOC architecture, global clocking is not required and each core can have its localized clocking. Usage of long wire can be avoided since communication between cores is made through routers and the network. Compared to traditional bus-based communication methods, the bandwidth that the NOC architecture is able to supply is much higher. Recently there has been real

NOC chips coming into world. For example, it is reported in [18] that Intel has created a 80-tile processor based on NOC.

According to [19, 20], SystemC is a single, unified design and verification language that expresses architectural and other system-level attributes in the form of open-source C++ classes. It enables design and verification at the system level, independent of any detailed hardware and software implementation, as well as enabling co-verification with RTL design [20].

SystemC is widely used in ASIC industry and VLSI circuit research [21]. The mini-NOC project [22] was developed in SystemC and then it is extended in this thesis work for post-silicon validation.

In this thesis, a variance validation method is proposed to perform post-silicon validation for each pair of functional core and validating core targeting a NOC platform. It is full of core resources in the NOC platform, and roles of cores in the NOC platform can be divided into functional role and validating role. To take advantage of the property in the NOC platform, the following changes are made. Some of the cores in NOC are in place to fulfill the function of NOC platform. While the other cores in NOC are reconfigured to perform the task of validation for these functional cores. In detail, two adjacent cores are paired as a functional core and validating core group to realize the idea. In addition, a FIFO buffer is inserted to serve as communication channel in each pair between functional core and validating core.

With the pairing of a functional core and a validating core, post-silicon validation can be performed by running the same program in both functional

core and validating core, and comparing the outputs to see if they match. Given the fact that most NOC platforms are composed of homogeneous cores, there is high possibility for a bug to escape because we have the same program running in cores with the same circuit design and the same erroneous design creates the same erroneous results. To resolve this problem, variances are created between functional program and validation program to detect potentially escaped bugs. Simulation results demonstrate the feasibility of the proposed method.

The thesis is organized as following:

Chapter 2 introduces the NOC architecture and reviews the background, progress and related works of post-silicon validation.

Chapter 3 introduces the NOC platform derived and modified from mini-NOC project.

Chapter 4 presents the method of variance validation and discusses its application to post-silicon validation and reliable system design.

Chapter 5 provides a case study on spatial filter application for post-silicon validation based on the proposed method.

Chapter 6 presents experimental results and design bug analysis.

Chapter 7 concludes the thesis work, summarizes the contributions and proposes possible directions for future research work.

Chapter 2

Background

2.1 Introduction to network on chip

With the development of IC technology, VLSI circuits have been made more and more complex and it is possible to integrate multiple complex designs, such as cores, into the same chip. Thus idea of inter-core communications is brought up and prototype of network on chip is proposed and researched for years [2, 23, 24, 25, 26, 27, 28] .

In general, there are several components in the NOC architecture. Multiple cores are integrated into the chip. Routers are connected in a network for communications between cores. Network interface connects each pair of core and router and provide network access support for cores.

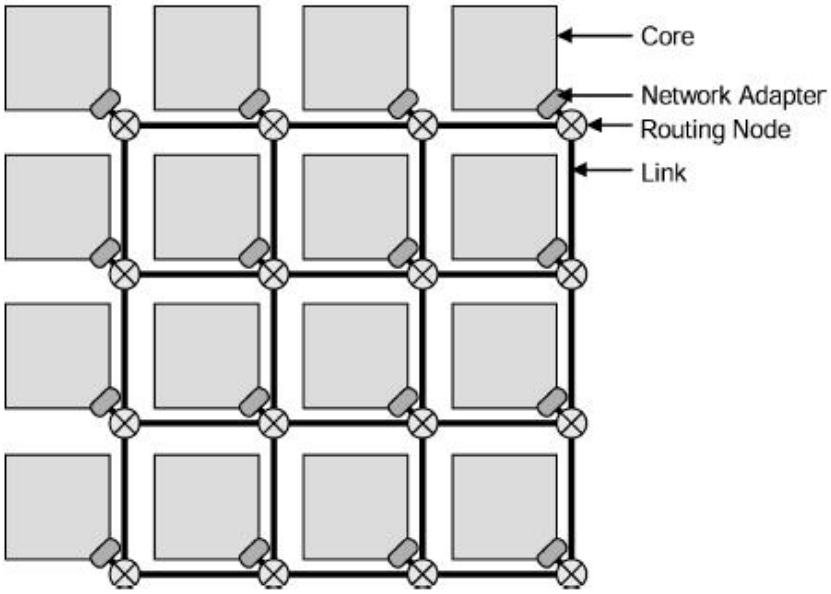

Figure 2.1: Diagram of 4 by 4 mesh network on chip [2].

2.1.1 Topology

Some typical topologies are listed below [3].

1. Ring

In a ring network, all routers are connected in a ring fashion. Every router has two neighbor routers [3].

2. Mesh

A mesh network consists of m columns and n rows. The network is in nature a plane and routers can be placed with equal length of links. The addresses of routers and cores can be easily defined as x-y coordinates in mesh network [3].

3. Torus

A torus network is an improved version of mesh network. Beside the structure of mesh network, the up sides of columns are connected to the down sides of columns and the left sides of rows are connected to the right sides of rows [3].

4. Star

A star network has a central router and all other routers are connected to the central router [3].

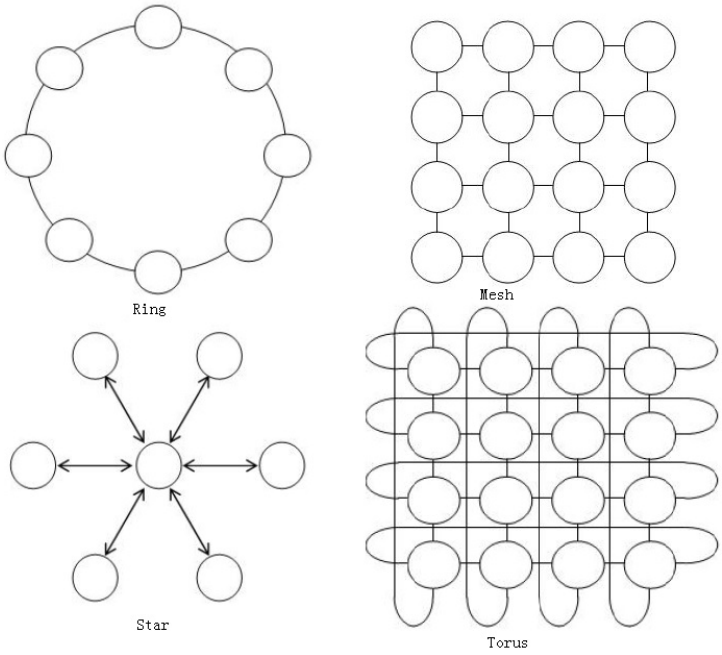

Figure 2.2: Examples of NOC topology [3]

2.1.2 Routing problems

Deadlock, livelock and starvation are potential problems [29] on existing routing algorithms.

- Deadlock

In the situation of deadlock, two packets are waiting for each other to be further processed. Both packets reserve some resources and are waiting for the other to release the resources. Hence, no one is releasing the resources until it gets the new resources and therefore the routing is locked.

- Livelock

Livelock is the situation in which a packet keeps spinning around its destination but never reaching it.

- Starvation

When using priorities with packets, there is a situation of starvation when the packets with low priorities never reach their destinations because these higher priorities hold the resources all the time and never release them.

2.1.3 Routing methods

Several popular routing algorithms used in NOC platform are listed below.

- XY routing

It is a dimension order routing which routes packets first in x or horizontal direction to the correct column and then in y or vertical direction to the receiver [30]. XY routing works well on a mesh or torus network. Addresses of the routers are their xy-coordinates.

- Turn model algorithms

Turn model algorithms [31] determine a turn or turns which are not allowed while routing packets through a network. West-first Routing is one example of turn model routing method. In a west-first routing algorithm, the packets going to west must be first transmitted as far to west as necessary because routing to west is not allowed later.

- Shortest path routing

A shortest path routing is the simplest deterministic routing algorithm. Packets are always routed along the shortest possible path. Distance Vector Routing [32] is one example. Routers route packets by counting the shortest path and then send packets forward.

2.1.4 Flow control

Flow control determines how network resources, such as channel bandwidth, buffer capacity and control state, are allocated to a packet traversing the network. It governs and determines when buffers and links are assigned to

messages, the granularity at which they are allocated and how these resources are shared among the many messages using the network [33].

Flow control techniques are classified by the granularity at which resource allocation is handled: based on message, packet or flit. Messages that have to be transmitted across the network are usually partitioned into fixed-length packets. Packets in turn are often broken into message flow control units called flits [34].

2.1.5 Major components in NOC platform

1. Router

Routers are connected to construct the network and perform the function of data transmission between cores. It usually consists of a set of input and output buffers, crossbar and control logic.

2. Network interface

Network interface usually handles the packetization, packet re-ordering and controlling the retransmissions. It serves as a bridge between cores and routers.

3. Core

Cores in NOC usually refer to microprocessors. They are duplicated in each node and paired with routers.

2.2 Post-silicon validation

2.2.1 Overview of post-silicon validation

Post-silicon validation is an integral part of VLSI circuits development. According to [13], post-silicon validation is becoming significantly difficult and prohibitively expensive because existing techniques cannot cope with the sheer complexity of future systems [35, 36, 37]. Recently, lots of research works have been going on to address the problem and propose solutions for post-silicon validation [4, 5, 6, 8].

Post-silicon validation is not a simple re-check after a great number of efforts in verification before chip manufacturing. As described in [14], typical bugs uncovered in post-silicon validation are almost impossible to be completely checked in simulation. These bugs includes functional bugs and electrical bugs. Functional bugs usually refer to the bugs that cannot be verified before chip manufacturing due to limited resources in simulation or emulation. There is everything inside the chip and it is easy to run chip level validation in the stage of post-silicon validation. The situation is completely different from verifications before chip manufacturing. Post-silicon validation aims at catching such functional bugs escaped from verification before chip manufacturing. Electrical bugs are due to operation of circuit conditions such as temperature, voltage, frequency, crosstalk or power-up problems. The electrical bugs can not be fully detected in the stage of manufacture testing. Manufacture testing ensures circuits are manufactured exactly the

same as the designed structure while electrical bugs escape from manufacture testing because they impact the functionality only in certain circumstances.

As post-silicon validation is performed after chip manufacturing, bug fixing can be expensive if there is a bug found in this step. According to [14], if a problem is found in silicon that requires redesign and a silicon re-spin, six months to a year will be lost in IP delivery. On the other hand, a problem found in emulation can be identified, debugged and fixed in days. Timing errors found after place and route can usually be fixed quickly with an ECO flow, and even bugs that require resynthesis add perhaps a few weeks to a schedule.

Usually in the ASIC design flow, there are several major checking steps such as pre-silicon verification, manufacturing test and post-silicon validation. Pre-silicon is done before chip manufacturing or even chip implementation. After chip manufacturing, manufacturing test is done first to ensure there is not any significant manufacture defects in the process of manufacturing. And then post-silicon validation is performed extensively so that design bugs and malfunctions are detected and fixed before end-user.

There are both differences and connections between Pre-silicon verification and post-silicon validation.

1. Observability:

For pre-silicon verification, prober can be attached to any signal in the design, but in the post-silicon validation, the only interface is the input and output pins of the chip.

Table 2.1: Comparison for pre-silicon verification, manufacturing testing and post-silicon validation. [13]

Pre-silicon verification	Manufacturing testing	Post-silicon validation
Excellent controllability and observability because any signal can be accessed	Controllability and observability primarily through scan DFT	Insufficient controllability and observability due to limited access to internal signals. Scan DFT useful for certain cases when a failure caused by a bug is repeatable.
Complex physical effects difficult to model	Several defect models exist	Accounts for signal-integrity, process variations, non-determinism
Simulation of full-chip designs very slow; formal verification only selectively applicable	Generally very fast (few seconds to minutes per chip)	Silicon speed orders of magnitude faster than simulation
Some metrics exist (e.g., code coverage, assertion coverage, mutation coverage)	Test Metrics (e.g., stuck-at, transition, N-detect coverage) widely used	Coverage metrics for post-silicon validation: Open research question
Bug fixing inexpensive	Bug fixing not the primary objective	Bug fixing can be expensive

2. Runtime speed and scale:

To ensure chip works as designed, exhaustive testbenches are used in the pre-silicon verification, but it can only be done in computer simulation. The simulation speed is extremely slow and capability to simulate large designs is limited. In the contrary, post-silicon validation runs at-speed and the whole chip is running without capacity issue.

From the example in [14], consider a 400MHz design simulated at 2kHz. The silicon is 200,000 times faster than the simulation, so 55 hours of simulation are required to simulate 1 second of actual operation.

Table 2.2: Throughput of design and various abstractions [14]

Approach	Throughput
System simulation	10^3
RTL simulation	$10^1 to 10^3$
Gate simulation	$10^{-1} to 10^1$
Emulation	10^5
FPGA prototyping	10^6
Silicon	$10^7 to 10^9$

3. Stimulus control:

External stimulus can be applied at any size or level of block in pre-silicon verification. While in post-silicon validation, it is difficult to apply signal stimulus to the low level block.

4. Electric and circuit impact:

Post-silicon validation suffers from electric and circuit issues, and these impacts cannot be detected in pre-silicon verification.

2.2.2 Bridging pre-silicon verification and post-silicon validation

Some researches focus on eliminating the gap between pre-silicon verification and post-silicon validation.

Both pre-silicon verification and post-silicon validation aim at verifying the VLSI circuit and improving coverage, but they are different in nature. It is easy to observe any signal in the stage of pre-silicon validation, but the

same thing is almost impossible in post-silicon validation. Observability is limited in the stage of post-silicon validation [38].

Even though pre-silicon verification checkers cannot be applied directly in post-silicon validation, the effort spent in pre-silicon verification is not in vain. It provides guidance for post-silicon validation and thus is possible to ease the task of post-silicon validation. Researches in bridging the gap between pre-silicon verification and post-silicon validation are addressed in [39, 40, 41, 42].

A unified methodology for pre-silicon verification and post-silicon validation is proposed in [42]. In their proposal, test specification is shared between pre-silicon verification and post-silicon validation. A random stimuli generator, threadmill, is developed to generate generic test patterns based on shared test specification. And accelerator platform is used in post-silicon validation to overcome the limited observability. From the experiment in which the unified methodology is applied to POWER7 processor, coverage from post-silicon validation is almost similar to the simulation coverage.

2.2.3 Scan-based approaches

Similar to the approaches used in the manufacture testing, record-analysis approach is also used post-silicon validation.

1. IFRA(Instruction Footprint Recording and Analysis) [4]

According to [4], a method called "Instruction Footprint Recording and Analysis" is used in post-silicon validation targeting processors. In their idea, low-cost hardware recorders are inserted into design. And during the stage of post-silicon validation, recorders are applied non-intrusively to capture the last few thousand cycles of history before a failure manifests. After a failure is caught, the recorded footprints are scanned out through a Boundary-scan JTAG interface [43]. The footprints, together with the binary of the program executed during post-silicon validation, are then post-processed using special analysis techniques for bug localization.

In their method, a set of distributed recorders with dedicated storage, and a post-trigger circuit is employed as footprint recorder to capture instruction ID and auxiliary data. With these recorded information, it is able to tell which instruction caused the problem and which microarchitectural block was involved to execute the instruction. In the post-analysis after dump data are scanned out through JTAG interface, the localization analysis begins and inconsistency is identified by high-level post-analysis. Then backtracing is performed to tell the final bug location.

The method successfully bridges the circuit level and system level post-silicon validation. And they applied the method to an Alpha 21264-like superscalar processor for experiment. From their report, they claim the method is able to locate most injected bugs by increasing less than 0.2% of the area.

2. NOC platform debug

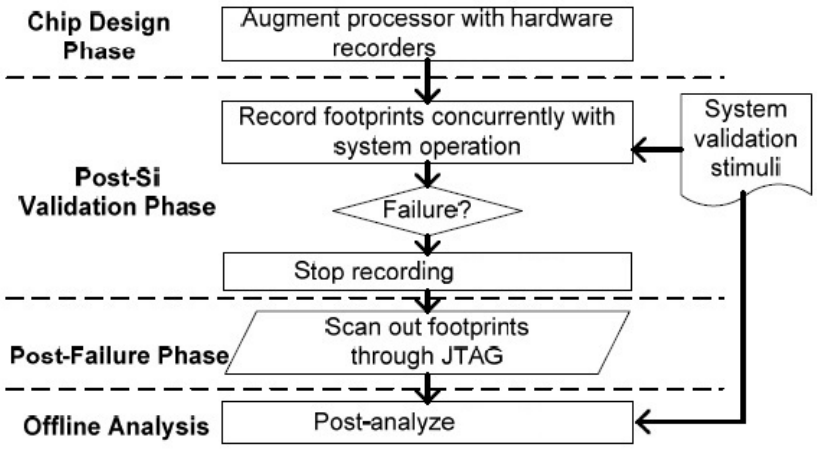

Figure 2.3: IFRA: Instruction Footprint Recording and Analysis [4]

According to [44], a debug platform is proposed for NOC-based systems. It includes three major components: the on-chip debug architecture, supporting debug software and off-chip debug controller. Debug probe modules are inserted asides cores and share the same network interface with cores. By reusing the data traffic network in the NOC system, debug agent plays the role of receiving debug commands from off-chip debug controller and controlling debug probes. Debug data are transmitted to off-chip debug controller through JTAG interface. And then debug software starts working on analysis of the debug data. Two-pass strategy is adopted in their method. Timestamps of trigger events are logged in the first pass, and then the trigger events are recovered with time information in the second pass.

Their method provides in-depth analysis features for NOC-based systems, such as NOC transaction analysis, multi-core cross-triggering and global synchronized timestamping. From their report, little area cost and limited NOC

traffic are spent in the proposed solution.

3. Dacota

As proposed in [5], a method is given to address the solution of post-silicon validation for memory subsystem. A simple in-hardware activity logging mechanism is employed to observe selected system activities during program execution in the stage of post-silicon validation. And periodically, a software-based validation algorithm examines the logs to detect violations in the ordering of memory operations, which is indicating an error in memory coherence or consistency.

From their report, dacota's post-silicon approach is able to offer significantly high coverage and effective in detecting subtle consistency and coherence bugs. And by reusing existing hardware resources, the solution is able to incur less than 0.01% area penalty for a commercial design.

4. Distributed logic analyzer

In the method [6], distributed triggering units are placed into design aside cores and different cores can be monitored simultaneously during post-silicon validation. Events from triggering units aside multiple cores are collected into trace buffer and offloaded through trace port for off-chip analysis at the end of debug session.

From the experiments in their report, their solution improves real-time observability with low area overhead in the design-for-debug architecture.

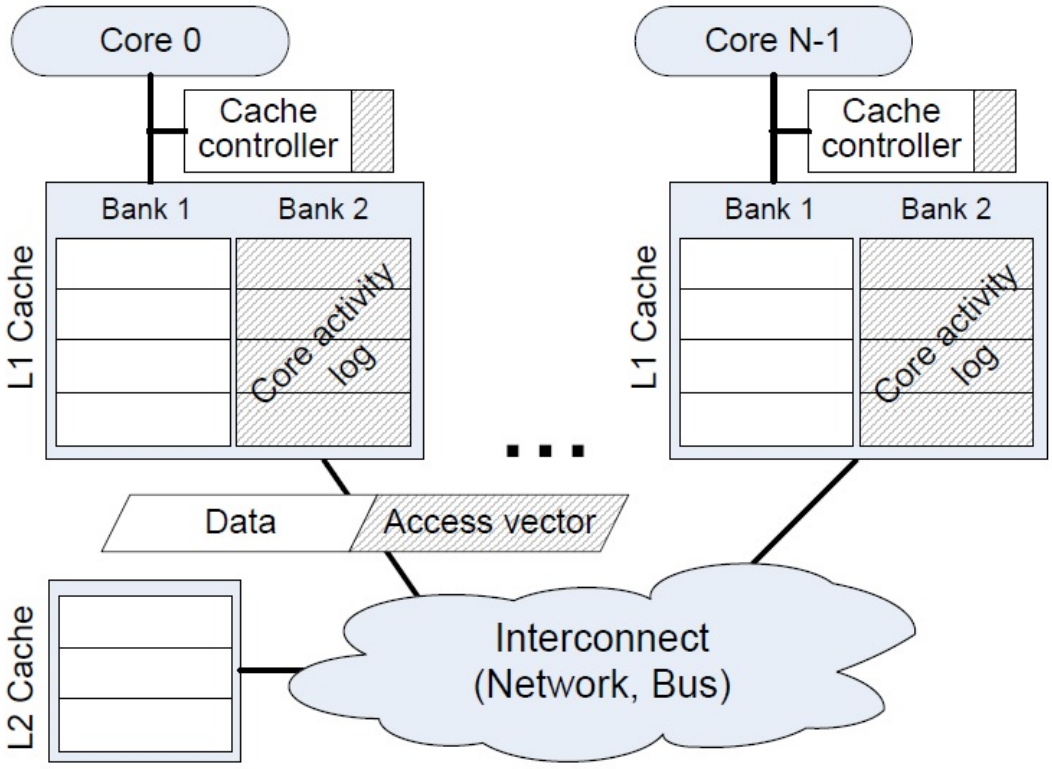

Figure 2.4: Dacota validation [5]

2.2.4 Formal method

Formal methods are also researched towards post-silicon validation in [7, 45, 46].

1. Specification-mining

According to [7], a method of scalable specification-mining is used in the bug diagnosis and bug locating. Normal trace and error trace are employed to find the first violation of consistency from distinguishing patterns. Since

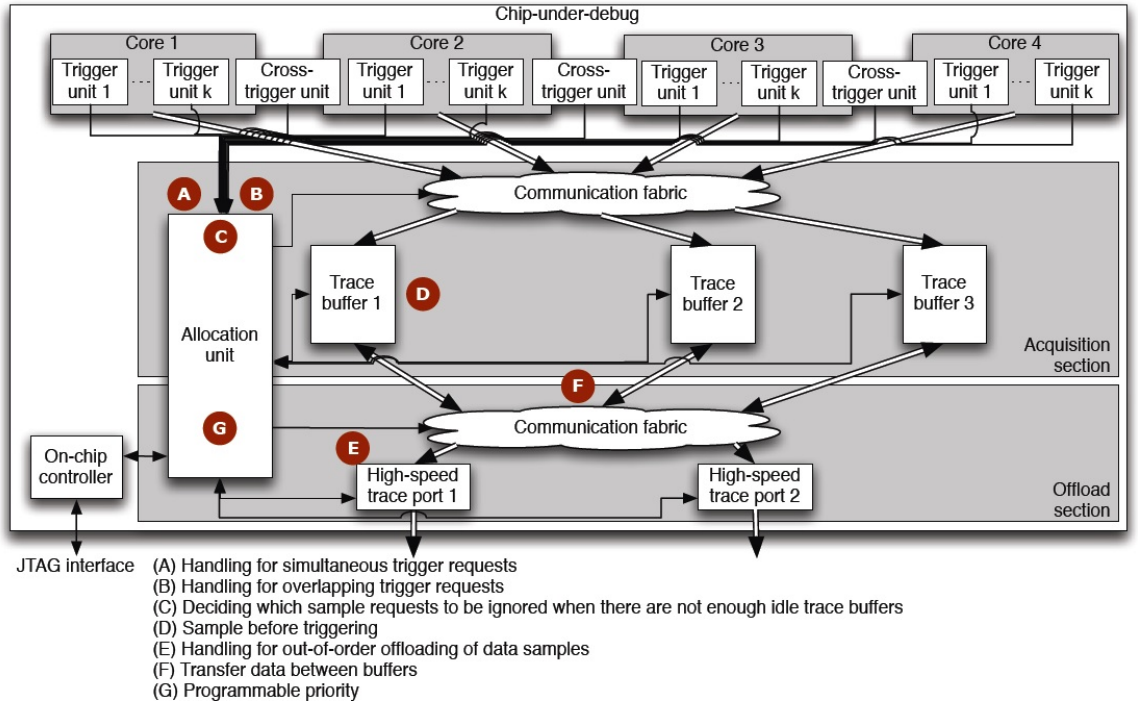

Figure 2.5: State-of-art design-for-debug architecture for trace buffer-based debug [6]

the pattern itself describes a specific erroneous behavior, it helps locate the bug.

2. Backspace

As discussed in [46], a new paradigm for using formal analysis is proposed. It is capable to automatically compute error traces that lead to an observed buggy state with some additional on-chip hardware support. Unlike platform dedicated approaches, the method has no targeting platform and applies to general digital designs.

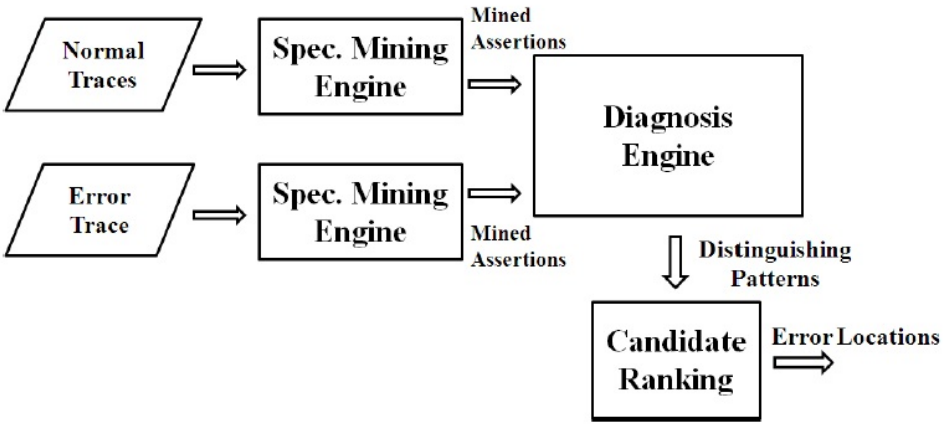

Figure 2.6: Specification mining for diagnosis [7]

In their framework, tests are running in the chip until crashes or bug exhibits. Then signature and full crash state are scanned out for formal analysis. Predecessor of the crash state is computed using formal method together with signature. The backtracing process continues until enough history trace is computed for debug.

They apply the method to two small processors/microcontrollers. From their experiments, they are able to compute hundreds of cycles of error trace backwards from a crash state.

2.2.5 Embedded logic checker

According to [8], they implemented an embedded FPGA blocks in a SoC design. By feeding in signals and states, the embedded FPGA block is re-configured to perform different assertion checkers. The method improves

the observability of internal signals inside the chip. And the whole system can run at speed for post-silicon validation. Both function tests and system validation are running simultaneously and on-chip.

From their report, the method is able to detect 39.4% of these hard-to-detect bugs by eighty hardware assertion checkers. And the area overhead is only 1.3%. They claim the method significantly reduces the time and effort for identifying the root causes of these detected bugs.

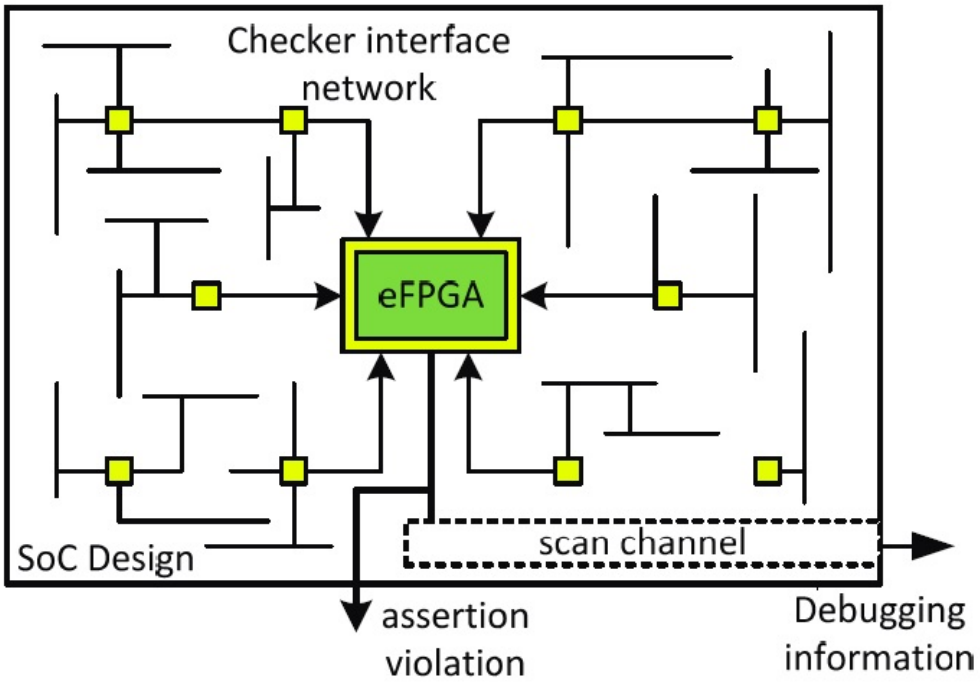

Figure 2.7: An example using a centralized eFPGA as the checker space [8]

2.2.6 Similar approaches

1. Quick error detection

According to [9], the method of "Quick Error Detection" is focused on eliminating the error detection latency during post-silicon validation. The test patterns are obtained by transformation of existing post-silicon validation tests and it is a trade-off of error detection latency, validation coverage and hardware/software overhead.

They present two families of quick error detection transformation. The first transformation is called *error detection by duplicated instructions for validation*. It is drawn and extended from the EDDI (error detection by duplicated instruction) in the fault-tolerant computing. As illustrated in Fig 2.8, each block of instructions is duplicated and a checker is inserted to compare the results from the original block and the duplicated block.

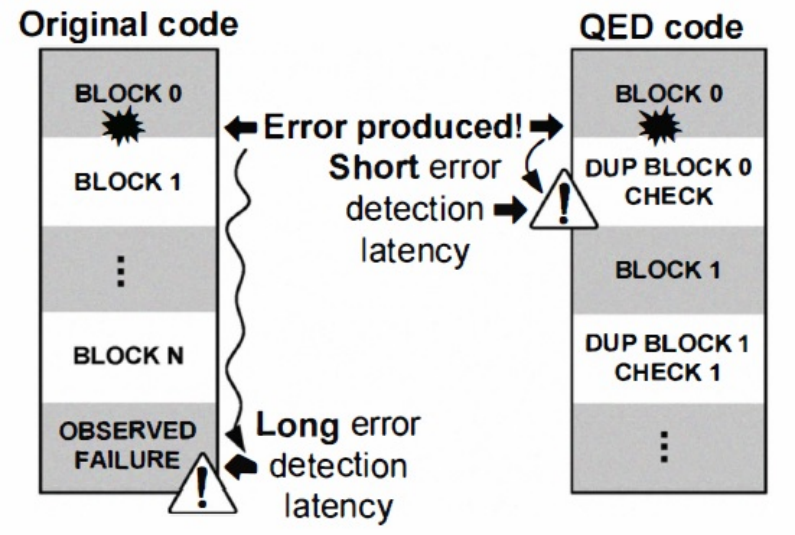

Figure 2.8: Error detection by duplicated instructions for validation [9]

In the process of post-silicon validation, general-purpose registers and

memory space are divided equally for original instructions and duplicated instructions. They are identically initialized to the same state. So in the bug-free system, original block and duplicated block perform the same operations and obtain the same results. Any mismatch between the results indicates an error.

The second transformation is called *redundant multi-threading for validation*. The idea is also inspired by fault-tolerant computing. It is similar to the first transformation in duplicating instructions. But it runs original instructions and duplicated instructions in different threads, and the two threads can be simultaneously executed on different cores.

In the process of post-silicon validation, two threads, main thread and check thread, are created. At the end of a block, main thread is responsible for sending out results while the check thread is designed to receive the main thread's results and compare them against its own results. Data transmission can be implemented by FIFO in either hardware approach or software approach.

From their experiments, the method of quick error detection significantly reduces error detection latencies by six orders of magnitude, from billions of cycles to a few thousand cycles or less.

2. ISA diversity

According to [10], a method of self-checking is proposed to accelerate post-silicon validation of microprocessors. Four major ISAs, i.e. ARM, MIPS,

PowerPC and x86 are analyzed. And they find out that more than three quarters of the instructions can be replaced with equivalent instructions. With inherent diversity in mind, they propose a method to exploit equivalent instructions and compare the results from the original response and equivalent response. Their approach aims at reaching the same results with the same input data but by different instructions that activate different logic paths in the processor.

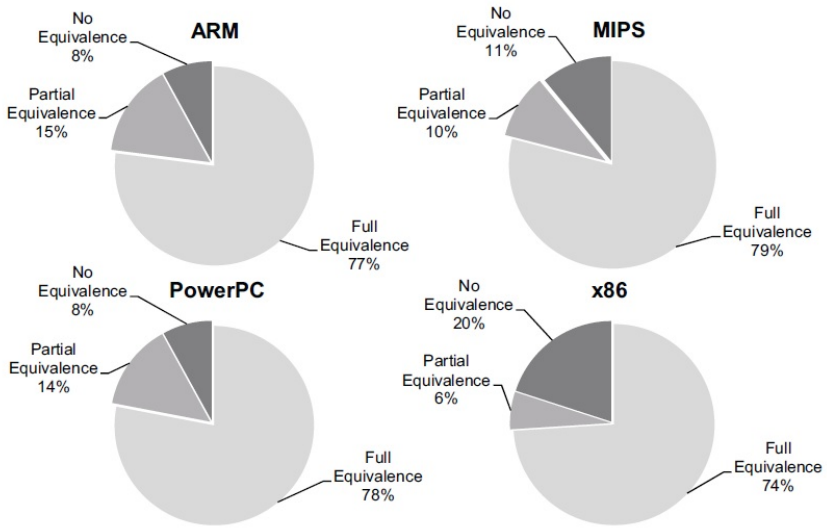

Figure 2.9: Diversity statistics of four popular ISAs [10]

The instructions of the ARM, MIPS, PowerPC and x86 are classified in three categories: full equivalence, partial equivalence and no equivalence. For the full equivalence category, there are always one or more equivalent ways to realize the same function. For the partial equivalence category, some instructions cannot be realized differently due to different modes or inherent

loss of accuracy. For the no equivalence category, there are no equivalent instructions to fulfill the same function.

There are in general four steps for the proposed self-checking method in post-silicon validation.

a. Generation of the ISA diversity database.

Instructions are classified and identified with equivalent instructions. It is better to have knowledge of the underlying architecture so as to classify the instructions that activate different hardware blocks.

b. Generation of enhanced random instruction tests.

Equivalent test patterns are created based on the ISA diversity database. When there are more than one equivalent instructions for the original instruction, a random equivalent instruction is selected. When there are not equivalent instructions, the original instructions are simply duplicated as equivalent instructions. Also a checking code is generated to compare the results of original test patterns and equivalent test patterns.

c. Hardware replay mechanism.

The hardware records the failing comparisons and pins the execution points of mismatches. When a mismatch is detected, the hardware mechanism allows replay of the test patterns by replacing the execution of the offending instruction with its equivalent instruction.

d. Post-processing.

Validation data from the hardware mechanism is used for clustering of failing nodes.

From their experiment, the method is able to detect all injected bugs. Further, the proposed method accelerates the process of post-silicon validation in terms of random test patterns.

3. Reversi

According to [11], they propose a test pattern generator called *reversi*. Given a program, it can be used to generate a random reversible program which has the identical initial state and final state. The benefit of identical initial state and final state is that there is no need in computing the final state based on the initial state using simulation. As the simulation of a golden model is several orders of magnitude slower than the hardware execution, the reversi-based method eliminates the trouble in computing known correct results.

The idea of reversi comes from the fact that many instructions in a processor's ISA have counterparts. Some instructions without their counterparts are called non-reversible instructions. For example, shift left and shift right operations cause some of the data bits to be lost. In order to restore them, the lost bits need to be stored in memory and retrieved in the reverse process.

From their experiment, they claim reversi handles all types of instructions. Since there is no architectural simulation step, reversi is able to generate and

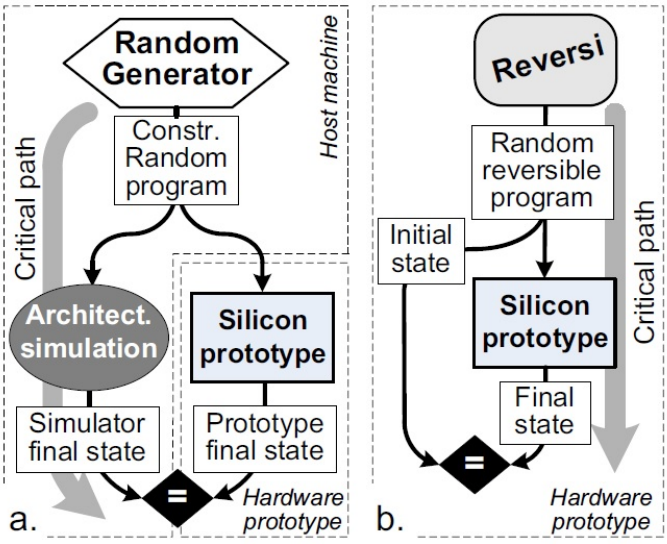

Figure 2.10: A typical post-silicon validation flow vs. a reversi-based flow [11]

run tests 20X faster than tools based on traditional post-silicon validation flow.

2.3 Conclusion

In this chapter, a list of ideas and works is reviewed towards the solution for post-silicon validation. There are methods of bridging the gap between pre-silicon verification and post-silicon validation, architectural level real-time record and off-chip post-process and instruction level techniques.

The above researches boost the progress of post-silicon validation and inspire us in working out this thesis work. In this thesis, the idea of functional core and validating core group comes from [5, 6, 8] while the idea of variance

in instructions comes from [9, 10, 11].

Chapter 3

Network on chip platform

3.1 Network on chip platform

The NOC(network on chip) platform in this thesis is derived from the mini-NOC project [22] based on SystemC [20] Model. It consists of 16 mMIPS cores [47, 48] and a torus network with E-cube routing.

3.1.1 IP core

IP core takes the role of computing and is one major part of the NOC platform. There are several major components inside the core.

1. mMIPS core: a simplified MIPS processor

This is a simplified MIPS [48] processor. It has pipeline and reduced ISA. Instructions used in mMIPS [47] core are shown below:

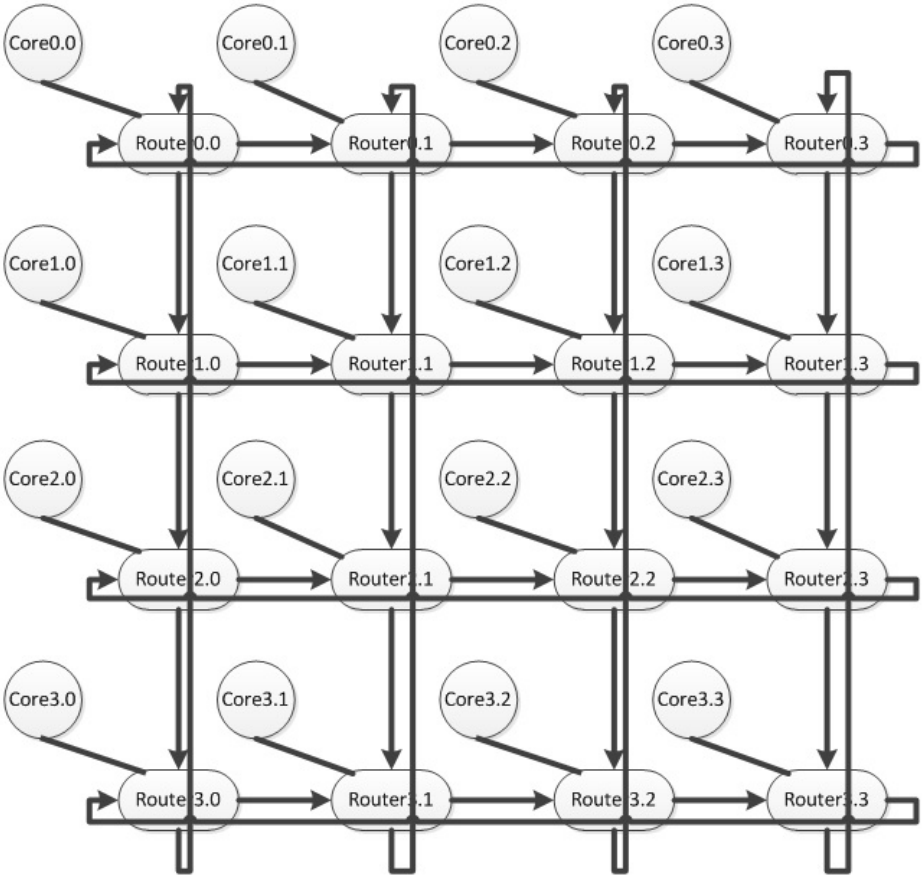

Figure 3.1: An overview of NOC platform

- addiu addu subu
- and andi or ori xor xori
- beq bne
- jal jalr jr j
- lb lw sb sw

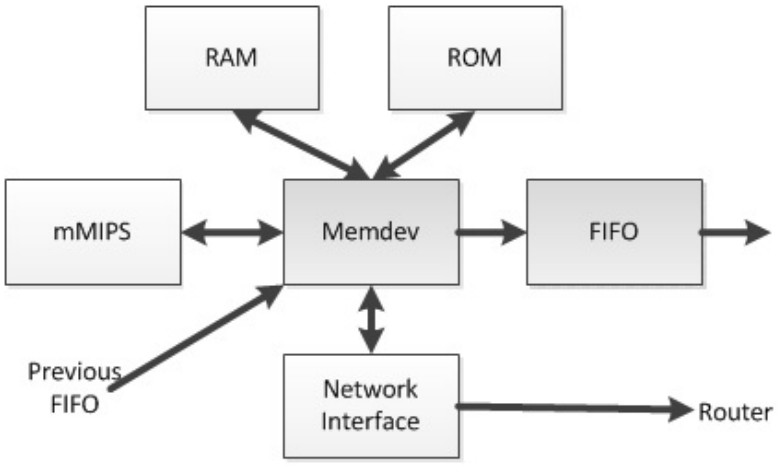

Figure 3.2: An overview of core inside the NOC platform

- lui
- slti sltiu slt sltu
- sll sra srl(1, 2, 8 bits)

Because of the reduced ISA, some of the operations are completed in software [12].

- All floating point operations
- Multiply, divide, module
- Variable distance shift
- Partial-word operation

2. RAM and ROM: data cache and instructions cache

There are two 16K memories used in the core. One is used for instruction cache as the ROM memory and the other is used for data cache as the RAM memory. Address mapping is shown in Fig 3.3. We can see from Fig 3.3 that not all the RAM memory is used for data cache and part of RAM memory, that is 4K, is reserved for debugging purpose and user data storage.

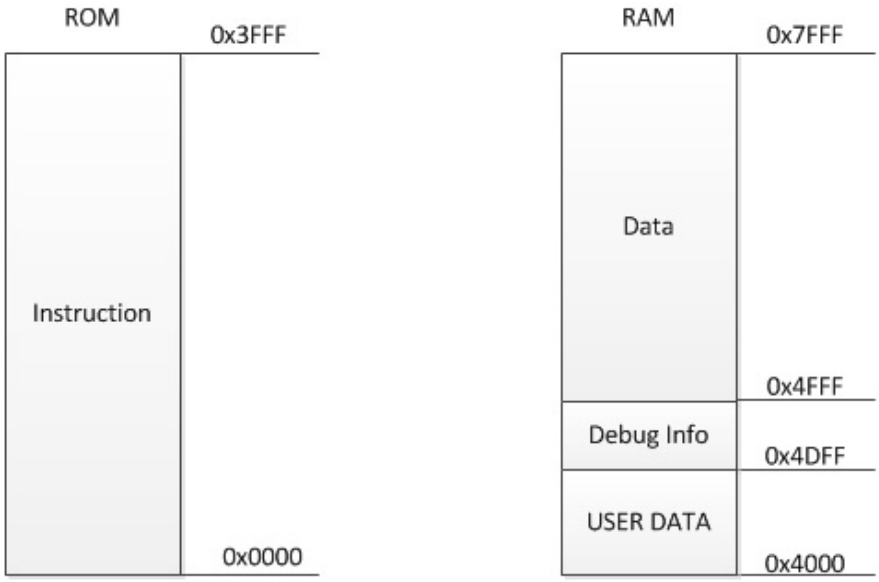

Figure 3.3: Memory layout for core in NOC platform [12]

- 0x0000 - 0x3FFFF 16K Instructions Cache
- 0x4000 - 0x4EFF User Data
- 0x4F00 - 0x4FFF Debug Info Block
- 0x5000 - 0x7FFF 12 K Data Cache

3. FIFO: transfering dump data

4k FIFO is inserted between adjacent cores in each x-dimension. Usage of FIFO will be discussed together with pair of functional core and validation core.

4. Memdev module: memory access

Memdev module communicates with NI (Network interface) using two addresses 0x8000000 and 0x80000004. Address 0x80000000 is used for data read and write while address 0x80000004 is used for NI control.

NI control word is modified by MIPS assembler in terms of load/store operations. So as to communicate between cores, C level primitives sc_nw_send(), sc_ff_send() and sc_received() are used to send and receive data. Details will be discussed in the software chapter.

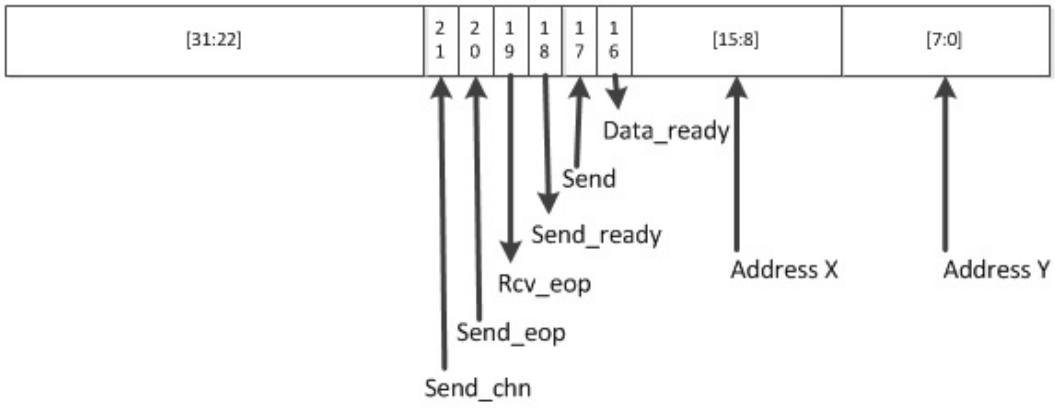

Figure 3.4: Meaning of NI control word

Bit 0 to bit 21 have been assigned meaning while bit 22 to bit 31 are reserved for no purpose.

The status bits include:

- data_ready (bit 16): data has been received and ready for read.
- send_ready (bit 18): previous data has been sent out and be ready for new data.
- rcv_eop (bit 19): if both this bit and data_ready are active, it is the last data to read.

The control bits include:

- address X/Y (bits 0-15): destination address of the packet (X distance - bits [15:8], Y distance - bits [7:0]).
- send (bit 17): written data is sent out when this bit is active.
- send_eop(bit 20): if both this bit and send are active, it is the last data to send out.
- Send_chn(bit 21): data is switched to network if this bit is inactive and to FIFO if active.

5. Network Interface: communicating between core and router

NI is used for sending and receiving packets through the network. The length of packets can be an arbitrary multiples of 32 bits. Before sending a packet, the 32 bit data is split into three flits, that is, head flit, data flit and trailer flit. In the process of receiving packets, the 32 bit data is reconstructed

by collecting three flits. The processes of packet sending and receiving are completed in two independent blocks of design. As a result, NI is able to send and receive data simultaneously.

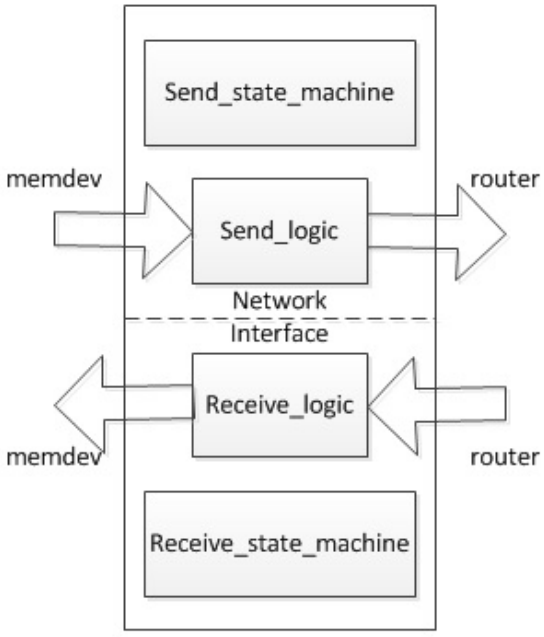

Figure 3.5: Network interface

A 32-bit packet is split in three flits:

Header flit : With a 2-bit header marker, 01 and relative destination address, 8-bit x relative address and 8 bit y relative address.

Data flit: With a 2-bit data marker, 00 and higher 16-bit data.

Trailer flit: With a 2-bit trailer marker, 10 and lower 16-bit data.

The procedure to send a packet.

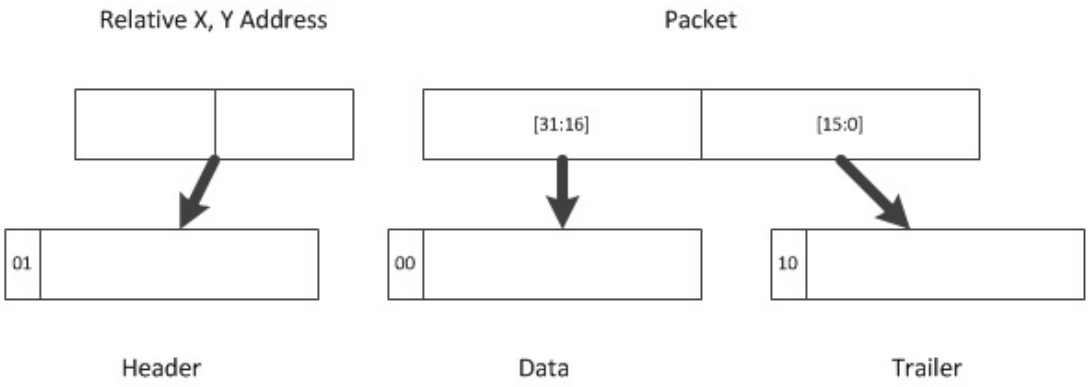

Figure 3.6: A packet and its flits

1. Wait for send_ready to be high.

2. Write the packet on the data bus.

3. write destination address.

4. Trigger send signal and trigger send_eop if it is the last packet.

5. if there are more packets to be sent, repeat from step 1.

The procedure to receive a packet.

1. Wait until data_rdy signal is high.

2. Read packet from the data bus.

3. Activate read and continue receiving new packet when buffer is read. it is the last packet if receive_eop is high.

3.1.2 Network

The network used in this thesis is a torus network. It connects all the routers and provides the channels for communication between cores.

1. Router

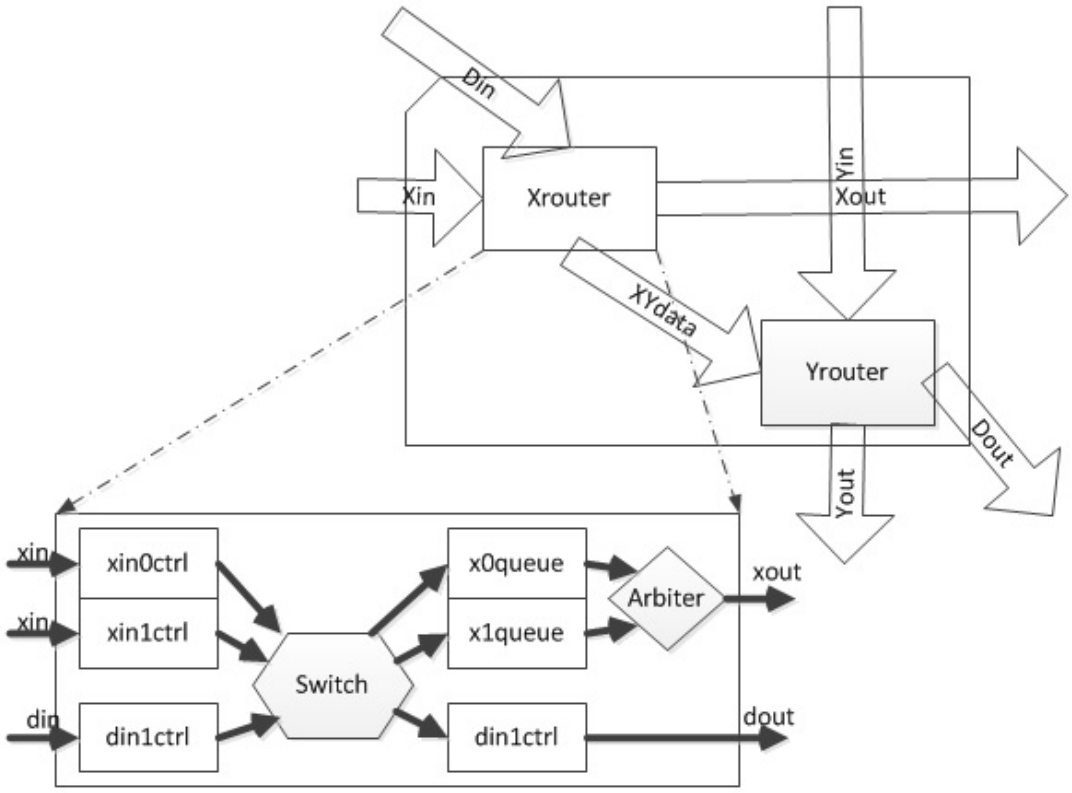

Figure 3.7: Block diagram for router

The X sub-router receives data from the NI on input data bus, din. This data is forwarded to the x output, which is connected to the x sub-router of the adjacent router. The data travels through x sub-routers until it reaches

the destination x address. Then it is forwarded to the d output of the x sub-router, which is connected to the d input of the y sub-router. And the x address is stripped off the packet and replaced with the y address. Further, the packet travels to the y address along the y dimension. When it reaches the destination y address, it is again forwarded to the data output bus, dout, which is connected to the input of the NI of destination core.

2. Link

Links between routers are implemented with 18-bit wide bus and it fits 16-bit data and 2-bit flit marker. There are two sets of hand-shake signals designed for virtual channel. According to Dally and Seitz[49, 50], a routing function is deadlock-free if and only if there is no cycle in its channel dependency graph. To break circular dependencies and prevent deadlocks, the packet moves from one virtual channel to the other one when it wraps around. This special design ensures deadlock-free communication.

3. Network topology

The network used in mini-NOC is a torus [51] network. Torus topology is a network, inside which a router is connected to its immediate adjacent router in both directions. At edges of the network, the connections wrap around and connect the last router in the given dimension with the first router.

4. Routing method

In the E-cube routing, each packet in the network is first routed along the X dimension, until it reaches a router with the X address equal to the packets destination X address. Then it starts to move in the Y dimension until it reaches the destination Y address. Since connections in the network are unidirectional, the packet can only travel in the direction of increasing addresses and wrap around at the edge of the network if necessary.

E-cube routing is a deterministic routing algorithm and routing path from one node to the other node is fixed. It is simple but does not perform well in terms of network congestion or latency.

3.2 Proposal for post-silicon validation

In recent NOC platforms, a great number of cores are integrated into the NOC platform. It is possible to reuse some of the cores for the purpose of post-silicon validation.

The basic idea is to divide the cores in the NOC platform into two groups. One group is for the purpose of functional execution, and the other group is for the purpose of validating functional processes. In functional mode, both groups are running functional programs. But in the validation mode, validating group is reconfigured to perform the tasks of validating functional cores. Dumped data are not transmitted outside the chip, instead, it is transmitted inside the chip through dedicated FIFO channels.

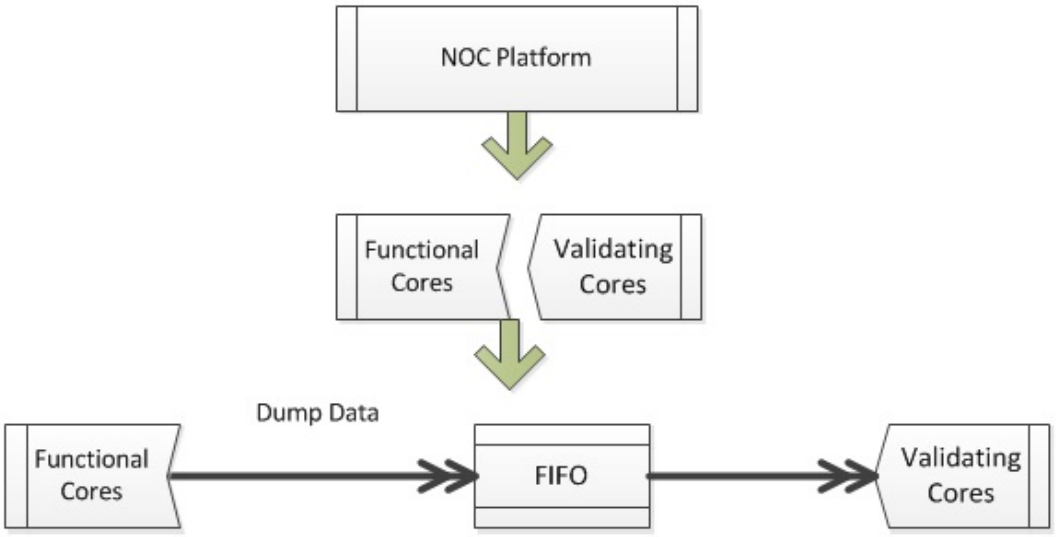

Figure 3.8: Division of functional cores and validating cores.

3.2.1 Pair of functional core and validating core

For each pair of cores in the NOC platform, it is called a FVP (Functional core and Validation core Pair). Inside the FVP, a FIFO module is inserted to serve as the transmission channel. At the read port of the core-router channel on the side of core, it is switched. In the functional mode, it switches to connect router for functional data communication. While in the validation mode, it switches to connect FIFO for dump data communication.

In the torus network, FVP is not fixed to cores. Each core can be configured to be either a functional core or a validating core. In Fig 3.9, the right two cores are paired. In this pair, the left core is considered as a functional core and the right core is considered as a validating core. If configured properly, the left two cores can also be paired as a FVP.

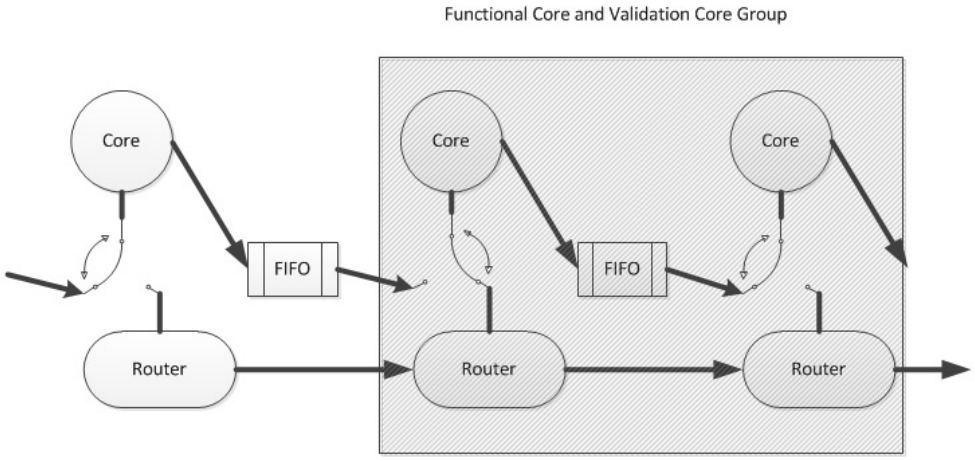

Figure 3.9: Pair of functional core and validating core.

With the pairing in the NOC platform, the only thing we need to be careful is to double the steps when transmitting data through the routers. The router for validation core is still working and there are two routers in each FVP.

Also FIFO is a limited size storage, and the goal is that FIFO never goes full when both functional cores and validation cores are working in the validation mode.

3.2.2 Procedure for post-silicon validation in FVP

1. Identify and configure the functional core and validation core.

2. Functional core performs functional task and dump data into FIFO in the FVP.

3. Validating core reads dump data and performs validation.

4. The above steps continue until all functional tasks are completed.

Chapter 4

Post-silicon validation strategy

4.1 Software infrastructure for network on chip platform

4.1.1 LCC compiler

LCC[52, 53] is a retargetable C compiler. The target of a C compiler is the processor when it generates assembly instructions. The LCC compiler used in the NOC platform has been ported to the mMIPS architecture with a reduced instruction set, 16k bytes RAM and 16k bytes ROM.

With the help of LCC compiler, it is capable to generate mMIPS assembly code that is used in this work later. Also it helps generate binary code which can be loaded into the mMPIS NOC platform for simulation.

4.1.2 C libraries for network on chip platform

To ease the development of the c level applications, two libraries provided in the mMIPS NOC project [47] are modified to meet our needs.

1. Library stdcomm

Library stdcomm provides the primitive C function for data communication in the NOC platform. For example, sc_nw_send(), sc_ff_send(), sc_nw_send_word(), sc_ff_send_word(), sc_receive() and sc_receive_word() are the basic functions in the library.

The files stdcomm.h and stdcomm.c include the interface and implementation of data communication library for the NOC platform. The difference between sc_nw_send() and sc_ff_send() is that they are used to send data to different places. Function sc_nw_send() sends data to the network and data is transmitted via routers before it is received by another core. Function sc_ff_send() sends data to a FIFO and data is transmitted via the FIFO before it is received by the paired validating core.

2. Library mtools

Library mtools provides the primitive C functions for debugging with the memory. The mtools library includes debugging and implemented functions that are dedicated to the mMIPS core.

In the debugging mode, data in the memory can be copied to the debug information block in the RAM at runtime and accessed when programs are completed.

4.1.3 Communication primitives

1. sc_nw_send() and sc_ff_send()

sc_nw_send() is used to send data through the network and reach the other functional core. It has three parameters:

- relative destination address
- data buffer
- number of bytes to be sent

Relative addressing is used in the NOC platform. In the address parameter, bits [15:9] stands for the x distance while bits [7:0] stands for the y distance. In the torus network, packet wraps around at each edge. Theoretically packet can be sent to any nodes in the network.

sc_ff_send() is used to send dump data through the FIFO from a functional core to its validating core in a functional core and validating core pair. It has two parameters:

- data buffer
- number of bytes to be sent

To distinguish these two functions in the hardware, a status bit in the network interface control word is switched to identify whether the data is sent to the network or the FIFO. The assembler controls the hardware switch by configuring the send_chn bit.

2. sc_receive()

There is only one receive primitive because the receiving channel is fixed. A functional core can only receive data from the network while a validating core can only receive data from the FIFO.

sc_receive() has two parameters:

- data buffer
- number of bytes to be received

4.2 Post-silicon validation strategy

In this thesis, the method used in the NOC platform is to create variances in the programs running on a pair of functional core and validating core, and to compare the results between their outputs. Both programs follow the same specification and the results are expected to be exactly the same.

4.2.1 Metric

Here we define a measuring parameter, level of variance. It describes how different they are in the assembly level between the functional program and the validating program.

Frames are defined as part of a program that has its input data and output data transmitted through FIFO and validated in the validating core. In terms of validation, we are pursuing the goal of coverage by increasing the level of

variance. In general, the higher coverage we expect, the higher percentage is the variance, and the more confidence we gain from the validation process but the more difficult to trace errors inside the cores.

4.2.2 Method of variance validating in a frame

1. Instruction level validation

The first and the most simplest one is to validate the program by each assembly line. For an addition operation, for example

$$sum = a + b$$

in the corresponding validation program, it is running as

$$a = sum - b$$

Each line of code is considered to be a frame. Both input and output data are transmitted through the FIFO so that validation cores are able to verify the result.

There may be more than one way to generate a reverse validation program. In the above example, the reverse addition operation in the validation program can be rewritten as

$$b = sum - a$$

It is similar to the previous validation program. But they are not identical in hardware execution. As a result they are considered to be different validation programs. Sometimes, we need more than one validation program to gain high level of variance.

Data transmission:

Inputs and outputs of each instruction need to be transmitted through the FIFO.

2. Tightly coupled frames

Obviously, the above method is not effective because a huge amount of data are transmitted through the FIFO for a relatively large program. In such a situation, tightly coupled assembly codes are grouped as a frame and taken as a whole piece. Intermediate variables are not transmitted in the FIFO. Only input and output data are transmitted.

One simple example is as following. Mathematic expression is

$$sum = a + b + c$$

In assembly code,

```
REG[0] = a;
REG[1] = b;
REG[2] = REG[0] + REG[1];
REG[3] = c;
```

REG[4] = REG[2] + REG[3];

Variables a, b, c, sum are transmitted through the FIFO from a functional core to its corresponding validating core. In the register view, REG[0], REG[1], REG[3], REG[4] are transmitted while REG[2], as a temporary variable, is not transmitted. With such a scheme, the amount of data transmission is greatly reduced.

In validation program, there are several ways to mirror the same code. For example, we an use

$$c = sum - (a + b)$$

And the corresponding assembly code is shown below.

REG[0] = a;
REG[1] = b;
REG[2] = REG[0] + REG[1];
REG[3] = sum;
REG[4] = REG[3] - REG[2];

In the validation program, we can see the functional program and the validation program share a small segment(i.e. the first three statements) of the assembly code. Thus, the level of variance is greatly decreased. In order to increase the level of variance, we need to create a more different validation program that has less shared assembly codes.

For a second example, we have

$$a = sum - b - c$$

Apparently, there can be other alternative forms:

$$b = sum - a - c$$

$$c = sum - a - b$$

Sometimes, to gain a high level of variance, we need to use all of them or part of them in the validation program.

Data transmission:

Inputs and outputs of each frame need to be transmitted through the FIFO.

3. Equivalent functional program and validation program

In some problem-resolving programs, there are two or more different ways for resolving the problem and validating the problem. In such a situation, it is appropriate to implement different ways of solving the problem, develop codes for them and assign the codes to the functional cores and validating cores.

One example is as following:

$$Y = AX + B$$

The above mathematical expression can be rewritten as

$$X = (Y - B)/A$$

or

$$X = Y/A - B/A$$

Here A and B can be taken as both constant scalars and matrices. Let us take constant scalars as an example, since matrices can be more complex in presenting the basic idea. The corresponding assembly code for

$$Y = AX + B$$

is :

```
REG[0] = X;
REG[1] = A;
REG[2] = REG[0] * REG[1] ;
REG[3] = B;
REG[4] = REG[2] + REG[3] ;
```

Data transmission:

Here we take A and B as constants and they can be precomputed and hard-coded in the program. Hence they are not transmitted through the

FIFO. However, X and Y must be transmitted through the FIFO, such that

$$X = Y/A - B/A$$

can be verified in the validating core.

4. Verify the attribute of output

For some programs, they are designed to created output data with a specified attribute. In this case, we do not need to create validation programs corresponding to the functional program. It only requires validation of the attribute of the outputs.

We have the following example to show the scenario. In a sorting program, it is given a list of numbers $A_1, A_2, \ldots, A_n$. After sorting, the output data are expected to be in a specific order. The complexity for bubble sorting is O(n^2). While in the validation program, it only requires to check the list of numbers sorted in a specific order. Thus, the complexity for the validating program is O(n). This is preferable because the validation program can be finished earlier than the functional program, such that the FIFO will not be full due to different traffic flows.

Data Transmission:

Outputs of the sorting program must be entered into the FIFO and transmitted to the validation program.

4.2.3 Creation of validation programs

The following shows steps in creating validation programs:

- Create frames from functional program
- Choose the strategy of variance creation for each frame
- Define the transmission data and assign ID for each frame function
- Create validation program for each frame and tag it with ID
- Insert data transmission codes into both functional program and validation program

For some programs involving memory storage, there is no way for the validation program to access the same location of memory in the progress of functional program. In such a situation, a copy of memory is created and mainstream program between frames is kept unchanged. That is to say, all memory operations are reproduced in the validation program together with frame validation program. This case is discussed later in the chapter.

4.2.4 Concerns in validation program design

1. Data transmission and limited FIFO size

Input data and output data for each frame are transmitted through the FIFO and sent to all validation cores. Functional cores are producing data

and validation cores are consuming data. We do expect dump data do not stay in a FIFO for a long time such that the FIFO will not be full. It is better that validation cores are consuming data faster than functional cores are producing data. As all of these are running in a real chip, there is no scheme to stop the process and wait for the FIFO not to be full. We have to be careful of FIFO operations, and thus a validation program needs to be designed in an appropriate way so as to eliminate the possibility of FIFO being full.

2. Memory reproduction

In some memory related programs, data are stored a place in the memory. In order to run both the functional and validation program, a copy of initial memory need to reproduced for the validation programs. Without memory issue, validation program just needs to run the validation process for each frames. With memory issue, validation program has to recreate all the functional program structure and embed validation process inside.

In this example, a block of memory is reserved for data storage, and changes of data are directly made to memory blocks. Usually amount of data stored in memory is too large and it is not appropriate to transmit them through the FIFO. Hence, changes in data blocks are verified by memory comparison.

3. Data accuracy

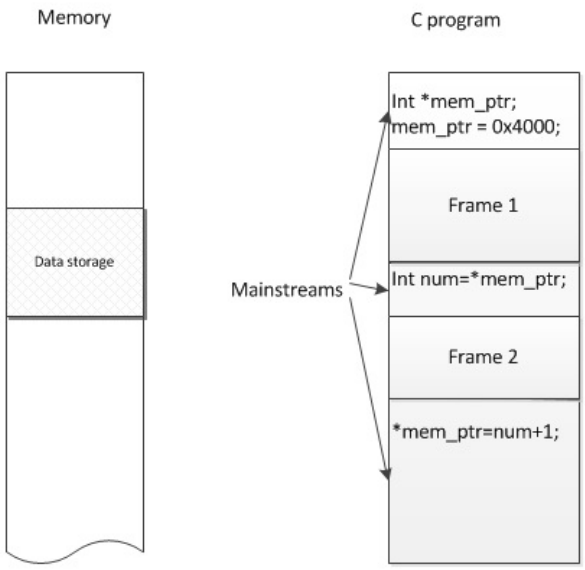

Figure 4.1: Example for data changes in memory

For a pair of functional program and validation program, theoretically they are approaching the same result in two different ways. While in real-world program executions, rounding may lead to different results by two different approaches. Here, there are two solutions for this problem. First, we keep all the operations that have the rounding issue without being changed. That is, these instructions are directly copied to the validating program. Second, we approximate the result and determine whether the functional program and the validation program reach the same result.

4.2.5 Process for post-silicon validation

The steps for the proposed post-silicon validation is shown below and also shown in Figure 4.2.

1. Create frames for each functional benchmark program.

2. Generate the corresponding validation program discussed in previous section based on frames identified from the functional programs.

3. Compile both functional program and validation program into binary files.

4. Load binary programs into the NOC platform.

5. Start simulation on the NOC platform and get the result.

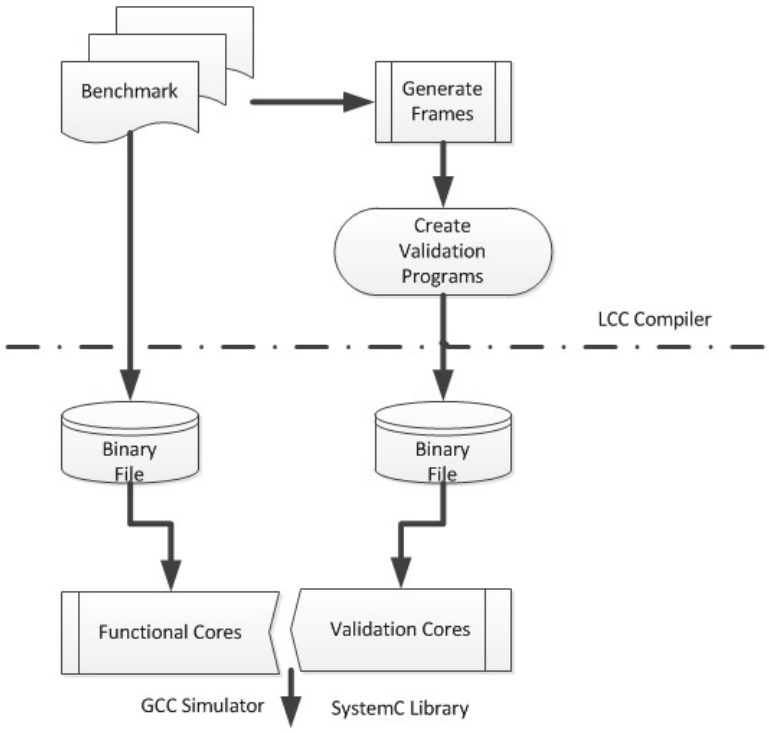

Figure 4.2: Process for post-silicon validation

The method developed in this thesis helps find out the hardware bugs and functional bugs. Compared to the instruction level debug method, it is able to catch complex system bugs. Further it greatly reduces the size of dump data. Thus, on-chip validation can be carried out instead of using off-chip analysis.

What is more important: the variance between functional program and validation program helps find out hidden problems that can not be found in the instruction level debug method, where both functional core and validation core run the same copy of program. It does not fail when the same erroneous hardware creates the same result. But in variance validating method, it does fail because erroneous hardware can not reach the same result in two different ways.

4.2.6 Process for reliable system

The method used in this work can also be applied to reliable system design with a little modification. In the post-silicon validation mode, a functional core is performing expected functions while its validating core in the pair is completing the task of validation. The pair of functional core and validating core is actually validating each other in terms of functions. This is useful in reliable system design in which redundant cores are used to reproduce the same result. Here the validating core can be considered as a redundant core of the functional core, and hence the pair of functional core and validating core is working jointly as a reliable system.

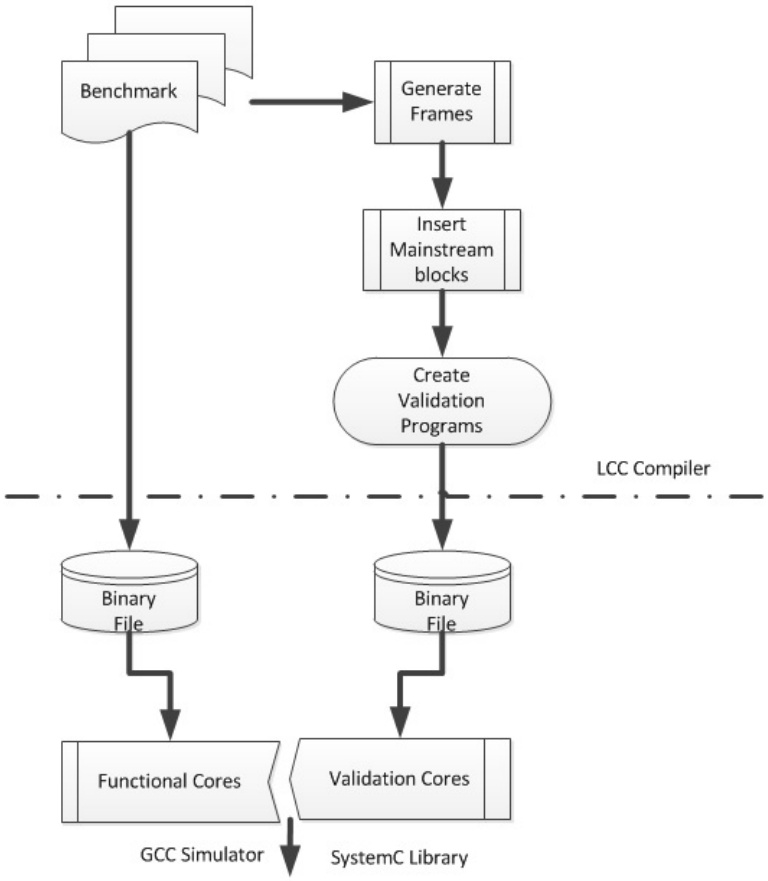

Figure 4.3: Process for reliable system

Changes in the validation program must be done before it can be used in the reliable system. Mainstream blocks need to be kept unchanged together with frames. So validation program is reconstructed and then functioning as a redundant core. Compared to system running same-design cores with the same program, the system that is running cores with different programs is more likely to catch bugs and hence more reliable.

Chapter 5

Case study

The well-known spafilter, spatial filter, is an application used in image processing. Here it is taken as an example to show how variance can be created from a functional program and how to generate its corresponding validation program.

5.1 Mathematical representation

Assume a weight matrix W(i,j) is given as follows:

$$\begin{pmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{pmatrix}$$

The weight matrix is designed for shift operations. If a pixel is multiplied by 2, it can be easily implemented by shifting the pixel data left by 1 bit.

It is shifted left by 2 bits when the pixel is multiplied by 4. In total, the weight matrix sums up to 16, which is the same as shifting right by 4 bits in computing the final pixel value. The relationship between an output pixel PixelOut(x,y) and its related input pixels is as follows:

$$PixelOut(x,y) = \frac{\sum_{\substack{-1\leq i\leq 1\\ -1\leq j\leq 1}} W(i,j) \times PixelIn(x+i, y+j)}{\sum_{\substack{-1\leq i\leq 1\\ -1\leq j\leq 1}} W(i,j)} \tag{5.1}$$

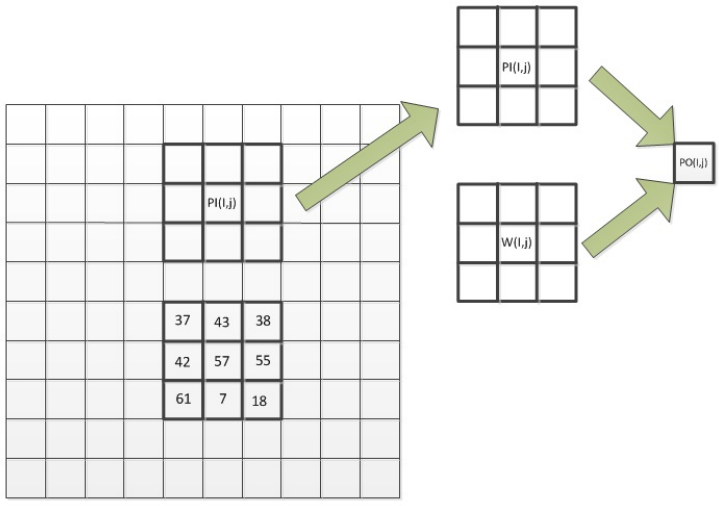

Figure 5.1: Illustration of spatial filter in a two-dimension image

As an example in Figure 5.1, for the 9 pixels shown in the lower 3×3 matrix, the output pixel should be calculated as follows:

PixelOut : $(37+43\times2+38+42\times2+57\times4+55\times2+61+7\times2+18)/16 = 42$

And the position where the pixel value is 57 will be changed to 42 in the output image.

5.2 C level program

Here is the body of c level implementation for spafilter. The input pixels are stored in an one-dimension array and the output pixels are stored in the same way. Suppose the size of the image is N × N, and each new output pixel value is calculated by nested loops.

```
for (i=1; i<N-1; i++){
  for (j=1; j<N-1;j++){
    sum = 0;
    sum += (framein[i*N+j-1] << 1) ;
    sum += (framein[i*N+j]   << 2) ;
    sum += (framein[i*N+j+1] << 1) ;
    sum += (framein[(i-1)*N +j-1] ) ;
    sum += (framein[(i-1)*N +j] << 1) ;
    sum += (framein[(i-1)*N +j+1] ) ;
    sum += (framein[(i+1)*N +j-1] ) ;
    sum += (framein[(i+1)*N +j] << 1) ;
    sum += (framein[(i+1)*N +j+1] ) ;
    frameout[i*N+j] = ( sum >>4 );
  }
}
```

5.3 Frames analysis

By observing the structure of the above program, the body of the loop can be taken as a whole frame, and the validation program can be created by reserving the data flow. Here addressing of array framein is not counted in and it is assumed the content of framein with an index can be available after loading it from the memory to a register.

Operations of shift-left can be guaranteed to be reversed; otherwise there is a problem of overflow regarding the filtering algorithm. One more special notice on the shift-right operation: it cannot be reversed since the precision is lost in this operation. We cannot reverse the operation by the operation of shift-left. Details will be discussed later in this case study.

In summary, all operations in the loop body except the shift-right operation are considered to be a single frame.

5.4 Data transmission in FIFO

Input data are those that are first loaded from the memory to registers. Here, all elements in framein are taken as the input data. Output data are those that are later stored from registers to the memory. Here, the element of frameout[i*N+j] should be taken as the output data. Though it is not due to the loss of precision in the shift-right operation, instead, sum is taken as the output data in this case study.

Identification is generated by the variables i and j so as to keep track of

the steps of the loop. Here variables i and j are concatenated to create the identification tag.

5.5 Data flow in validation program

The following is an example of reversing each operation. Data sum is initialized to the output data stored in the FIFO. Each element of framein comes from the input data stored in the FIFO, and the final result of sum should be 0 if the validation program correctly verifies the functional program.

```
sum -= (framein[i*N+j-1] << 1) ;
sum -= (framein[i*N+j]   << 2) ;
sum -= (framein[i*N+j+1] << 1) ;
sum -= (framein[(i-1)*N +j-1] ) ;
sum -= (framein[(i-1)*N +j] << 1) ;
sum -= (framein[(i-1)*N +j+1] ) ;
sum -= (framein[(i+1)*N +j-1] ) ;
sum -= (framein[(i+1)*N +j] << 1) ;
sum -= (framein[(i+1)*N +j+1] ) ;
```

There is still one further improvement based on shift-left operations. Each shift-left operation above is kept the same as the one in the functional program, and variance can be created to make the functional program and validation program as different as possible. With a little change, a shift-left operation can be rewritten by another shift-right operation and the new

data flow is as follows.

```
sum -= (framein[i*N+j-1] << 1) ;
//sum -= (framein[i*N+j]   << 2) ;
sum -= (framein[i*N+j+1] << 1) ;
sum -= (framein[(i-1)*N +j-1] ) ;
sum -= (framein[(i-1)*N +j] << 1) ;
sum -= (framein[(i-1)*N +j+1] ) ;
sum -= (framein[(i+1)*N +j-1] ) ;
sum -= (framein[(i+1)*N +j] << 1) ;
sum -= (framein[(i+1)*N +j+1] ) ;
sum = (sum >> 2) ;
sum -= framein[i*N+j];
```

In the above program, the line

```
//sum -= (framein[i*N+j]   << 2) ;
```

is commented out and at the end of subtraction, variable sum equals to $(framein[i * N + j] << 2)$. At the end of the program, if sum is shifted right by 2 bits, it equals to $framein[i * N + j]$. Finally with a subtraction of $framein[i * N + j]$, we have sum equals 0 at the end.

There are several other ways the program can be rewritten. Here, only one example is given, and all these programs can be used as a validation program.

5.6 Validation program

The validation program is following data flow of the above example with minor changes.

```
// fifo[0] is id
// fifo[1] is output data
// fifo[2], ... , fifo[10] are input data

result = fifo[1];
result -= (fifo[2] << 1) ;
result -= (fifo[4] << 1) ;
result -= (fifo[5] ) ;
result -= (fifo[6] << 1) ;
result -= (fifo[7] ) ;
result -= (fifo[8] ) ;
result -= (fifo[9] << 1) ;
result -= (fifo[10] ) ;
result = (result >>2 ) ;
result -= fifo[3] ;
```

5.7 Tuned C level and assembly level program

Here is an apple to apple comparison between C level program and assembly level program. To improve the readability, several changes are made without changes to functions.

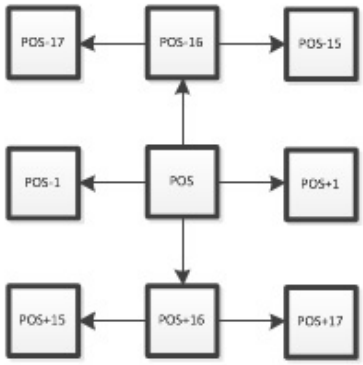

Figure 5.2: Relative addressing to the central position

All positions of pixels in the following program are relative to the central position as shown in Figure 5.2. This change is made to reflect optimization by the compiler. C level functional program is as following:

```
int spafilter(unsigned char *framein, unsigned char *frameout ) {
    int i, j;
    unsigned int sum;
    int pos;
```

```
    for (i=1; i<=14; i++){
        for (j=1; j<=14; j++){
            pos = (i << 4);
            pos += j;
            sum = 0;
            sum += (framein[pos] << 2) ;
            sum += (framein[pos-1] << 1) ;
            sum += (framein[pos+1] << 1) ;
            sum += (framein[pos-16] << 1) ;
            sum += (framein[pos+16] << 1) ;
            sum += (framein[pos-17] ) ;
            sum += (framein[pos-15] ) ;
            sum += (framein[pos+15] ) ;
            sum += (framein[pos+17] ) ;
            sum = sum >> 4;
            frameout[pos] = (char) (sum);
        }
    }
    return 0;
}
```

The corresponding assembly level functional program is as follows:

```
sw $18,16($sp)
```

```
    sw $19,20($sp)
    sw $20,24($sp)
    sw $21,28($sp)
    sw $22,32($sp)
    sw $23,36($sp)
    sw $31,40($sp)
    move $23,$4
    move $22,$5
    la $18,1
                                    ; loop of i
L.2:
    la $19,1
                                    ; loop of j
L.6:
```

; **frame starts from here**

```
    la $24,4                        ; specify shift bit 4
    sw $24,-4+56($sp)               ; 4 is stored
    move $4,$18
    move $5,$24
    jal __sll                       ; pos = i << 4
    move $20,$2
    addu $20,$20,$19                ; pos+=j
```

```
move $21,$0
addu $24,$20,$23
lb $24,($24)                ; load byte framein[pos]
sll $24, $24, 2             ; (framein[pos] << 2)
addu $21,$21,$24
la $24,1                    ; specify shit bit 1
sw $24,-8+56($sp)           ; 1 is stored
subu $15,$20,1              ; pos -1
addu $15,$15,$23
lb $4,($15)                 ; load byte framein[pos-1]
move $5,$24
jal __sll                   ; (framein[pos-1] << 1)
move $24,$2
addu $21,$21,$24            ; sum +=
la $24,1($20)
addu $24,$24,$23            ; pos + 1
lb $4,($24);
lw $24,-8+56($sp)
move $5,$24
jal __sll                   ; framein[pos+1] << 1
move $24,$2
addu $21,$21,$24            ; sum +=
subu $24,$20,16             ; pos - 16
```

```
addu $24,$24,$23
lb $4,($24);
lw $24,-8+56($sp)
move $5,$24
jal __sll                    ; framein[pos-16] << 1
move $24,$2
addu $21,$21,$24             ; sum +=
la $24,16($20)
addu $24,$24,$23             ; pos + 16
lb $4,($24);
lw $24,-8+56($sp)
move $5,$24
jal __sll                    ; framein[pos+16] << 1
move $24,$2
addu $21,$21,$24             ; sum +=
subu $24,$20,17              ; pos - 17
addu $24,$24,$23
lb $24,($24)
addu $21,$21,$24             ; sum += framein[pos-17]
subu $24,$20,15              ; pos - 15
addu $24,$24,$23
lb $24,($24)
addu $21,$21,$24             ; sum += framein[pos-15]
```

```
    la $24,15($20)
    addu $24,$24,$23
    lb $24,($24)
    addu $21,$21,$24            ; sum += framein[pos+15]
    la $24,17($20)
    addu $24,$24,$23
    lb $24,($24);
    addu $21,$21,$24            ; sum += framein[pos+17]
    move $4,$21
```

; **frame ends here**

```
    lw $24,-4+56($sp)           ; specify shift bit 4
    move $5,$24
    jal __srl                   ; sum = sum >> 4;
    move $21,$2
    addu $24,$20,$22
    move $15,$21
    sb $15,CVII4_int            ; frameout[pos] = (char) (sum);
    sb $15,($24)
L.3:
    la $18,1($18)
    la $24,14
    slt $30,$24,$18
```

```
        beq  $30,$0,L.2
        move  $2,$0
L.1:
        lw  $18,16($sp)
        lw  $19,20($sp)
        lw  $20,24($sp)
        lw  $21,28($sp)
        lw  $22,32($sp)
        lw  $23,36($sp)
        lw  $31,40($sp)
        addu  $sp,$sp,56
        j  $31
```

The C level validation program is below. This validation program breaks original shift-left-by-1-bit and shift-left-by-2-bit operations. Instead, shift-left-by-1-bit is replaced with two addition operations, and shift-left-by-2-bit is split into two shift-left-by-1-bit operations.

```
int spafilterVerify( unsigned char *fifo ) {
    unsigned int sum;

    sum = fifo[1];
    sum -= (fifo[3] << 1) ;
    sum -= (fifo[3] << 1) ;
```

```
    sum -= (fifo[2]) ;
    sum -= (fifo[2]) ;
    sum -= (fifo[4]) ;
    sum -= (fifo[4]) ;
    sum -= (fifo[6]) ;
    sum -= (fifo[6]) ;
    sum -= (fifo[5]) ;
    sum -= (fifo[7]) ;
    sum -= (fifo[9]) ;
    sum -= (fifo[9]) ;
    sum -= (fifo[8]) ;
    sum -= (fifo[10]) ;

    if (sum == 0) {
        return 0;
    } else {
        return 1;
    }
}
```

The corresponding assembly level validation program is given below.

```
    sw $22,16($sp)
    sw $23,20($sp)
```

```
sw $31,24($sp)
move $23,$4
la $24,1
sw $24,-8+40($sp)
lb $15,1($23)                    ; sum = fifo[1];
move $22,$15
la $15,3($23)
sw $15,-4+40($sp)
lb $4,($15)
move $5,$24
jal __sll                        ; fifo[3] << 1
move $24,$2
subu $22,$22,$24                 ; sum -=
lw $24,-4+40($sp)
lb $4,($24)
lw $24,-8+40($sp)
move $5,$24
jal __sll                        ; fifo[3] << 1
move $24,$2
subu $22,$22,$24                 ; sum -=
la $24,2($23)
lb $15,($24)
subu $22,$22,$15                 ; sum -= fifo[2]
```

```
lb $24,($24)
subu $22,$22,$24          ; sum -= fifo[2]
la $24,4($23)
lb $15,($24)
subu $22,$22,$15          ; sum -= fifo[4]
lb $24,($24)
subu $22,$22,$24          ; sum -= fifo[4]
la $24,6($23)
lb $15,($24)
subu $22,$22,$15          ; sum -= fifo[6]
lb $24,($24)
subu $22,$22,$24          ; sum -= fifo[6]
lb $24,5($23)
subu $22,$22,$24          ; sum -= fifo[5]
lb $24,7($23)
subu $22,$22,$24          ; sum -= fifo[7]
la $24,9($23)
lb $15,($24)
subu $22,$22,$15          ; sum -= fifo[9]
lb $24,($24)
subu $22,$22,$24          ; sum -= fifo[9]
lb $24,8($23)
subu $22,$22,$24          ; sum -= fifo[8]
```

```
        lb  $24,10($23)
        subu  $22,$22,$24               ; sum -= fifo[10]
        bne  $22,$0,L.2
        move  $2,$0
        b  L.1
L.2:
        la  $2,1
L.1:
        lw  $22,16($sp)
        lw  $23,20($sp)
        lw  $31,24($sp)
        addu  $sp,$sp,40
        j  $31
```

5.8 Optimization in functional program and impact on validation program

In the spatial filter program, when comparing the input pixels required for calculating two adjacent blocks PixelOut[x,y] and PixelOut[x+1,y], six out of nine total input pixels are shared. As a result, there is no need to read in shared pixels and only three new pixels are loaded each time.

Based on the optimization made in the functional program, corresponding

changes can be made on the side of the validation program. Data transmitted from the FIFO during the current round of validation should be kept and reused in the next round of validation. This can greatly reduce the data transmission time.

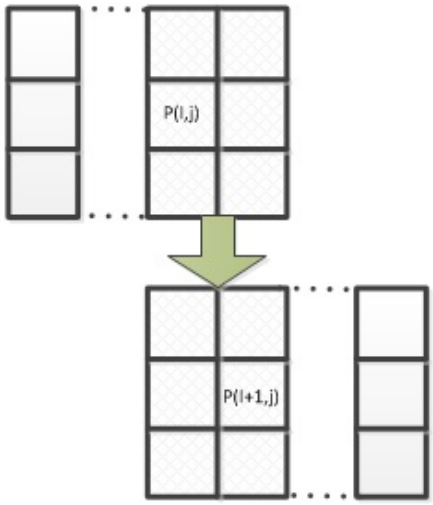

Figure 5.3: Shared pixels between P(x,y) and P(x+1,y)

5.9 Derivation for level of variance

LCC is used to translate a C level program into its assembly level program. And a perl script is used to count the number of lines in the assembly program.

A functional program is first compiled into its corresponding assembly level program and the number of lines, L1, is counted. Then, each frame that can be validated by a validation program is replaced by an empty function.

The new functional program is recompiled into an assembly level program and the number of lines, L2, is counted, too. Level of variance between a pair of functional program and validation program is defined as the percent of assembly codes that can be included in the frames and verified in the validation program .

In the case of spatial filter, there are 112 lines(L1) of assembly codes in total, and there are 78 lines(L1-L2) of assembly codes included in the frames. As a result, 78 out of 112 lines of assembly code can be validated and the level of variance is 70 percents.

Level of Variance = $\frac{L1-L2}{L1}$

Chapter 6

Experimental results and analysis

6.1 Experiment setup

1. Benchmark programs

We have benchmark programs from four types of applications. The first is a spatial image filter used in digital image processing. The second is the traditional quick sort program used in sorting. The third is the SHA encryption program used in security applications. The last is a DJPEG program used in the image compression applications.

2. Experiment environment

- Program Compiler : lcc 4.1

- Operation System : Ubuntu 12.1 Linux
- SystemC Simulator: gcc 4.7.2
- SystemC library : systemc 2.2.0

3. Steps of experiment

NOC platform preparation:

- Get the NOC SystemC model and systemc 2.2.0 library.
- Configure the NOC platform to be post-silicon validation mode.
- Compile the source files into executable file using gcc.

Benchmark Simulation:

- Create a validation program based on the functional program.
- Compile both functional program and validation program into binary files using lcc.
- Load the data into RAM and load the programs into ROM.
- Simulate the NOC platform with both RAM and ROM using gcc.
- Analyze the report from validating cores.

6.2 Experiment results

Each assembly code size is collected in the assembly file generated from a full-functional program. Benchmark spafilter is deployed in a way that each core processes part of the spatial filtering task. An image is split into equal-sized pieces and each core is running spafilter on its own piece of input image pixels. Benchmark quicksort is sorting a list of randomly generated numbers. Each core is running the sorting program on a set of pre-loaded random numbers. There is no communication between functional cores in benchmarks spafilter and quicksort, and each core is playing a similar role. Benchmark SHA has a father process and a child process that are mapped to two adjacent functional cores, and the structure of two coupled functional cores is extended to the whole NOC platform. Benchmark DJPEG is split into three streaming pieces and deployed in a four-core structure, and the structure of four functional cores is extended to the whole NOC platform.

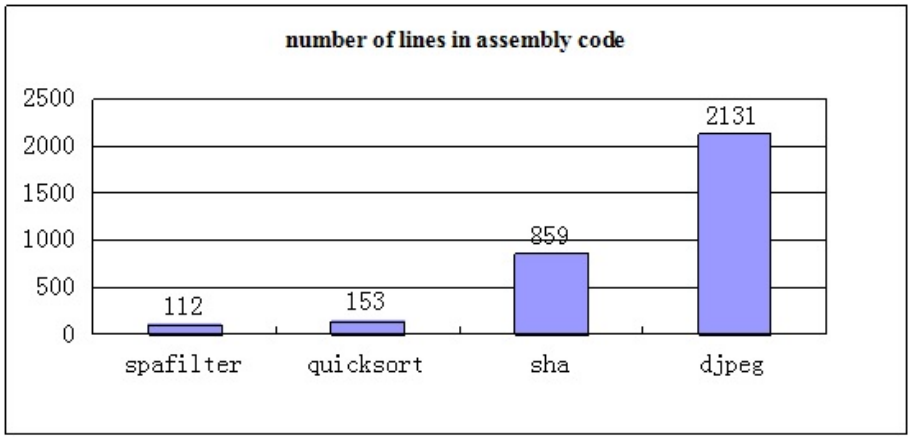

Figure 6.1: Assembly code size for benchmark programs

Benchmark spafilter is a spatial filter program. When comparing the functional program and its validating program, logical operations are varied but the loop is kept the same. Benchmark quicksort is a sorting program. Method of attribute verifying is taken and full level of variance is achieved. Benchmark SHA is an encryption program. Loops and the program structure make it difficult to create enough variance. Benchmark DJPEG is an integer JPEG decoding program. It achieves better level of variance due to large assembly code size and more logic operations. The level of variance for each benchmark program is shown in Fig 6.2.

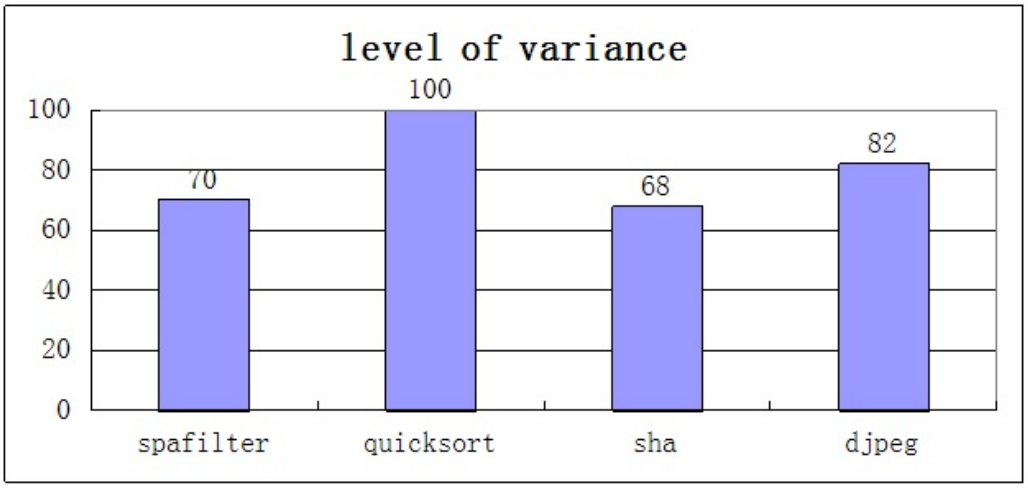

Figure 6.2: Level of variance for benchmark programs

Each benchmark program runs through the NOC platform simulation with the SystemC library, and the computer simulation time is given in Table 6.1.

Table 6.1: Simulation time for benchmark programs

	spafilter	quicksort	SHA	DJPEG
Simulation time	48m	2h 52m	7h 12m	10h 21m

6.3 Design bug analysis

Here is an insertion on experimental results that are demonstrating usage of variance validation.

6.3.1 Configuration

Benchmark program spafilter is used in this experiment. Design bugs are injected into the NOC platform for both functional cores and validating cores.

Basic Configuration : Shift logic in ALU block is configured to be with design bugs. In Experiment BS-1, logic unit for shift-left-by-2-bit is modified to perform the operation of shift-left-by-1-bit. This shift-left-by-2-bit operation exists only in the functional program. In the validation program, the shift-left-by-2-bit operation is replaced by 2 shift-left-by-1-bit operations and it is gone. In Experiment BS-2, logic unit for shift-left-by-1-bit is modified to perform nothing but keeping its original value. This shift-left-by-1-bit operation is used in both functional program and validation program. In the functional program, it is used for multiplication by 2. While in the validation program, it is used to take the place of shift-left-by-2-bit.

Advanced Configuration: Shift logic in ALU block is configured to perform with conditional design bugs. That is to say, the output is incorrect

Figure 6.3: Basic configuration: BS-1 (L) and BS-2 (R)

only when the condition is met. Otherwise, the block functions correctly. For example, shift logic outputs incorrect result when both bit 3 and bit 5 of input are high.

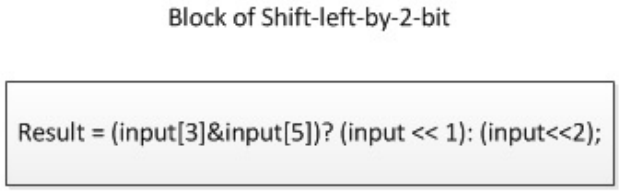

Figure 6.4: Advanced configuration

6.3.2 Experimental Results

For basic configuration, in both experiment BS-1 and experiment BS-2, all validation results fail. In experiment BS-1, the functional program creates incorrect results and they fails the validation program. In experiment BS-2, both the functional program and the validation program are creating incorrect results. Since the design bug activates different parts of assembly codes in the frames, the erroneous functional results do not match the erroneous

validation results. As a result, it fails the validation program.

For advanced configuration, depending on the input pixel data, some results are correct while some other results are incorrect.

6.3.3 Conclusion

For basic configuration, it shows that the method of variance validation works when there are design bugs. Even if design bugs exist in both functional cores and validating cores, validation process fails because functional program and validation program do not create the same erroneous results.

For advanced configuration, it is similar to the situation of noise in a chip. In such a situation, erroneous behavior of chip can only be detected by certain input stimulus. This kind of design bugs cannot be caught in pre-silicon verification because of lack of noise model. What's more difficult, this kind of design bugs can only be detected by certain input patterns. Hence the bugs are easy to escape the post-silicon validation without enough test patterns.

Chapter 7

Conclusions and future research

7.1 Conclusions

By dividing cores into functional cores and validating cores and pairing them in a NOC platform, it is able to proceed with the ideal comparison and complete on-chip post-silicon validation with little change to the NOC platform. It is also possible to automate the process of post-silicon validation by properly programming the validating cores.

Compared to software-based approaches, the method used in this thesis greatly reduces workload for dump data transmission. In traditional software-based approaches, the communication channel is shared between functional data transmission and dump data transmission, and this imposes a great demand on bandwidth of the communication channel. The method used in this thesis work separates the communication channel for functional

purpose and validation purpose. Normal communication is not impacted and dump data transmission is done by a FIFO between adjacent cores. They do not interfere with each other due to different channels.

Compared to hardware-based approaches, the method used in this thesis greatly reduces the cost for post-silicon validation. Expensive logic analyzer is often used in hardware-based approaches. Also, on-chip cache for dump data or trace buffer are expensive, and the volume or size of the data storage is limited. The dump data for the whole post-silicon validation process from starting to breakpoint or crash are stored in the storage before off-chip analysis for debugging. The method used in this thesis work does not require off-chip equipments. Each FIFO working as a data storage is also included in the chip design, but the required FIFO size is potentially smaller because on-chip validation is introduced and the dump data are validated at the same time when data are dumped into the FIFO.

By creating variance between functional programs and validation programs, the method is able to detect design bugs and is robust in post-silicon validation.

Compared to approaches that run the same copies of a program in different cores, the method used in this thesis can detect hidden hardware problems. A bug in design specification can escape from verification due to insufficient testbenches. The tape-out cores with the bug will still generate the same response when executing the same copies of program even though the hardware does not function correctly. Variance created in this thesis work

helps catch this kind of bugs and the proposed method is much more robust in post-silicon validation.

Compared to approaches that reverse the functional program to reproduce the initial state, the method does not require the program to be fully reversible. This is preferable because not all programs can be easily reversed. When it comes to a decision whether to reverse a fragment of the functional program, a better compromise can be made based on how difficult it is to reverse the fragment of the program.

With little effort, the idea can be used in a reliable system design. Given the fact of variance between cores, it is more reliable for the system compared to traditional methods.

7.2 Future research work

1. Grouping more cores together with more validating cores for one functional core.

   In this thesis work, one functional core and one validating core are paired and there is only one validating core to complete the task of validation. Thus it is a requirement that the validating program must be more efficient(i.e. runs faster) than the functional program. In fact, it is possible to group more cores together and there can be more than one validating core for the functional core. In such a situation, there is no need to request a more efficient validating program and validating

can be more effective.

For example, four cores can be grouped together and divided into one functional core and there validating cores. A 2 by 2 matrix of cores are grouped and can be extended in the NOC platform.

2. Increasing the diversity of instructions

   Variances are created by different approaches to the same solution. A variety of instructions helps improve the level of variance.

   A simplified MIPS instruction set is used in this thesis work and it provides a limited alternative choices for instructions. But in modern processors, there are multiple choices of instructions to complete the same task and thus variance can be created based on variety of instructions.

3. Reusing FIFO as shared memory

   FIFO takes the role of transmitting dump data from the functional core to the validating core in each pair, and it is hardware overhead for the NOC platform.

   There is an alternative choice for FIFO implementation if there is a shared memory between adjacent cores. The shared memory can be accessed by both cores and plays the same role of data transmission as a FIFO. As a result, the shared memory can be reused and taken as a FIFO so that there is little hardware overhead for the NOC platform.

4. Automating the process of variance creation

   Variance is created manually in this thesis work. It is preferable to generate the validation programs with variance automatically. By doing so, it prevents errors introduced by human effort and reduces the effort and time to perform post-silicon validation.

Bibliography

[1] P. Das and S. K. Gupta, "Efficient post silicon validation via segmentation of the process variation envelope: Global vs. local variations," in *Proc. 50th Design Autom. Conf. (DAC)*, Austin, TX, Jun. 2013.

[2] T. Bjerregaard and S. Mahadevan, "A survey of research and practices of network-on-chip," *ACM Computing Surveys (CSUR)*, vol. 38, no. 1, p. 1, 2006.

[3] P. Gillard, C. Li *et al.* (2011) Network-on-chip (noc) topologies and performance: A review. [Online]. Available: http://necec.engr.mun.ca/ocs2011/

[4] S.-B. Park and S. Mitra, "Ifra: instruction footprint recording and analysis for post-silicon bug localization in processors," in *Proc. 45th Design Autom. Conf. (DAC)*, Anaheim, CA, Jun. 2008, pp. 373–378.

[5] A. DeOrio, I. Wagner, and V. Bertacco, "Dacota: Post-silicon validation of the memory subsystem in multi-core designs," in *Proc. Int. Symp. on High Performance Computer Architecture(HPCA)*, Raleigh, NC, 2009, pp. 405–416.

[6] H. F. Ko, A. B. Kinsman, and N. Nicolici, "Distributed embedded logic analysis for post-silicon validation of socs," in *Proc. IEEE International Test Conference(ITC)*, Santa Clara, CA, Oct. 2008, pp. 1–10.

[7] W. Li, A. Forin, and S. A. Seshia, "Scalable specification mining for verification and diagnosis," in *Proc. 47th Design Autom. Conf. (DAC)*, Anaheim, CA, Jun. 2010, pp. 755–760.

[8] M. Gao and K.-T. Cheng, "A case study of time-multiplexed assertion checking for post-silicon debugging," in *Proc. IEEE International High*

Level Design Validation and Test Workshop (HLDVT), Anaheim, FL, Jun. 2010, pp. 90–96.

[9] T. Hong, Y. Li, S.-B. Park, D. Mui, D. Lin, Z. A. Kaleq, N. Hakim, H. Naeimi, D. S. Gardner, and S. Mitra, "Qed: Quick error detection tests for effective post-silicon validation," in *Proc. IEEE International Test Conference (ITC)*, Austin, TX, Nov. 2010, pp. 1–10.

[10] N. Foutris, D. Gizopoulos, M. Psarakis, X. Vera, and A. Gonzalez, "Accelerating microprocessor silicon validation by exposing isa diversity," in *Proc. 44th IEEE/ACM Int. Symp. on Microarchitecture*, New York, NY, 2011, pp. 386–397.

[11] I. Wagner and V. Bertacco, "Reversi: Post-silicon validation system for modern microprocessors," in *Proc. IEEE International Conference on Computer Design(ICCD)*, Lake Tahoe, CA, Oct. 2008, pp. 307–314.

[12] I. M. de la Rosa, "Hardware communication services for synchronous data flow in the mini-noc," Master's thesis, 2006.

[13] S. Mitra, S. A. Seshia, and N. Nicolici, "Post-silicon validation opportunities, challenges and recent advances," in *Proc. 47th Design Autom. Conf. (DAC)*, Anaheim, CA, Jun. 2010, pp. 12–17.

[14] J. Goodenough and R. Aitken, "Post-silicon is too late avoiding the $50 million paperweight starts with validated designs," in *Proc. 47th Design Autom. Conf. (DAC)*, Anaheim, CA, Jun. 2010, pp. 8–11.

[15] K. Balston, M. Karimibiuki, A. Hu, A. Ivanov, and S. Wilton, "Post-silicon code coverage for multiprocessor system-on-chip designs," *IEEE Trans. Comput.*, vol. 62, no. 2, pp. 242–246, Feb. 2012.

[16] (2010) The economics of verification. [Online]. Available: http://www.testandverification.com/files/Verification_Awareness.ppt

[17] J. Keshava, N. Hakim, and C. Prudvi, "Post-silicon validation challenges: how eda and academia can help," in *Proc. 47th Design Autom. Conf. (DAC)*, Anaheim, CA, 2010, pp. 3–7.

[18] S. R. Vangal, J. Howard, G. Ruhl, S. Dighe, H. Wilson, J. Tschanz, D. Finan, A. Singh, T. Jacob, S. Jain *et al.*, "An 80-tile sub-100-w

teraflops processor in 65-nm cmos," *IEEE J. Solid-State Circuits*, vol. 43, no. 1, pp. 29–41, Jan. 2008.

[19] (2013) SystemC wiki. [Online]. Available: http://en.wikipedia.org/wiki/SystemC

[20] (2013) SystemC Homepage. [Online]. Available: http://www.accellera.org/

[21] L. Benini, D. Bertozzi, D. Bruni, N. Drago, F. Fummi, and M. Poncino, "Systemc cosimulation and emulation of multiprocessor soc designs," *IEEE Computer*, vol. 36, no. 4, pp. 53–59, Apr. 2003.

[22] (2006) Mini-NOC Homepage. [Online]. Available: http://www.es.ele.tue.nl/mininoc/

[23] L. Benini and G. De Micheli, "Networks on chip: a new paradigm for systems on chip design," in *Proc. Design, Automation & Test in Europe Conference & Exhibition(DATE)*, Paris, France, Mar. 2002, pp. 418–419.

[24] H. G. Lee, N. Chang, U. Y. Ogras, and R. Marculescu, "On-chip communication architecture exploration: A quantitative evaluation of point-to-point, bus, and network-on-chip approaches," *ACM Transactions on Design Automation of Electronic Systems (TODAES)*, vol. 12, no. 3, p. 23, 2007.

[25] A. Ivanov and G. De Micheli, "Guest editors' introduction: The network-on-chip paradigm in practice and research," *IEEE Des. Test. Comput.*, vol. 22, no. 5, pp. 399–403, Sep./Oct. 2005.

[26] E. Salminen, A. Kulmala, and T. D. Hamalainen. (2008, Mar.) Survey of network-on-chip proposals. [Online]. Available: http://ocpip.biz/uploads/documents/OCP-IP-Survey-of-NoC-Proposals-White-Paper-April-2008.pdf

[27] S. Kumar, A. Jantsch, J.-P. Soininen, M. Forsell, M. Millberg, J. Oberg, K. Tiensyrja, and A. Hemani, "A network on chip architecture and design methodology," in *Proc. IEEE Computer Society Annual Symposium on VLSI*, Pittsburgh, PA, Apr. 2002, pp. 105–112.

[28] A. Agarwal, C. Iskander, and R. Shankar, "Survey of network on chip (noc) architectures & contributions," *Journal of engineering, Computing and Architecture*, vol. 3, no. 1, pp. 21–27, 2009.

[29] V. Rantala, T. Lehtonen, and J. Plosila, "Network on chip routing algorithms," Turku Center for Computer Science(TUCS), Turku, Finland, Tech. Rep. 779, Aug. 2006.

[30] M. Dehyadgari, M. Nickray, A. Afzali-Kusha, and Z. Navabi, "Evaluation of pseudo adaptive xy routing using an object oriented model for noc," in *Proc. IEEE Int. Conf. Microelectron.*, Dec. 2005, pp. 204–208.

[31] C. J. Glass and L. M. Ni, "The turn model for adaptive routing," in *Proc. ACM SIGARCH Computer Architecture News*, vol. 20, no. 2, 1992, pp. 278–287.

[32] C. E. Perkins and E. M. Royer, "Ad-hoc on-demand distance vector routing," in *Proc. IEEE Workshop on Mobile Computing Systems and Applications(WMCSA)*, New Orleans, LA, Feb. 1999, pp. 90–100.

[33] M. Gerla and L. Kleinrock, "Flow control: A comparative survey," *IEEE Trans. Commun.*, vol. 28, no. 4, pp. 553–574, Apr. 1980.

[34] D. Bertozzi and L. Benini, "Xpipes: a network-on-chip architecture for gigascale systems-on-chip," *IEEE Circuits and Systems Magazine*, vol. 4, no. 2, pp. 18–31, Sep. 2004.

[35] M. Abramovici, P. Bradley, K. Dwarakanath, P. Levin, G. Memmi, and D. Miller, "A reconfigurable design-for-debug infrastructure for socs," in *Proc. 43th Design Autom. Conf. (DAC)*, San Francisco, CA, Jun. 2006, pp. 7–12.

[36] P. Patra, "On the cusp of a validation wall," *IEEE Des. Test. Comput.*, vol. 24, no. 2, pp. 193–196, Mar./Apr. 2007.

[37] S. Yerramilli, "Addressing post-silicon validation challenge: Leverage validation & test synergy (invited address)," in *Proc. Intl. Test Conf*, 2006.

[38] G. Gopalakrishnan and C. Chou, "The post-silicon verification problem: Designing limited observability checkers for shared memory processors," in *Proc. 5th International Workshop on Designing Correct Circuits (DCC)*, Barcelona, Spain, 2004.

[39] A. Nahir, A. Ziv, R. Galivanche, A. Hu, M. Abramovici, A. Camilleri, B. Bentley, H. Foster, V. Bertacco, and S. Kapoor, "Bridging pre-silicon verification and post-silicon validation," in *Proc. 47th Design Autom. Conf. (DAC)*, Anaheim, CA, Jun. 2010, pp. 94–95.

[40] A. Adir, A. Nahir, G. Shurek, A. Ziv, C. Meissner, and J. Schumann, "Leveraging pre-silicon verification resources for the post-silicon validation of the ibm power7 processor," in *Proc. 48th Design Autom. Conf. (DAC)*, New York, NY, Jun. 2011, pp. 569–574.

[41] S. Ray and W. Hunt, "Connecting pre-silicon and post-silicon verification," in *Proc. Formal Methods in Computer-Aided Design(FMCAD)*, Austin, TX, Nov. 2009, pp. 160–163.

[42] A. Adir, S. Copty, S. Landa, A. Nahir, G. Shurek, A. Ziv, C. Meissner, and J. Schumann, "A unified methodology for pre-silicon verification and post-silicon validation," in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Grenoble, France, Mar. 2011, pp. 1–6.

[43] K. P. Parker, *The boundary-scan handbook*. New York, NY: Kluwer Academic Pub, 2003.

[44] S. Tang and Q. Xu, "A multi-core debug platform for noc-based systems," in *Proc. Design, Automation & Test in Europe Conference & Exhibition(DATE)*, Apr. 2007, pp. 1–6.

[45] O. Dahmoune and R. de B Johnston, "Applying model-checking to post-silicon-verification: Bridging the specification-realisation gap," in *Proc. International Conference on Reconfigurable Computing and FPGAs (ReConFig)*, Quintana Roo, Dec. 2010, pp. 73–78.

[46] F. M. De Paula, M. Gort, A. J. Hu, S. J. Wilton, and J. Yang, "Backspace: formal analysis for post-silicon debug," in *Proc. International Conference on Formal Methods in Computer-Aided Design*, Portland, OR, 2008, pp. 1–10.

[47] (2006) mMIPS Home. [Online]. Available: http://www.es.ele.tue.nl/mininoc/doc/minimips.htm

[48] (2013) mips home. [Online]. Available: http://www.mips.com

[49] W. J. Dally and C. L. Seitz, "Deadlock-free message routing in multiprocessor interconnection networks," *IEEE Computer*, vol. 100, no. 5, pp. 547–553, May 1987.

[50] F. Verbeek and J. Schmaltz, "Formal specification of networks-on-chips: deadlock and evacuation," in *Proc. Design, Automation & Test in Europe Conference & Exhibition (DATE)*, Dresden, Germany, Mar. 2010, pp. 1701–1706.

[51] W. J. Dally and B. Towles, "Route packets, not wires: On-chip interconnection networks," in *Proc. 38th Design Autom. Conf. (DAC)*, Las Vegas, NV, Jun. 2001, pp. 684–689.

[52] (2013) Lcc doc. [Online]. Available: http://www.es.ele.tue.nl/mininoc/doc/lcc.htm

[53] C. W. Fraser and D. R. Hanson, *A retargetable C compiler: design and implementation.* Punta Gorda, FL: Addison-Wesley, 1995.