TALLINN UNIVERSITY OF TECHNOLOGY

Faculty of Information Technology

IEE70LT

Apneet Kaur 132094IVEM

Online Fault Detection Methodology for Control and Data Path of IP Cores: Case Study on Bonfire Network-On-Chip

Master's Thesis

Supervisor : Dr. Paul Annus
(PhD, Sr. Research Scientist)
Co-Supervisor : Prof. Jaan Raik
(PhD, Professor)
Er. Siavoosh Payandeh Azad
(MSc, Early Stage Researcher)
Er. Behrad Niazmand
(MSc, Early Stage Researcher)

Tallinn 2016

TALLINNA TEHNIKAÜLIKOOL

Infotehnoloogia Teaduskond

IEE70LT

Apneet Kaur 132094IVEM

Metodoloogia tuumade juht- ja andmeosa rikete avastamiseks töörezhiimis: juhtumiuuring Bonfire kiipvõrgul

Magistritöö

Juhendaja : Dr. Paul Annus
(PhD, Vanemteadur)
Kaasjuhendaja : Prof. Jaan Raik
(PhD, Professor)
Er. Siavoosh Payandeh Azad
(MSc, Nooremteadur)
Er. Behrad Niazmand
(MSc, Nooremteadur)

Tallinn 2016

Author's Declaration

I hereby certify that I am the sole author of this thesis. All the used materials, references to the literature and the work of others have been referred to. This thesis has not been presented for examination anywhere else.

Name:Apneet Kaur

Date: June 7, 2016

Acknowledgement

I would first like to sincerely thank my advisor and Co-Supervisor, Prof. Jaan Raik for his guidance, patience and support. He gave me much freedom to explore interesting problems of NoC, while providing invaluable insights and comments on my thoughts. My research has been benefited greatly from his knowledge. His high standards played a key role to build up my confidence in both the normal and research life, which is the most precious achievement I am gaining during the study.
I would like to thank my Supervisor, Dr. Paul Annus for availing me with the great opportunity of carrying out my research thesis work at department of Computer Engineering at Tallinn University of Technology.
Special thanks to my third and fourth Co-supervisors Siavoosh Payandeh Azad (Jr. Researcher, *TTU*) and Behrad Niazmand (Jr. Researcher, *TTU*) for their patience and valuable guidance. Thanks for considering me capable for the position of IEEE-WIE Chairperson. They have lead the whole team of 12, working on the Bastion and Immortal Project through weekly meetings and intensive brainstormings. Thanks for continuous feedback and correcting me at every step of thesis that motivate me to aim for perfection. A special thanks to my colleagues in Computer Engineering department Vineeth Govind (Sr. Researcher, *TTU*, Estonia) who taught me initially about the designing of Network Interface. Thanks to Karl (Jr. Researcher, *TTU*), Hardi (Masters, *TTU*), Tsotne (Masters, *TTU*), Martin (Masters, *TTU*) and Nevin (Masters, *TTU*) for the support and enjoyable working environment.Thanks to Erasmus-Mundus Heritage (2013-2015) and Estonian Science Foundation Grant 9429(2015-2016) for funding my studies.
A Special thanks to my Family, Guneet Kaur *(Mother, creative home maker)*, Baljinder Singh*(Father,Retd.Bank Manager)* for their support through phone and skype calls and to my siblings Dapinder Singh *(Brother, Chief Engg. MMS)* and Tanzeer Kaur *(Sister, Assistant Prof. PU)* for they are my great role models and motivating to be like them.
Finally, I would like to thank my fiancée Deepak Pal, *(Jr. Researcher/Software Developer Elvior)* for the support, encouragement and patience provided during my research. Thanks for cheering me up in my hard times.

Abstract

The continuous scaling in technology has caused the components of a network-on-chip to be more vulnerable to faults; therefore, there is a need for methods to maintain circuit performance. This dissertation propose a methodology to partition design into control and data parts, each to be protected by different circuitry. An increasing number of faults appear during the run-time of the system caused by temperature-time dependent degradation and environmental effects like radiation or electromagnetic noise. Thus, online fault monitors are required for detecting faults during circuit's life-time. The concurrent checkers monitor the control part and error control code checks the data-path of the NoC. A new method and tool framework is developed for checker qualification and minimization, which aim at sufficient fault coverage and checker efficiency index with minimum area overhead. Concurrent checkers and error control codes are implemented for a case study on a TUT design project Bonfire NoC evaluating different error control mechanisms and strategies.

Annotatsioon

Kiibitehnoloogiate pidev miniaturiseerimine on muutnud kiipvõrgud rikete suhtes tundlikumaks. Seega on vajadus uute meetodite järgi, mis säilitaksid kiipide jõudluse. Käesolev dissertatsioon pakub välja metodoloogia skeemi tükeldamiseks juht- ja andmeosaks, milledest kumbki oleks kaitstud erineva arhitektuurse lahendusega. Üha kasvav hulk rikkeid toimub süsteemi eluajal põhjustatuna temperatuurist tulenevast degradeerumisest ning keskkonnast tulenevast kiirgusest ja elektromagneetilistest häiretest. Seega on tarvis rikkemonitore ja veakodeerimisel põhinevaid lahendusi skeemi eluajal tekkivate vigade avastamiseks. Monitorid jälgivad kiipvõrgu juhtosa ning veakodeerimisel põhinevad skeemid jälgivad selle andmeosa. Dissertatsioonis töötati välja uus meetod ja programne keskkond rikkemonitoride kvantitatiivseks analüüsiks ning nende hulga minimiseerimiseks, mille eesmärgiks on piisava rikete katte saavutamine minimaalse lisakuluga. Rikkemonitore ja veakodeerimisel põhinevaid mehanisme rakendati TTÜ Bonfire kiipvõrgu projektis toimunud juhtumiuuringus, et hinnata erinevaid tõrkekindlaid strateegiaid.

Abbreviations

NOC	-Network on Chip
IC	-Integrated Chips
SoC	-Semiconductor on Chip
MPSoc	-Multiple processors on system on chip
ECC	-Error Control Code
SECDED	-Single Error Correction Double Error Detection
NI	-Network Interface
SP	-Single Parity
CRC	-Cyclic Redundancy Code
DAP	-Duplicate Add-Parity
BCH	-Bose-Chaudhuri-Hocquenghem Codes
RS	-Reed-Solomon
DMR	-Double-Modular-Redundancy
TMR	-Triple-Modular-Redundancy
SEU	-Single Event Upset
BICST	-Built-In Concurrent Self-Test
ROWR	-Reduced Observation Width Replication
LBDR	-Logic Based Distributed Routing
IIR	-Inherent Information Redundancy
ForEVeR	-Formally Enhanced Verication at Runtime for NoCs
VCs	-Virtual Channels
CTS	-Clear To Send
RTS	-Ready To Send
EMI	-Electro-Magnetic-interference
SER	-Soft Error Rate
FCC	-Forward error correction
CEI	-Checkers Efficiency Index
FC	-Fault Coverage
FPR	-False Positives Ratio
TT	-Turbo Tester
SSBDD	-Structurally Synthesized Binary Decision Diagrams
BDD	- Binary Decision Diagrams
FFR	- fan-out free regions
SAF	-Stuck-At-Fault
LFSR	-Linear Feedback Shift Registers
B-M	-Berlekamp-Massey
FCS	-Frame Check Sequence
FF	-Flip-Flop
FSM	-Finite State Machine
RR	-Round-Robin
DRTS	-Detect Ready To Send
Read en	-Read Enable
empty out	- Empty signal

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Motivation

The continuous scaling in semiconductor technology enables the amount of transistors on a single chip to be doubled in every 18 months as theorized in Moore's Law. Today IC's have a high capacity of transistors and capable of connecting various components onto a single chip forming a system on chip. Conventionally, SoC had a single processor connected to peripherals using bus or crossbar architecture. Now, SoC also provides possibility to connect multiple processors system on chip MPSoC.

As semiconductor technology scales down further, the SoC design problem focus shifts from data processing towards effective communication which is now a major contributor for SoC performance and implementation cost. Traditionally, the interconnection between multiple cores in SoC followed conventional bus or crossbar structures. However, with the increasing number of processing cores, the conventional communication media become the system's bottleneck due to their low scalability. Though, to enable concurrent data transmission, busses can be segmented [1] at the expense of higher arbitration complexity or replaced with crossbars at the expense of a significantly larger number of wires. However, segmented buses and crossbars are not fully scalable and thus can provide only an intermediate solution. The Network-on-Chip NoC model [2] has been proposed and is widely accepted as a scalable and reliable communication infrastructure replacement of buses and crossbars[5]. NoC is not just more scalable, can also operate at much higher clock

frequency and has lower power consumption [8]. However, further scaling of integrated circuits makes the NoC design more vulnerable to different kinds of faults, thus, making design's reliability a real challenge. For correct behavior of the system, data integrity is important. Different error control codes are implemented and analysed in terms of area in this work. Also, an increasing number of faults appear during the run-time of the system caused by temperature-time dependent degradation and environmental effects like radiation or electromagnetic noise. Thus, concurrent fault monitors for detecting faults during circuit's lifetime are needed. These checkers would report errors within routers and would allow reconfiguration of the routing infrastructure, that operate during the lifetime of system and parallel with its normal operations. The thesis proposes a new methodology of partitioning systems into control and datapath with the former to be covered by online checkers and the latter with error correcting codes. In addition methods for automated checker qualification and minimization are presented. In this chapter a brief overview of related work about the error control codes techniques and on-line testing is detailed.

1.2 Related work

The related work is divided into two parts one for fault detection and correction in data and other for designing concurrent online-checkers for faults in control logic.

1.2.1 Data Error correction and Detection

Error detection and correction techniques help to maintain the data integrity of the system. There are two well-known schemes namely, end-to-end encoding that work at network level and switch-to-switch encoding at link level. A hybrid of both schemes can also be used along with retransmission [10]. The error control schemes are not efficient in case of intermittent errors. For instance, for intermittent errors a system get stuck at retransmissions of a single piece of data due to continuous error. The error detection and correction scheme fails in case of permanent error, and it takes a large area overhead to develop strong error control schemes for permanent and intermittent errors.

This literature survey provides a large amount of useful data in order to select a suitable error control scheme for a given NoC design. The end-to-end encoding strategy is applied in the network interface on the entire packet at the source and destination points. The power consumption of end-to-end error control is less than that of hop-to-hop error control at low error rate, route length and small number of retransmissions [21]. More errors can appear in data because of no error recovery at hops in the router. Retransmission causes increase in latency, if it occurs frequently. In end-to-end, ECC require strong error control scheme to meet the same reliability level of hop-to-hop ECC, possibly requiring a larger codec in NI in case of high noise conditions. A simple end-to-end ECC technique combined with a timeout retransmission mechanism has been proposed to reduce the latency caused by packet corruption or loss [23].

In [25] author proposed a method that makes use of the extended Hamming code known as a Single Error Correction Double Error Detection (SECDED) code in the NI. The method is able to improve energy efficiency for a variety of design parameters with area overhead as the main limitation. In [26], the author compares different error detection and correction schemes to find the most efficient error control technique from energy viewpoint, that includes Hamming codes, extended Hamming codes, parity check codes, and CRC codes.

The number of simultaneous permanent, transient, intermittent or burst errors are expected to increase in future with the continuous scaling of the IC's. A multiple-error correction code is constructed by combining Hamming codes with Duplicate-Add-Parity (DAP) codes [29]. In this method, the outputs of a hamming encoder are duplicated and an extra parity bit generated from the original hamming code is added. This method requires a large number of additional interconnect wires that greatly increases the energy consumption and link area overhead. Bose-Chaudhuri-Hocquenghem Codes (BCH) codes are considered as an important class of linear block codes for multiple error correction. The BCH encoder follows approach similar to cyclic codes. Reed-Solomon (RS) codes are a subclass of BCH codes good at correcting multiple symbol errors. The BCH and RS codes are implemented in NoC links, that can correct multiple errors with limitation of large codec area, power consumption and decoding time[27] .

1.2.2 Concurrent Online Testing

The faults in logic circuits and its online detection is a thoroughly studied research area. Techniques like Double-Modular-Redundancy *(DMR)* or Triple-Modular-Redundancy *(TMR)* are most simple way to implement online detection with redundancy and are costly in terms of area overhead and increase in power consumption. To minimize this overhead, selective *TMR* can be used to identify Single Event Upset *(SEU)* in the circuits [35]. The Built-In Concurrent Self-Test (BICST) technique[36] and Reduced Observation Width Replication (ROWR) technique [37] achieve high fault coverage at low area overhead with limited pre-computed test vectors. Though, these approaches are susceptible to miss faults during normal operation of the circuit.

Checkers can be extracted from the functional logic of the design [38] , but it has limitation of low fault coverage and large area overhead, higher than DMR scheme. On the other hand, an approach based only on functional assertions or reuse of verification assertions suffers from low fault coverage of structural faults, and the obtained functional coverage is difficult to relate with the structural one [39].

A number of previous works propose error control methods for reducing faults in a router of a NoC. Fault tolerance was added in NoC topologies for detecting errors in routing logic using Logic Based Distributed Routing mechanism [42] and its extension was used for high-radix topologies, i.e LBDRhr [43]. This method utilizes the Inherent Information Redundancy (IIR) to reduce the area overhead, but it did not had full fault coverage. Authors of [44] have presented a method for online error detection and diagnosis of NoC switches. This method addresses routing faults caused by packet corruption in such a way that it forwarded to a wrong destination port. A high-level fault model has been proposed that has fault coverage only at the functional level and lacks generality with area overhead. In [45], ForEVeR (Formally Enhanced Verification at Runtime for NoCs) method proposes a network-level detection and recovery solution. It provides a solution to detect functional errors in NoCs.This approach detects only 30 percent of the faults in first clock cycle and has large latency.

The work in [46] ensures network functional correctness by implementing checkers based on using set of 32 invariance. The unexpected behavior is captured by any one of the 32 assertion checkers. However, it is not clear how injection of faults, fault simulation and evaluation of coverage are accomplished by this method. Moreover the approach is not automated and lacks the completeness and minimization aspects. An automated framework is proposed which provides a minimized list of online checkers for checking the control part of on-chip routers. This method is closest to the idea of concurrent online detection which is considered in this dissertation. It makes use of 5 checkers from set of 31 and with covering only the control part of NoC with large area overhead. In this dissertation LBDR considered for designing checkers are more generic i.e considering all 4 ports (E, W, S, N), whereas in [40] it considers East - LBDR. The Checkers design for this dissertation are inspired by the work of [40] and same method and filtering approach is used for the input stimuli (environment).

1.3 Contribution

This dissertation propose a methodology to partition design into control and data parts, each part to be protected by checkers and error control code respectively. As background information fault types, its sources, classification and a survey of fault tolerance methods are studied and presented in this dissertation. For reliable on-chip communication different types of error control codes are studied out of which hamming, cyclic redundancy codes and single parity are implemented and compared in terms of area. The thesis proposes a new methodology of partitioning systems into control and data-path with the former to be covered by online checkers and the latter with error correcting codes. In addition, new method and tool framework for checker qualification and minimization is developed.

The case study Bonfire NoC includes implementation of the ECC and Checkers which is shown in Chapter 6 and Chapter 7. This case study is a part of a "EU's Horizon 2020 Research and Innovation Action IMMORTAL" and "EU's FP7 Collaborative Research Project BASTION". The design decisions made in the case study will be used in the real manufacturing of the fault tolerant IC by the team

members of projects from TTÜ .

1.4 Outline of Thesis

The introduction chapter has given an overview of the thesis. The rest of thesis is organized as follows: chapter 2 provides the basis for better understanding of Network on Chip (NoC) describing its basic building blocks, network topologies, routing algorithms and switching techniques. Chapter 3 classifies the types of faults, most common fault sources are discussed. It gives basis for understanding of fault tolerance methods divided under two categories namely static and dynamic. In chapter 4 the basics of concurrent checkers and the new metrics used in fault evaluation are explained. Chapter 5 explains about the ECC techniques concept in theory. In chapter 6 selected ECC are implemented and its area is compared. Chapter 7 discuss about the checkers devised to deal with the control faults and its experimental results are presented with discussion. Chapter 8 Concludes about the results and explains the possible future work.

Chapter 2

Network On Chip Architechture

The main components of NoC are routers, network interface and physical links that provide communication between multiple processors on chip. The figure 2.1 shows A 4x4 NoC example in 2D mesh topology .

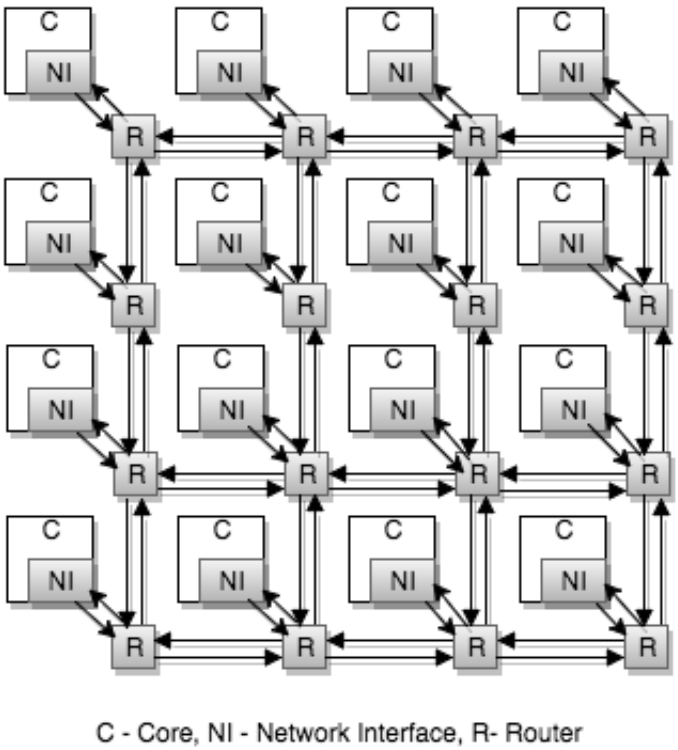

Figure 2.1: A 4x4 NoC example in 2D mesh topology.

- **Network interfaces** act as an interface between the cores and the routers. It accepts data in form of packets from router and de-packetize it to send it into core. For this purpose it has a packetizer and a de-packetizer and a buffer to store data.

- **Routers** are considered to be the backbone of the NoC. It has five output ports out of which four ports are connected to the four neighboring routers and fifth port is connected to network interface. The routing algorithm, flow control and switching techniques play a very important role in routers. A NoC router contains of a number of input and output ports, a switching matrix connecting the input ports to the output ports, and a local port to access the IP resource connected to the same router.

- **Links** connect adjacent routers. The interconnect between two adjacent routers are usually composed of two unidirectional links which in charge of the outgoing or incoming traffic, respectively. It is possible to replace. The cores in NoC can be any processing or memory units.

The factors that influence the performance of a NoC are *topology* that determine the arrangement of channels and routers, *routing technique* defines the way a message is transmitted from the source to the destination, *flow control* that tells how are network resources allocated when packets travel through the network, *router architecture* with various components such as buffers and switches, and *traffic pattern.*

2.0.1 Topologies

Topology defines the way routers are connected to each other to form a network. The routers can be connected in a direct or an indirect form. NoC with direct topologies are where every node in the network is both a terminal and a switch for example *mesh, torus, hypercube, octagon* topologies. The Indirect topology are where nodes are either switches or terminal for example *fat-tree and three-stage butterfly.* The mixed topology are application specific. All the topologies are illustrated in figures 2.2 and 2.3 below for better understanding. All circular nodes in the figures are represented as routers and black squares are resources.

The well known basic topologies are the *ring* where the routers connect forming a ring and star, where all routers in network are connected to main central router. *Mesh* topology is the most common form of direct topology in network-on-chip.

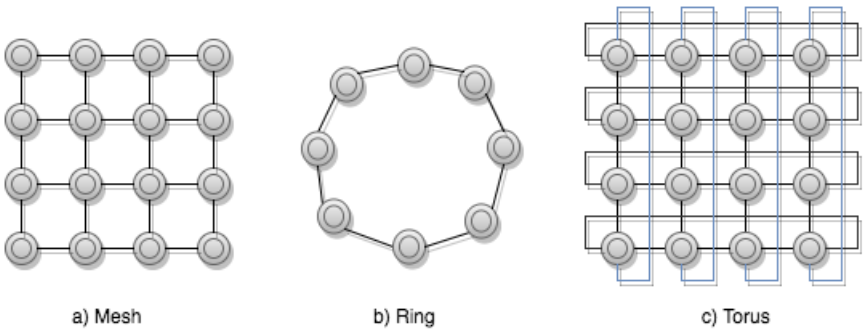

Figure 2.2: Direct topologies.

In this, each cores is connected to one router via network interface. The router is connected to its four neighboring routers in four different directions, viz. east, west, north, south except routers in corners which are connected to two. The major drawback of this topology is the long distance between the edge routers. This drawback was overcome in *torus* topology, where routers on edges are connected to each other directly. This topology have problem of uneven link length which could be controlled by *folded torus.* The *tree topology* and its variations are part of indirect topology, the two common tree topologies are *binary tree and fat tree.* In the binary tree topology the top most router is the parent router or resource and nodes below it are its descendants. Fat tree structure shows more balance and has less traffic congestion even with more nodes added to it. However, its major drawback is the complexity in design for real implementation.

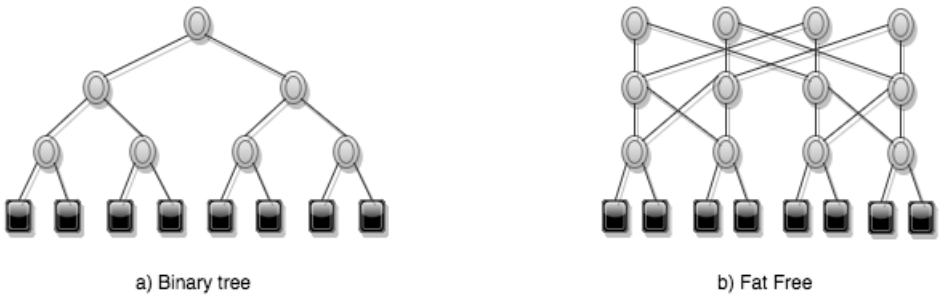

Figure 2.3: Indirect topologies

2.0.2 Routing

Routing algorithms establish the path to travel by packets from its source to destination. There are many routing algorithms available and discussion of all is out of

scope of this dissertation. In this section we will introduce some common routing algorithms terms with main focus on XY routing which is used in our case study. The selection of the route for a packet through the network is defined by its routing algorithm. Based on routing decisions the algorithm can be *Distributed, Source, Hybrid or Centralized*. In distributed routing, the function is calculated as the the packet travels through the network, While in Source routing, the algorithm is decided prior to the packet entry in the network. Hybrid is a multiphase routing where the address can be calculated at the source but paths are decided by the routers like in distributed routing. Centralized routing has a main router as controller where all decisions are made for routing. The routing decisions are not only limited to the address information but on the basis of adaptivity the routing is categorized as *oblivious, deterministic and adaptive*. In oblivious routing any change encountered during the travel of the packet through the network does not change the routing function. An oblivious routing can be deterministic but vice versa is not true. In adaptive routing any change observed in path can alter the route of packet. For example, packet can bypass the congested path with another route to reach the destination. It offers better reliability at the cost of complex routing function. The deterministic routing algorithms have fixed predetermined route. The common form of deterministic routing function is source routing and XY routing. In XY routing a packet is first traveled along the x-dimension up to correct column and then along the y-dimension up to the row. Deterministic and oblivious routing algorithms are minimal as they provide shortest path from source to destination. For example, let (X1, Y1) and (X3, Y4) be the source and destination coordinates respectively. Consider a 2D mesh topology, for the X-Y routing first the packet travel in X direction until X1 reach to X3 and then it travel in Y direction until Y1 becomes equal to Y4 as shown in figure 2.4. Routing algorithms with complex functions can create a situation where a packet stay in network without reaching the destination. This state is known as *livelock*. Another critical situation is of *deadlock and starvation*. *Deadlock* is a situation, where two or more packets wait for a busy resource and at the same time also reserve a resource meant for another packet. Therefore, the network comes to halt because of no packet movement. Starvation is a condition in which a low priority packet may have to wait for its

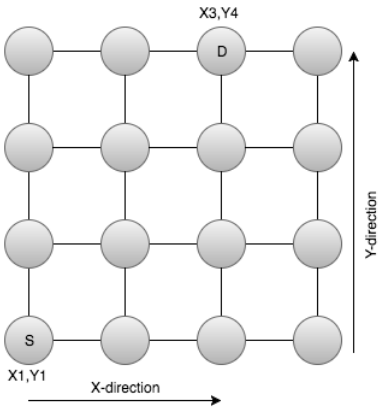

Figure 2.4: An example of XY routing algorithm.

turn forever. XY routing is best example of a deadlock-free routing algorithm with no cyclic dependencies in packets routing.

2.0.3 Switching

Switching is a technique that decides the transmission of data from source to destination nodes. Switching is broadly classified as Circuit and Packet switching. In *circuit switching* approach the whole path is pre-established and reserved between source and node before the actual transmission of data. The other switching technique is *packet based*, in which the flits of packets are ready for transmission once the header makes connection between routers, this can be done again in two ways, either by storing the flits and forwarding them when there is successful connection between routers or making the flits transmission in a pipeline fashion. The former method is called as the *store-and-forward*, sufficient buffer size is necessary to store the whole packet before transmission permission is granted. In latter method of *wormhole switching*, the flits are send over transmission path as soon as header is encountered at the node. Wormhole switching is also a variant of *virtual-cut-through mechanism*. This method require buffers for storage of packets which are passed only upon confirmation to next node. In this dissertation, wormhole switching is used in the Noc.

As per the architectural parameters discussed above the case study use 2x2 mesh topology with XY-Routing and Wormhole switching. The message is divided into packets of maximum length 2^{12} with each flit of 32 bit. The Packet format chosen for the case study is shown in the figure 2.5. In figure 2.5 Type = Type of the packet

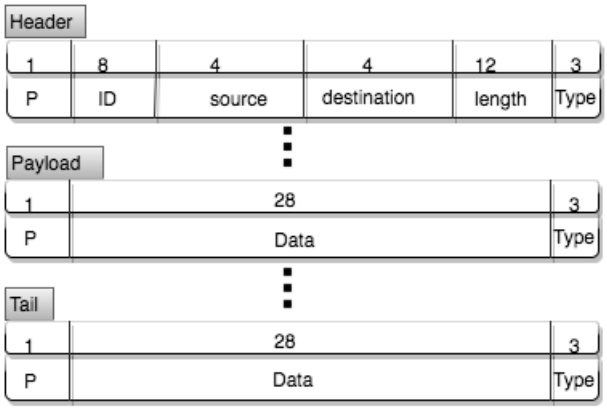

Figure 2.5: Packet structure for NoC

(one-hot), length= Length of the packet in number of flits = Header+Body+Tail, Destination = Address of the destination Node, Source = Address of the source Node, ID = One-hot Counter that is generated by NI and keeps track of the order of sent packets, Data = The DATA (Payload). In this work we have used router design that utilize a simple architecture without Virtual Channels (VCs) with 4 pipeline stages. It is written in VHDL with simple design (without Virtual Channels (VCs)) and with CTS/RTS handshaking for flow control.The high level NoC router architecture is shown below in figure.2.6. With the knowledge of above flit

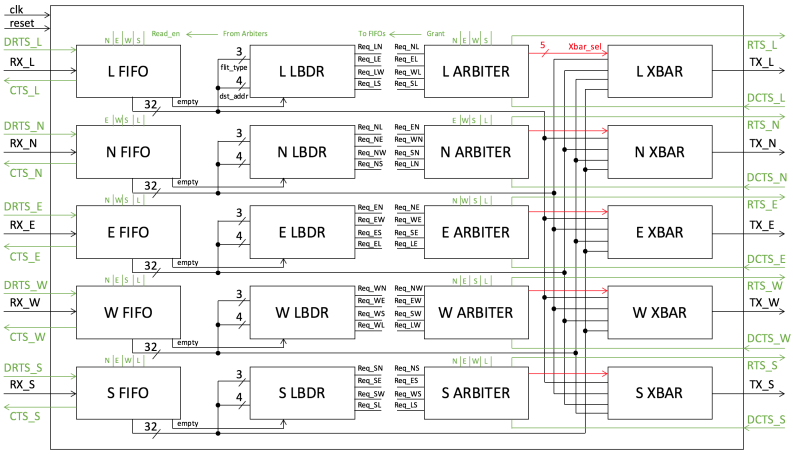

Figure 2.6: Architecture of router for NoC.

structure and NoC router architecture the ECC techniques and Checkers design is decided.

Chapter 3

Faults and Fault Tolerance

In electronics a fault is defined as a flaw that occur in a hardware or software component. A fault not necessarily causes an error in the system, whereas failure can occur due to an error [48]. In this chapter the fault types are classified and the most common fault sources are discussed and different techniques for fault tolerance are explained from which we can choose for the target NoC [6].

3.1 Fault sources and classification

Faults are broadly classified into transient, intermittent and permanent faults depending upon the duration and location of their occurrence. The source of faults generally depends on the phenomenon causing it, for example, It can occur during a manufacturing process due to lithography variation, resulting in spot or bridging defects, which are permanent in nature. Also other sources include physical changes in the circuit during operation due to electro-migration or increased power consumption. Among other sources are the internal noise in the circuit that can be caused by high frequency, timing uncertainty, crosstalk or due to other parts in the circuit whereas external noise, which is not considered as severe noise is due to the medium radiations in which chip is to be used for e.g. space, nuclear plants etc. These fault types are described briefly in the following subsections.

3.1.1 Transient faults

Transient faults (soft errors) are temporary faults also known as Single Event Upset (SEU). The term SEU explains the fact that soft errors are caused by single events such as absorbed radiation, etc. Such errors occur for a short duration of time due to various factors such as process and temperature variations [14] , capacitive and inductive crosstalk noise [15], alpha particles, neutrons or other radiations [16], other noise sources such as Electro-Magnetic-interference (EMI), voltage fluctuation, and electrostatic discharge[17]. The most common example of transient fault is flip of single bit. It is difficult to detect its occurrence because of its random nature. It is generally measured in terms of soft error rate *(SER)*, that tells the probability of error occurrence. Transient faults are the most frequently occurring fault in NoC; up to 80% of all the system failures are associated with transient faults[13].

3.1.2 Intermittent faults

Intermittent faults are periodic in nature but not continuous like permanent faults. It can occur at any time due to temporary environmental change, certain input combinations or conditions making it hard to detect in system. For example, It can be caused by change in certain parameters of the environment such as temperature or voltage change, that alter the normal behavior of the chip. In some cases it can be a considered prior stage to the permanent faults. For instance, due to repeated increase in the resistance wire, it can in turn cause permanent breakdown of wire. The general way to recover from it is to either replace or bypass the faulty circuit in the system [3],[4].

3.1.3 Permanent faults

Permanent faults once occurred cannot disappear from the system. It usually results from manufacturing defect or due to electro-migration, aging of components or due to broken components. Intermittent faults when occur repeatedly for longer duration can lead to the occurrence of permanent errors. Generally, chips are tested during manufacturing in order to detect the permanent faults and rejecting

the faulty circuits. If fault is observed during the operation of a circuit, it either has to be replaced or provided with very strong fault tolerance techniques[3], [4].

3.2 Fault Tolerance Techniques

Fault tolerance strategy is broadly divided in two categories. *static fault tolerance, and dynamic fault tolerance.* [48], [52], [6].

3.2.1 Static Fault Tolerance

The Static fault tolerance methodology is built into the system to counter the effect of faults. This method usually make use of some type of redundancy which is categorized into hardware, information and time redundancy. The type of redundancy to be used depends upon the particular requirement of the system, sometimes a combination of these can also be used.

Hardware Redundancy : This methodology makes use of a physical redundant module with a verifier circuit. The size of module can vary from complete physical circuit to a certain part of the circuit or a logic gate. The most common example of this methodology is triple-modular-redundancy(TMR) that has three redundant modules with a verifier. n-modular redundancy is a more generalized form, with N number of redundancy. In case a module fails to give correct output, the verifier compares the result with other redundant modules output, deciding correct output for the system. The major drawback of TMR is if there is an error in voter circuit, it can result in complete failure of the system. Online testing of the verifier circuit could solve the purpose of internal faults detection, but crosstalk between the module output and verifier input has severe impact resulting in the wrong verifier decision [48].

Time Redundancy : In this methodology the redundancy is in terms of time where the system is made to perform same operation for certain time and later the results are compared. This method does not have any extra copy of circuit resulting in no area overhead but huge latency.

Information Redundancy :The redundancy in this methodology is added as additional bits to the information bits, based on ECC techniques. These techniques

are described in detail later in chapter 5. The forward error correction (FCC) is a technique that makes use of ECC to detect errors and correct at its location.

3.2.2 Dynamic Fault Tolerance

Dynamic fault tolerance is a methodology implemented on active operations of the system. This method follows a set of sequences in its operation as shown below: [13], [6].

Fault detection → Fault Location→ Fault Containment → Fault Recovery

This method requires special control circuitry and elements to be built for the system, which are usually very complex in its design, but offers better reliability in terms of detecting simultaneous permanent and multiple errors with small area overhead as compared to static fault tolerance approach. It is considered as a promising method for networks-on-chip. Each step of the sequence is explained below.

Fault Detection : The fault detection can be implemented in any one of the following forms [52].

- periodic tests : Tests are performed occasionally and output is verified.
- self-checking circuits: Duplication in terms of circuit or time with comparison for output verification.
- watchdog timers : It checks the timer value and resets when its value has exceeded. It is commonly used in multiple processor systems.

The fault tolerance mechanism is decided depending upon the type of error being addressed. Error can be unidirectional, bi-directional, single or burst and various codes for error detection are discussed here. A code with total n number of bits will have m ones and n-m zeros. These code are useful in single and multiple unidirectional errors detection, for example in an online error detecting adder a code of 1-out-of-3 has been used [28]. The berger code is another way to detect all the possible unidirectional errors. This code count the number of 1's in the word and append a complement of the count to the word. For this code the number of check

bits are calculated as $[log_2(k+1)]$ which is least as compared to all available separable codes for detecting unidirectional errors [52]. In order to have much less check bits codes for e.g. the modified berger code, borden code and bose-lin codes were formulated [52].

Error control codes are widely used to detect errors in data. The most common form of ECC is single parity where one bit is appended to the data based on even or odd parity. There are many more sophisticated ECC available which will be discussed later in this section [52].

Checkers are special circuits with robust design used to determine error by comparison of the two values which are generated as per the concept of fault-secure or self-testing design methods. These method ensures that a valid input generates a valid output through the circuit. An invalid result is easily observed by this method as they do not fulfill the codeword requirement of this method. A self-testing circuit, indicates fault for certain combination of inputs. In this dissertation the concept of checkers are used and implemented in the case study and is described in more detail in this chapter [52].

Fault Location: This step is necessary for error correction and recovery. Sometimes fault location can be found during detection or using self diagnostic methods after detection [52] . ECC can also be used in different combination in NoC for locating the faults.

Fault Recovery : There are three common ways to accomplish fault recovery namely, ARQ, FEC and Hybrid ARQ. ARQ is a method that request for a retransmission upon an error detection. ARQ is suitable for error recovery from transient and intermittent errors but not permanent error because it will give same error no matter how many times it is repeated. FEC techniques are used to recover from error by its correction. FEC schemes cause area overhead because of complex encoder and decoder designs whereas, ARQ method result in huge latency. However, third type of error recovery is popular but costly where Combinations of FEC schemes can be used with timeout ARQ [52], [28]. In the following chapters the concept of error control codes and checkers for NoC is explained.

Chapter 4

Concurrent Online Checkers

This chapter provides basics for the understanding of checkers. Its implementation with the functional module and role in fault detection is explained. In figure 4.1 checkers are connected with the functional module through its input and output ports. Checkers contain logical assertions based on the input and output ports of the functional module [40].

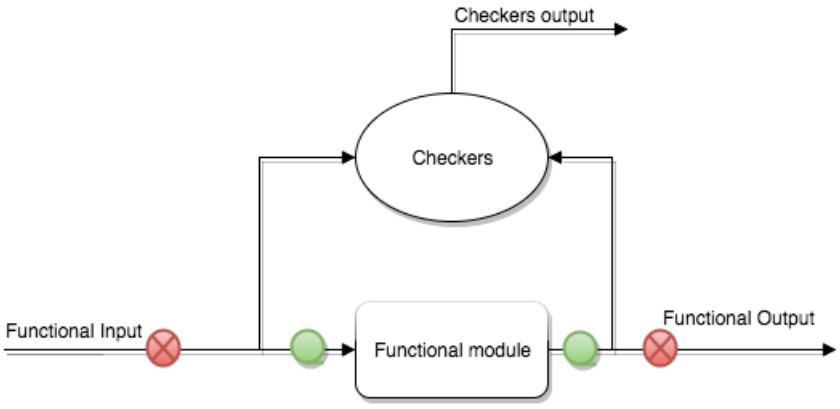

Figure 4.1: Concurrent checker logic.

In figure 4.1 the logic checks for fault at input or output of each gate of the functional logic, as shown in green circle. On the other hand it do not check faults occurring in functional inputs and output. The terminology for the possible scenarios of fault detection of the injected faults [40] are describes as follows:

- **True Detection** : The fault is detected both by circuit and checkers.
- **True Miss**: The fault is detected only by circuit but not by checkers.

- **False Positive** : The fault is detected only by checkers but not by circuit.
- **Benign Miss** : The fault is detected neither by circuit nor by the checkers

The faults detected by the circuit is a case, in which for a given fault and input test pattern the fault is propagated to functional outputs of the circuit. In order to check the effective behavior of the checkers model, indices used in this work are taken from the work of [40], [53] denoted as, CEI - Checkers Efficiency Index, FC - Fault Coverage, FPR - False Positives Ratio. If we denote D as true detections by checker for a set of input pattern. F is the fault positive, W and X as true misses and benign misses. Based on these notations, mathematically the indices are given as follows:

$$CEI = \frac{D}{(D+W)}, FC = \frac{D+X}{(D+X+W)}, FRP = \frac{F}{(F+X)}$$

These indices values are obtained in form of outputs from the evaluation tools Turbo Tester (TT) and set of tools used in this work. TT is a software package with low cost approach used for testing and diagnostic purposes. In this work, TT is used to analyze the set of input test patterns using 'analyze' command and to perform fault simulations on the design circuit-under-test. TT can read the schematic from various VLSI tools (in this work synopsis). It present the circuit design in terms of Structurally Synthesized Binary Decision Diagrams (SSBDD). The concept of SSBDD is explained in following section [55], [32]. To learn more about the tool, please refer to [51].

4.1 Structurally Synthesized Binary Decision Diagrams (SSBDD)

4.1.1 SSBDD Structure

SSBDD is an extended class of Binary Decision Diagrams (BDD). Unlike BDD, it is not canonical, but provides the complete structural information necessary to identify any problem in circuit and it do not suffer from memory explosion. SSBDD representation is best suitable for complex designed circuits, and it guarantees

improved test generation and fault simulation efficiency. It is based on the concept of macros i.e sub-circuits with fan-out free regions (FFR). This model is suitable for Logic level simulations, unlike gate level simulations done in conventional BDD. A SSBDD is constructed using a graph superposition procedure. For this a circuit with gate level description is divided into sub-circuits with FFR. Each sub-circuit has its own SSBDD, thus the final circuit is set of SSBDD's [55]. Explanation of FFR is given in appendix B.

4.1.2 SSBDD Simulation

The baseline TT suite provides a tool named analyze, to perform fault simulation for combinational circuits. This is done using Algorithm 1 [55] as explained below:

The enhanced analyze is used in order to identify the presence of online checkers

Algorithm 1 Fault Simulation

```
1: Perform fault-free Logic Simulation (Alg.2)
2: for each SSBDD in the model do
3:       take the first node of current SSBDD
4:       while current node is not a terminal node do
5:          evaluate the current node
6:          inject corresponding fault
7:          perform logic simulation to evaluate output (to evaluate fault prop.)
8:           If current output differs from fault-free output then
9:                mark currently considered fault as detected
10:          end if
11:         remove currently considered fault
12:         move to next node
13:      end while
14: end for
```

Algorithm 2 Logic Simulation

```
1: for each SSBDD in the model do
2:       take the first node of current SSBDD
3:       while current node is not a terminal node do
4:          evaluate the current node
5:          move to proper next node
6:       end while
7:       save output value of current SSBDD
8: end for
```

logic in the considered design, and thus produces new statistics and information

concerning detection capability of the checkers. Since analyze simulates stuck-at faults in combinational circuits, for a sequential circuit, a pseudo-combinational version of it is devised, which can be obtained by breaking the flip-flops in design. Baseline analyze tool requires the description of currently considered design, in terms of *.agm* SSBDD representation file, and an appropriate file listing the test patterns is provided with *.inp* extension. The current extended version of analyze considers no parallelization during fault simulation, in order to avoid memory issues and grant the possibility to evaluate large designs. Also, it considers no fault dropping, every fault is simulated with every input vector (test pattern). This is because in case of using the concurrent online checkers, we want each single fault to be simulated using all the possible input test patterns to calculate the final fault coverage, therefore, in our fault simulation tool (analyze), no fault dropping is performed. It is also worth noting that during normal (fault-free) simulation of the circuit, the checkers should not fire, otherwise, that would mean either there is a bug in the checkers themselves, or the test pattern environment is not correctly specified, i.e. it contains some invalid test patterns. It is important to stress the fact that checker logic is meant to be effective during the normal operation activity of the functional logic to which it is applied, when inputs are totally unpredictable. Only in the evaluation of their behavior, checkers are fed with a predetermined set of stimuli. These considerations led to the introduction of new paradigm of statistics to describe fault detection capabilities of concurrent online checker logic, including a reformulation of fault coverage (FC) and introduced of two other new metrics, checkers efficiency index (CEI) and false positive ratio (FPR), explained on page 19.

The possible detection outcomes after the injection of a fault, are extended to an alphabet consisting of 7 symbols, to be used in the computation and evaluation of the outputs of the tool, listed as follows:

- 0 - stuck-at-zero fault is detected by the circuit and by the checkers (True Detection).
- 1 - stuck-at-one fault is detected by the circuit and by the checkers (True Detection).

- w - stuck-at-zero fault is detected by the circuit and not by the checkers, which can be a critical case. (True Miss).
- W - stuck-at-one fault is detected by the circuit and not by the checkers, which can be a critical case (True Miss).
- o - stuck-at-zero fault is not detected by circuit and is detected by the checkers (False positive), which can cause some performance degradation.
- i - stuck-at-one fault is not detected by circuit and is detected by the checkers (False positive), which can cause some performance degradation.
- X - none of the stuck-at-faults is detected, nor by the circuit, neither by the checkers, i.e. the node is not tested with the considered input pattern (Benign Miss). Based on the introduced alphabet of symbols, the newly devised metrics for coverage - CEI, FC, and FPR, are defined as follows.

$$CEI = \frac{\Sigma[0,1]}{\Sigma[0,1,w,W]}, FC = \frac{\Sigma[0,1,X]}{\Sigma[0,1,w,W,X]}, FPR = \frac{\Sigma[o,i]}{\Sigma[o,i,X]}$$

where each symbol represents the number of occurrences of the symbol itself, i.e. of the corresponding situation. FC, CEI and FPR are among the outputs of the enhanced analyze tool for concurrent online checkers evaluation, but the previously introduced alphabet of symbols, to describe detection outcomes after the injection of a fault, led to devise further useful information as output of the tool.

4.2 Enhanced analyze tool outputs

In addition to the previously introduced newly devised statistics, CEI, FC and FPR, the tool generates the following output information [40]:

4.2.1 Fault Table

Fault table presents one row for each input vector and one column for each node in the SSBDD representing the considered design. Detection of Stuck-At-Faults (SAFs) is extensively described, marking with the previously introduced alphabet of symbols the detection outcome for each node where faults are injected, using the

input pattern corresponding to the row. Further symbols are introduced to label those nodes where faults are not injected: $*$: fan-out input nodes , #: fan-out output nodes and $-$: checker logic nodes.

4.2.2 Nodes Detection Information

Nodes detection information is extracted from the fault table, for those nodes where faults are injected, while the remaining nodes are simply listed. For any node the number of occurrences of the introduced alphabet is reported, omitting zero entries. This information allows to spot those nodes which eventually present a large amount of true misses (w and W occurrences), the worst situations in which checkers do not detect critical faults, thus suggesting where to act in order to increase the detection effectiveness of the checking logic.

4.2.3 Checkers Detection Information

Checkers detection information portrays the capabilities of each checker in detecting faults, in form of a table, with a row for each checker and a double column for each node. Each double column gives information for stuck-at-0 fault on the left side and stuck-at-1 fault on the right side, for the corresponding node, if faults are injected in that node. Each intersection between a row and a double column provides the numbers of detection for stuck-at-0 and stuck at-1 faults, injected in the node corresponding to the column, provided by the checker corresponding to the row. Checkers detection table could also be used to spot eventual overlapping or independence in the action of different checkers.

4.2.4 Checkers Detection Absolute Weights

Processing the previously introduced table row by row, information about the absolute weights of checkers detection is produced. Absolute weight is evaluated for each checker, as the number of provided detections over the considered set of stimuli, giving a first hint on the capabilities of detection of the checker itself. On one hand it would be wrong to suppose a checker better than another one only considering the number of provided detections, because a checker firing for a limited number of

faults could be the only one detecting those faults. On the other hand this information may suggest some optimization work, based on a heuristic approach, trying to use a limited set of checkers derived from the whole. For instance, it could be interesting to start using at first only the checker providing the highest number of detections, and gradually increasing the size of the used set of checkers, keeping trace of the evolution of both coverage information and area overhead.

4.2.5 Fault Free Checker Firing Information

The fault free simulation behavior of checkers is evaluated, in search of eventual undesired firings. This information is not related to the evaluation of the detection capability of the checkers, but it could be useful to catch two different possible erroneous situations. First, an input vector represents an incorrect situation, thus suggesting the possible need of change in the test set of input patterns. Second, a checker is either meaningless or wrongly designed, thus suggesting the need of removing, replacing or modifying it.

4.2.6 Input Patterns True Misses Count

There is a strong dependence of the behavior of each checker from the considered set of inputs stimuli. For this reason it is reasonable to keep track of the number of occurrences of true misses, depicted by symbols w and W, for each input pattern. This way the distribution of true misses over the adopted set of vectors could be outlined, highlighting two different classes of patterns . First is, true misses free patterns, which could be eventually used to check periodically the correct functionality of the checker logic, that could begin missing some faults in case of damages or failures in its circuitry. Second as,true misses suffering patterns, which could outline the most critical valid operative conditions for the functional logic under study. Finally the tool reports the number of meaningful nodes over the total, altogether with the corresponding number of SAFs injected during fault simulation, and the number of evaluated input stimuli.

Chapter 5

Error Control Codes

Error control codes (ECC) have been widely used in communication systems. In ECC redundant check bits are calculated based on input data. The input data and check bits are transmitted across a noisy channel. In the receiver, an ECC decoder is used to detect errors induced during transmission. A powerful ECC usually requires more redundant bits and more complex encoding and decoding processes which increases the area overhead. For better reliability and performance, ECC must be carefully selected to meet the expected requirement of the NoC.

5.1 Single Parity

The single parity (SP) is one of the simplest error control code. In SP, an additional parity bit is added to a $k-bit$ data block such that the resulting $(k+1)$-bit codeword has an even number (for even parity) or an odd number (for odd parity) of 1's. SP codes can be used only for error detection. In the decoder, an even/odd parity check bit is recalculated and compared to the transmitted parity check bit. If the recalculated parity check bit is different from the transmitted parity check bit, errors are detected. SP codes can detect all odd numbers of errors in the codeword. The encoding and decoding process of SP codes can be easily realized by simple XOR trees [7].

5.2 Cyclic Redundancy Codes

Cyclic codes are the redundancy check codes used for only detecting errors in data. Cyclic codes are a type of linear block code where a cyclic shift of a input data generates a new code which is appended with the output data as redundancy bits before transmission. It is realized through a process of long division which can be implemented using a digital logic circuit consisting of XOR gates and linear feedback shift registers.

In CRC, to verify data integrity, the frame is treated as one very large binary number, which is then divided by a generator polynomial number. This division produces a remainder, which is transmitted along with the data. At the receiving end, the data is divided by the same generator number and the remainder is expected to be zero this time. If the remainders is not zero, then an error has occurred during the data transmission.[11]. To learn more about CRC algorithms refer to [9] [11]. The implementation of CRC is shown in chapter 6.

5.3 Hamming Codes

Hamming codes are a family of linear error-correcting codes invented by Richard Hamming in 1950. Simple hamming codes can detect and correct errors when the error rate is low. For instance, in computer memory, where bit errors are extremely rare, hamming codes are widely used. An extended hamming code with one extra parity bit is often used. Extended hamming codes allows the decoder to distinguish between when at most one one-bit error occurs and when any two-bit errors occur. In this sense, extended hamming codes are single-error correcting and double-error detecting (SECDED).

It involves transmitting data with multiple check bits and decoding the associated check bits when receiving data to detect errors. The check bits are parallel parity bits generated from XORing certain bits in the original data word. If bit error(s) are introduced in the codeword, several check bits show parity errors after decoding the retrieved codeword. The combination of these check bit errors display the nature of the error. In addition, the position of any single bit error is identified from the check bits. To know more about SECDED refer to [20]. The implementation of

Hamming code is explained in chapter 6.

5.4 Bose-Chaudhuri-Hocquenghem Codes

Error correcting codes capable of correcting multiple errors are needed. Popular linear block codes for multiple error correction are the BCH codes, which are also class of cyclic codes. They can be easily constructed according to specifications for correcting as many errors as is required. The BCH code is similar to Hamming code when used as a single error correcting code, actually Hamming codes are BCH codes with $t_c = 1$, but commonly codes with $t_c = 2$ are called BCH codes. In on-chip signaling, where only two logic states are possible, only binary BCH codes are of interest [27].

The syndrome decoding in hamming is limited by the size of the minimum error vector table. When the number of check bits is high the table is impractically large, and therefore alternative decoding methods should be used. The decoding of BCH codes can be done using the Berlekamp-Massey (B-M) algorithm. The algorithm is iterative requiring $2t_c$ iterations, where t_c is the error correction capability of the code. The calculations are done in Galois Field $(GF)2^m$, where m depends on the length of the code ($2^m \geq n$). In addition to the actual algorithm a pre-processing circuit is needed. Using Fourier transform in $GF(2^m)2t_c$ syndromes are calculated. To read more about BCH refer to [6],[27].

5.5 Reed-Solomon Codes

The Reed-Solomon codes are other cyclic codes that can be used to correct multiple errors. The codes are non binary, which means that instead of bits, groups of m bits are used as symbols for the codes. The Reed-Solomon codes are optimal meaning that they provide the maximum distance at the used number of check symbols. If a word contains k groups of data and its length is n groups, at most $[(n-k)/2]$ errors can be corrected. Because on-chip signals are binary, each symbol must be coded by binary bits. These binary-coded non binary codes are especially effective in correcting burst errors, since the detection and correction is based on symbols which consist of many adjacent bits. On the other hand, they are not very effec-

tive in single error tolerance because a single bit fault in a binary-coded symbol takes the whole correction capability of that symbol. Binary codes, such as binary BCH, provide the same tolerance against single faults with a lower number of check bits[27] .

Decoding of Reed-Solomon codes is also based on the Berlekamp-Massey algorithm. The main difference is that in addition to the error vector e also the error values are needed to perform the error correction. The error vector $\vec{e}$ points the erroneous $GF(2^m)$ symbol and the actual correction is done by adding the error value of that particular symbol and the transmitted symbol itself. The error values can be extracted using the Forney algorithm with inputs obtained from the B-M algorithm. Also the fourier transform requires slight changes compared to the one used for BCH decoding. In Reed-Solomon decoding all the calculations are done with $GF(2^m)$ symbols while for binary BCH codes the syndrome calculation is just calculating parities of different sets of bits [27].

Chapter 6

Implementation of ECC

6.1 Hamming Codes Implementation

In this proposed method 8 parity bits are generated for 32 bit information data to send 38 bit data string for transmission at source by using even parity check method [20]. The minimum number of check bits required for a single bit error correction is derived from the following equation:

$$D + P + 1 \leq 2P$$

The above equation is of extended hamming code which can correct single bit errors and detect double bit errors also known as single error correction and double error detection (SECDED). Parity bits are added to a D-bit data word, forming a code word of D+P+1 bits. Parity bits are calculated by XOR operation based of combination of data bits. Combination of data bits are shown in Table 6.1 and calculation of parity is shown in following equations.Calculation of parity is done using the combinations as mentioned in table, represented for p0 in following equation [20].

$p0 \leq datain(0) \oplus datain(1) \oplus datain(3) \oplus datain(4) \oplus datain(6) \oplus datain(8) \oplus datain(10) \oplus datain(11) \oplus datain(13) \oplus datain(15) \oplus datain(17) \oplus datain(19)) \oplus datain(21) \oplus datain(23) \oplus datain(25) \oplus datain(26)$

Data word is applied as an input in the encoder circuit which performs XOR operations on the given data word and thus the required parity bits are generated from the parity generator. Parity bits and data bits together form the code word.This

Table 6.1: 32 bit data combination for parity calculation.

Input	P0	P1	P2	P3	P4	P5	P6
d0	✓	✓					✓
d1	✓		✓				✓
d2		✓	✓				✓
d3	✓	✓	✓				✓
d4	✓	✓	✓				✓
d5				✓			✓
d6	✓	✓		✓			✓
d7			✓	✓			✓
d8	✓		✓	✓			✓
d9		✓	✓	✓			✓
d10	✓	✓	✓	✓			✓
d11	✓				✓		✓
d12		✓			✓		✓
d13	✓	✓			✓		✓
d14			✓		✓		✓
d15	✓		✓		✓		✓
d16		✓	✓		✓		✓
d17	✓	✓	✓		✓		✓
d18				✓	✓		✓
d19	✓			✓	✓		✓
d20		✓		✓	✓		✓
d21	✓	✓		✓	✓		✓
d22			✓	✓	✓		✓
d23	✓		✓	✓	✓		✓
d24		✓	✓	✓	✓		✓
d25	✓	✓	✓	✓	✓		✓
d26	✓					✓	✓
d27		✓				✓	✓
d28	✓	✓				✓	✓
d29			✓			✓	✓
d30	✓		✓			✓	✓
d31		✓	✓			✓	✓

codeword is the input for the decoder. The incoming 28-bit data along with the 8-bit parity are XOR'd together to generate the 8-bit syndrome (S1 through S8). Syndrome first bit equation is shown below:

$syndrome(0) = decin(0) \oplus decin(1) \oplus decin(3) \oplus decin(4) \oplus decin(6) \oplus decin(8) \oplus decin(10) \oplus decin(11) \oplus decin(13) \oplus decin(15) \oplus decin(17) \oplus decin(19) \oplus decin(21) \oplus decin(23) \oplus decin(25) \oplus decin(26) \oplus decin(28)$

Then the syndrome search for the error type and the error location. The value of

Table 6.2: Syndrome values for error detection.

syndrome	Overall Parity (P5)	Error Type	Remarks
0	0	No Error	
!=0	1	Single Error	Correctable.
!=0	0	Double Error	Not correctable.
0	1	Parity Error	Overall Parity.

the syndrome indicate the bit position in error. If all of the elements of the syndrome vector are zeros, no error is reported. Any other non-zero result represents the bit error type and provides the location of any single bit errors. It is then used to correct the original incoming data. Table 6.2 illustrates the value of the syndrome and overall parity bit in detecting both single and double errors.

6.2 CRC Implementation

The cyclic redundancy check (CRC) is a technique used for detecting errors in digital data, but not for error corrections. In the CRC method, a specific number of check bits are concatenated with the data stream before transmission. In receiver, check bits are verified for any error. In CRC, the input data is treated as binary number, which is divided by a generator polynomial number. This division produces a remainder, which is transmitted along with the data. At the receiving end, the data is divided by the same generator number and the remainder is expected to be zero. If the remainders is not zero, then an error has occurred during the data transmission.[11]

The encoder generates a check sequence number from the given input data stream. The encoding process starts by dividing the information bits by the predefined generator number. The encoder then concatenates the check sequence number with the input data. The serial division requires a number of clock cycles equal to the number of information bits in order to calculate the check sequence, however the parallel one requires only one clock cycle but number of wires equal to the number of data bits[11].

The CRC decoder is similar to encoder, based on division method. The decoding process starts by dividing the received check sequence number by the generator number. If the remainder is zero, assume there is no error detected. For nonzero re-

mainder, it means that the received check sequence number got an error detected.[9] [11].

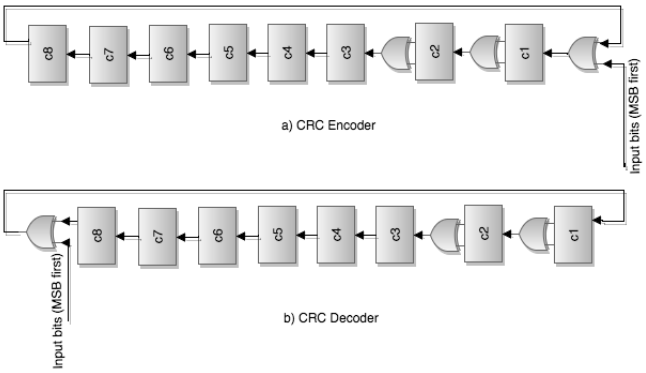

Figure 6.1: CRC encoder and decoder block diagram.

Choosing a good generator polynomial is something of an art and for our implementation we choose CRC8 : the ATM-8 HEC polynomial $0x83 = x^8 + x^2 + x + 1$, is said to be from ITU standard I.432 in bits it is written as "100000111". The input data of 20 bits is divided by this polynomial to obtain a CRC of 8 bits which is appended with output data before its transmission. The CRC code is generated through a process of long division. This is implemented using a digital logic circuit consisting of exclusive-or (XOR) gates and shift registers.The shift register is a string of 1-bit storage devices, each with an input line and an output line. At specific clock times, the stored bit in each register is shifted out and is replaced by a new bit from the input line. After complete frame of data pass through all the registers it gives the final value of CRC. The xor gates position with the shift registers depends on the polynomial. For $x^8 + x^2 + x + 1$ the xor gate is present before and after first,second and third shift register. Figure 4.2 shows the block diagram of CRC encoder and decoder[9] [11].

6.3 Single Parity Implementation

The single parity is the most widely used error detection techniques for the data transmission. It can detect single bit errors. When binary data is transmitted, it

may be subjected to noise that can change $0's$ to $1's$ and $1's$ to $0's$. Hence, a parity check bit is added to the data stream that makes the number of $1's$ either even or odd. A parity generator is a combinational logic circuit. The parity bit is generated in the transmitter and checked in receiver.

In the implementation of parity circuits the sum of data bits is checked by xor of all bits. If it is odd number of $1s$ its sum results in 1 and 0 if even. It is combinational circuit that add one extra bit (parity bit) to the input data stream by xor all input bits. In proposed design we used even parity bit which is calculated as 0 if there are even number of $1's$ and 1 if there are odd number of $1's$ in the data stream. The design is made using VHDL and simulated in Xillinx ISE 10.1. The simulation results are shown in the appendix A.4.

Chapter 7

Implementation of Checkers

7.1 Online fault detection in NoC Router

In this dissertation, we have utilized the concept of online fault detection. It is introduced as an additional circuitry to detect single stuck-at faults (stuck-at 0 (SA0) and stuck-at 1 (SA1)) during the operation of the system. In general, we exploit two types of checkers, one group for the data-path and the other group for the control part of the NoC router. The circuit-under-test is made pseudo-combinational, in which all the sequential components (such as registers or Flip-Flops (FFs)) are broken. The final circuit-under-test obtained would have a set of primary inputs/outputs, plus some pseudo- inputs/outputs. This way we can evaluate the checkers for all possible combinations of valid input stimuli to the circuit, when single stuck-at faults occur at different locations in the circuit. The evaluation of the checkers is based on a framework which takes into account an initial set of checkers for the module under examination. These checkers are introduced based on the functionality of the circuit and the properties that should hold between the inputs and outputs of it (in form of hardware assertions). The main goal is to keep the logic of checkers as simple as possible, thus, it would be desirable that the total area of a checker would not exceed the area of the module it is checking. The framework used for evaluating the checkers tries to find the set of checkers that can achieve a target CEI And FC, while considering the minimization aspect. For more details regarding the checkers and the framework for evaluation of the checkers, the reader is referred to the following publications[40], [46] .

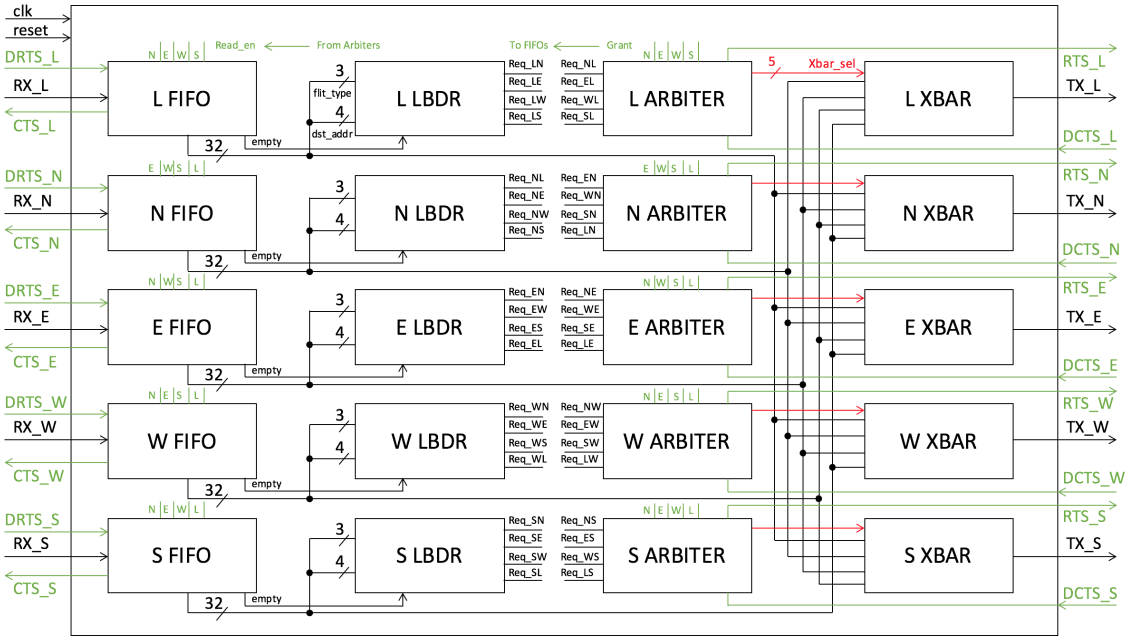

Figure 7.1: HIgh level NoC router architecture for control part.

NoC Router The control part of the NoC router in this dissertation consists of the following components which are also depicted in the figure 7.1:

- **FIFO control part** : It is the input buffer for the NoC router which is sequential in nature, used to store incoming bits. The control part of FIFO contains the read and write pointers, status signals like empty and full, and the read and write enable signals indicating the reading from and writing to FIFO. The FIFO architechture considered in this case is one-hot. To learn more about FIFO refer to[49].

- **Routing Logic (LBDR)**: It is the routing computation unit of the NoC router. it is based on a distributed routing mechanism and our case study use XY dimensioned ordered-routing algorithm. The concept of distributed routing mechanism is explained in section 2.3. To learn more about LBDR refer to [43]

- **Arbitration logic (Arbiter)** : It decide the output port to which the data is to be directed based on the logic it is devised. In this work, we have assumed a finite state machine (FSM) based round-robin (RR) arbiter which is based on dynamic prioritizing, giving highest priority to the L input port,

then N, E, W and finally S and again it starts from L input port. This way of circular prioritizing can guarantee that there would not be any starvation and all input ports will eventually get access to their requested output port.To learn more about Arbiter refer to [49].

In this work, the NoC router architecture is first divided into data-path and control path, On the basis of which we devise checkers for the control part of NoC router. The inputs and outputs extracted from the pseudo-combinational circuit are described in the following sections.All the checkers are devised for the pseudo-combinational version of the circuit. The results of checkers evaluation is shown below in terms of Absolute weight (the number of detections of faults) for FIFO, LBDR and arbiter in figure 7.2, 7.3, 7.4 respectively. The evaluation results for arbiter is shown in appendices.

7.2 Pseudo-combinational version of FIFO control part

We separated the data-path of FIFO from its control part, and propose checkers for the control part. The control part of the FIFO includes the following input and output ports at its interface[50]:

- Inputs

  DRTS (Detect Ready To Send) from previous router/NI

  Read_en (Read Enable) signals from the arbitration modules (read_en_N, read_en_E, read_en_W, read_en_S, read_en_L)

- Outputs

  CTS (Clear to Send) to previous router/NI

  empty_out (Empty signal) to routing logic

Some of the inside components had generated memory modules, which are broken and takes as pseudo-inputs and pseudo-outputs to the interface of the module. Also, for the sake of higher FC and CEI, we took some of the other internal signals

out to the interface in order to check some properties that must hold for them. The resulted pseudo-combinational version of the control part of the FIFO includes the following inputs and outputs.:

- Inputs
  - DRTS (Detect Ready To Send) from previous router/NI
  - Read_en (Read Enable) signals from the arbitration modules (read_en_N, read_en_E, read_en_W, read_en_S, read_en_L)
  - CTS_out (Clear To Send) (which is a pseudo-input, denoting the previous value of CTS)
  - read_pointer (Read pointer of FIFO) taken from internal signals of FIFO (which is a pseudo-input, denoting the previous value of the read pointer)
  - write_pointer (Write pointer of FIFO) taken from internal signals of FIFO (which is a pseudo-input, denoting the previous value of the write pointer)
- Outputs
  - CTS_in (Clear To Send) to previous router/NI (which is a pseudo-output, denoting the current value of CTS)
  - read_pointer_in (Read pointer of FIFO) taken from internal signals of FIFO (which is a pseudo-output, denoting the current value of the read pointer)
  - write_pointer_in (Write pointer of FIFO) taken from internal signals of FIFO (which is a pseudo-output, denoting the current value of the write pointer)
  - empty_out (Empty signal) to routing logic
  - full_out (Empty signal) taken from internal signals of FIFO (taken to the interface for evaluating CEI and FC by checkers)
  - read_en_out (Empty signal) to routing logic (taken to the interface for evaluating CEI and FC by checkers)
  - write_en_out (Empty signal) to routing logic (taken to the interface for evaluating CEI and FC by checkers)

7.2.1 FIFO control part checkers

Based on the primary inputs/outputs and pseudo-inputs/outputs inferred from the logic of the control part of the FIFO, the following checkers are introduced, which correspond to the properties that must always hold. In other words, if the property in a checker does not hold, it becomes one, otherwise, if the logic is working properly, it becomes zero:[50]

- err_FIFO_control_part_DRTS_CTS: When FIFO has not received a request, it can not send Clear to Send (When DRTS is zero, CTS must be zero).
- err_FIFO_read_pointer_update: If there is a request for reading from the FIFO (one of the read_en signals is active) and the FIFO is not empty, read pointer must be updated.
- err_FIFO_read_pointer_not_update: If there is no request for reading from the FIFO or the FIFO is empty, read pointer must not be updated, it must remain the previous value.
- err_FIFO_write_pointer_update: If there is a request for writing to the FIFO and the FIFO is not full, write pointer must be updated.
- err_FIFO_write_pointer_not_update: If there is no request for writing to the FIFO or the FIFO is full, write pointer must not be updated, it must remain the previous value.
- err_FIFO_full_empty: The FIFO cannot be full and empty at the same time.
- err_FIFO_empty: If read_pointer and write_pointer are pointing to the same location, empty must be high (active).
- err_FIFO_full: If read_pointer is pointing to one location after write_pointer in a circular manner, full must be high (active).
- err_FIFO_write_pointer_onehot: The read_pointer of FIFO must follow the one-hot fashion (both the previous value and the current value).
- err_FIFO_read_pointer_onehot: The write_pointer of FIFO must follow the one-hot fashion (both the previous value and the current value).

Table 7.1: FIFO Control Part checkers' Detection Absolute Weight.

Checker name	Checker Weight (no. of detections)
err_FIFO_write_pointer_onehot	4350
err_FIFO_read_pointer_onehot	4160
err_FIFO_write_pointer_not_update	3445
err_FIFO_empty	1200
err_FIFO_read_pointer_not_update	1120
err_FIFO_read_pointer_update	560
err_FIFO_control_part_DRTS_CTS	380
err_FIFO_full	300
err_FIFO_full_empty	160
err_FIFO_write_pointer_update	135

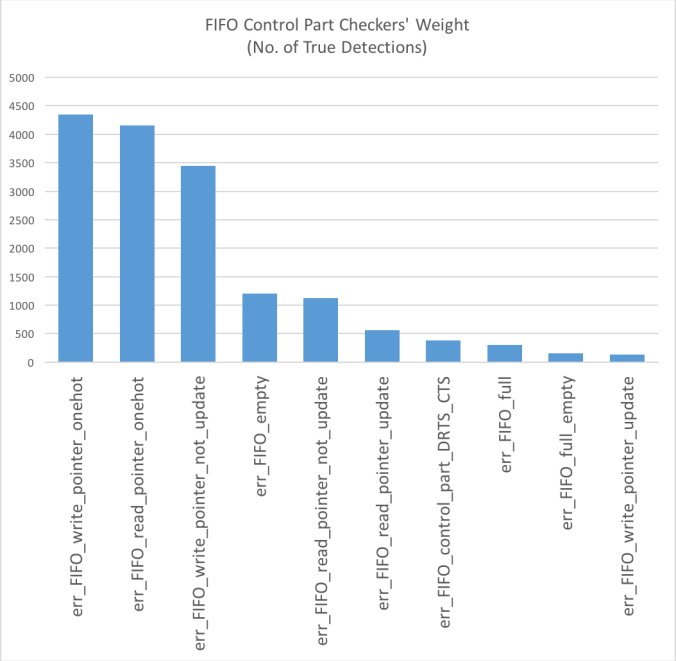

Figure 7.2: FIFO control part checkers detection absolute weights

7.3 Pseudo-combinational version of LBDR

The pseudo-combinational version of the routing logic (LBDR) has the following primary and pseudo- inputs and outputs at its interface [50]:

- Inputs
  - empty, from the input FIFO

  - flit_type, encoded as one-hot, which is extracted from the data coming from FIFO and going to the crossbar switch (XBAR) (001: Header flit, 010: Body flit, 100: Tail flit)
  - dst_addr, which is extracted from the header flit of the packet coming from the FIFO and going to the crossbar switch (XBAR), and denotes the destination address ID of the packet. It is parameterized based on the network size (NoC_size). By default, it can be set to 4 bits, which can cover the address ID from 0 to 15 (0000 to 1111 in binary)
  - Req_N_FF, Req_E_FF, Req_W_FF, Req_S_FF, Req_L_FF, which store the previous values of the requests for output generated by the LBDR logic (because of using XY routing, they must follow the one-hot fashion).

- Outputs
  - Req_N_in, Req_E_in, Req_W_in, Req_S_in, Req_L_in, which store the current values of the requests for output generated by the LBDR logic (because of using XY routing, they must also follow the one-hot fashion).

7.3.1 LBDR checkers

Based on the primary inputs/outputs and pseudo-inputs/outputs inferred from the LBDR logic, the following checkers are introduced (which correspond to the properties that must always hold). Similar to the previous checkers, if the property in a checker does not hold, it becomes one, otherwise (if the logic is working), it becomes zero:[50]

- err_LBDR_Req_onehot: If empty is zero (If FIFO is not empty), Request outputs of LBDR must be one-hot.
- err_LBDR_Req_allzero: If empty is one (If FIFO is empty), all the Request outputs of LBDR must be zero.
- err_LBDR_dst_addr_checker: Depending on the location of the destination node with respect to the current node, if wrong requests from LBDR go active, there is a fault.

- err_LBDR_Req_Local: If the header flit has reached its destination node, the L output request of LBDR must go active and others must be zero

Table 7.2: LBDR checkers detection absolute weights.

Checker name	Checker Weight (no. of detections)
err_LBDR_Req_onehot	1276
err_LBDR_Req_allzero	832
err_LBDR_dst_addr_checker	436
err_LBDR_Req_Local	46

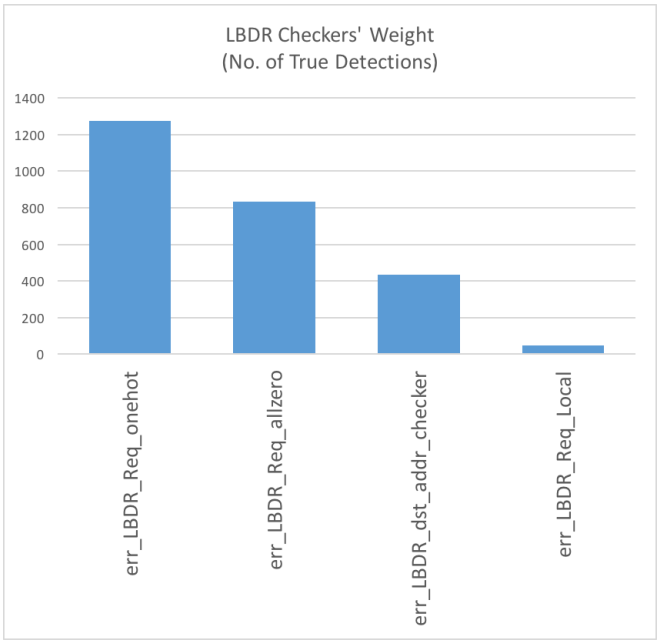

Figure 7.3: LBDR checkers detection absolute weights.

7.4 Pseudo-combinational version of Arbiter

The pseudo-combinational version of the arbitration logic (Arbiter) has the following primary and pseudo- inputs and outputs at its interface:[50]

- Inputs
  - Req_N, Req_E, Req_W, Req_S, Req_L, which denote the requests coming from the LBDR modules of different input ports (because of using XY routing, they must follow the one-hot fashion).
  - DCTS (Detect Clear To Send), which detects the CTS signal from the input FIFO of the next router/NI (for hand-shaking)

  - RTS_FF, which stores the previous value of the RTS signal (used for signaling the next router that the data on the output port is valid)
  - state, which is taken from the internal signal of the Arbiter, it shows previous value of the state of the arbiter (normally it is not encoded as one-hot, but for better detectability of faults, we can assume them as one-hot). State shows the arbiter has in which condition according to the grant given previously (maybe) to one of the requesting inputs.

- Outputs
  - Grant_N, Grant_E, Grant_W, Grant_S, Grant_L, which are the grant signals from the Arbiter sent to the corresponding FIFOs. Each grant is generated based on the request that comes from the LBDR of an input port. Only of the grant signals can go active at a time.
  - Xbar_sel, which is encoded as one-hot, and controls the select line of the XBAR. The XBAR is implemented as a one-hot Multiplexer (MUX). The reason for choosing one-hop is better detectability of single stuck-at faults.
  - RTS_FF_in, which stores the current value of the RTS signal (used for signaling the next router that the data on the output port is valid)
  - state_in, which is taken from the internal signal of the Arbiter, it shows the current value of the state of the arbiter (normally it is not encoded as one-hot, but for better detectability of faults, we can assume them as one-hot). State shows the arbiter has in which condition according to the grant given previously (maybe) to one of the requesting inputs.

7.4.1 Arbiter checkers

Based on the primary inputs/outputs and pseudo-inputs/outputs inferred from the Arbiter logic, the following checkers are introduced,which correspond to the properties that must always hold. Similar to the previous checkers, if the property in a checker does not hold, it becomes one, otherwise if the logic is working properly, it becomes zero:[50]

- err_Arbiter_Grants_onehot: Arbiter Grants must be one-hot.
- err_Arbiter_Xbar_sel_onehot: Arbiter select lines for Crossbar Switch must be one-hot.
- err_Arbiter_state_onehot: If we make the state variable of the arbiter (state_in) one-hot, it must follow the one-hot fashion.
- err_Arbiter_no_req_Grant: If there is no request for arbitration and the next router/NI is also not ready to receive any flits, the arbiter must not active any Grant signals in its output.

Table 7.3: Arbiter checkers detection absolute weights.

Checker name	Checker Weight (no. of detections)
err_Arbiter_state_onehot	18239
err_Arbiter_Xbar_sel_onehot	5084
err_Arbiter_Grants_onehot	1891
err_Arbiter_no_req_Grant	15

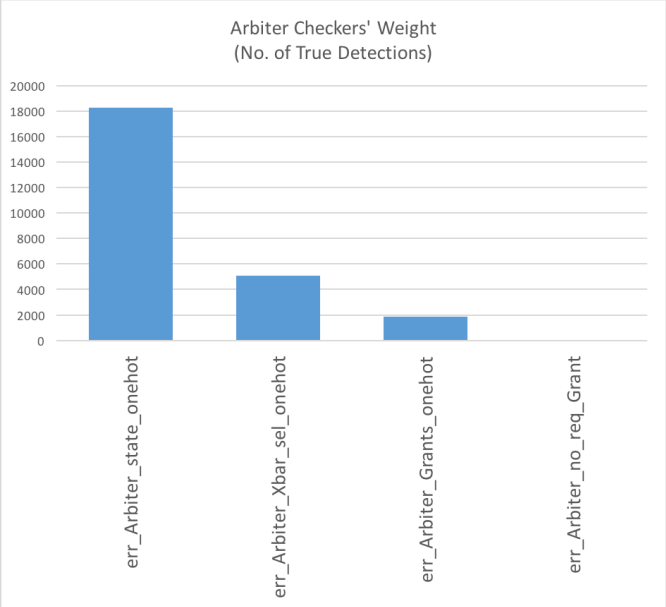

Figure 7.4: Arbiter checkers' detection absolute weights

7.5 Experiment Results

All the checkers were devised for the pseudo-combinational version of the circuit-under-test. The experiments are carried out on the pseudo-combinational circuit

using the Turbo Tester tools as described in chapter 4. First the initial set of checkers and the set of input stimuli are prepared in order to perform fault simulation to evaluate the checkers logic. The maximum exhaustive set of input stimuli presented in this work is $2^{11} = 2048$ vectors, which is reduced by the filtering scheme to valid input vectors, 320 vectors for FIFO, 768 vectors for Arbiter, and 288 for LBDR. The results of the evaluation process in terms of Absolute weights are reported in figure 7.2, table 7.1 for FIFO, in figure 7.3, table 7.2 for LBDR and figure 7.4 and table 7.3 for Arbiter. The fault coverage and checkers efficiency index obtained from the set of checkers for each unit of router is mentioned in Table 7.10. No false positives ratio were reported in output of evaluation during this experiment, so FPR metric is not considered. All the results in tables are in μm^2 based on AMS 0.18 um CMOS technology library.

Table 7.4: Area report for NoC router modules.

Module	Combinational (μm^2)	Sequential (μm^2)	Sum (μm^2)
Arbiter	601	259	860
LBDR	211	324	536
XBAR	2173	0	2173
FIFO one-hot	5105	8893	13999
Router	40515	47388	87903

The information about the combinational and sequential area of each module of NoC router is reported in Table 7.4. The table shows the maximum area is taken by FIFO followed by XBAR, Arbiter and then LBDR. The XBAR reported zero combinational area. Also the total area of router is shown.

Table 7.5: Area report for error control codes.

Module	Combinational(μm^2)	Sequential(μm^2)	Sum(μm^2)
Parity	663	0	663
Hamming encoder	1484	0	1484
Hamming decoder	7064	0	7064
CRC	299	908	1207

The area for error control codes is reported in Table 7.5. It indicates the least area for Parity, due to the fact only one redundant bit is added to the input data

stream. The area for CRC is less than Hamming. The high area of Hamming can be explained due to the presence of decoder which also correct single bit errors and detect double bit errors, which is absent in the other two Error Control Codes,Parity and CRC.

Table 7.6: Area report for pseudo-combinational circuits.

Module	Area(μm^2)
LBDR	217
Arbiter	685
FIFO Control part	5582

Table 7.7: Area report for checkers.

Module	Area(μm^2)
LBDR checkers	338
Arbiter checkers	392
FIFO Control part checkers	973

In Table 7.6, the area report for the pseudo-combinational circuits of router modules is shown. The pseudo-combinational circuits were devised as circuit-under-test for checkers evaluation. The Table 7.7 shows the results of area for checkers. The checkers have purely combinational architecture. The Arbiter checker occupies only 50% of the total area of pseudo-combinational Arbiter circuit. Similarly FIFO occupies 17.4% of psedo-combinational. It is observed that the the area of the LBDR larger that its circuit-under-test. At the same time with higher area-overhead, LBDR provides 98.98% percent of CIE and 99.7% FC, followed by FIFO with CIE of 90.901% and FC 96.157% as shown in Table 7.8.

Table 7.8: checkers efficiency index (CEI) and fault coverage(FC) results.

Module	CEI (in %)	FC (in %)
LBDR checkers	98.985	99.755
Arbiter checkers	73.795	92.284
FIFO Control part checkers	90.901	96.157

In the end, checkers efficiency index (CEI) and fault coverage(FC) are obtained which is indicated in Table 7.8. With the present set of checkers maximum CEI and FC was obtained for LBDR. The least CEI and FC is shown in case of Arbiter.

7.6 Result Analysis

The results and analyses are provided to determine the appropriate error control scheme for the case study. For the implemented error control codes area was calculated and provided in terms of two types of encoding schemes, namely hop-to-hop and end-to-end. These codes are proposed for tolerating transient faults in data-path. The end-to-end encoding scheme shows less area overhead compared to hop-to-hop. This can be explained on the fact that end-to-end encoding scheme includes ECC in network interface of source and destination router. In hop-to-hop, ECC is incorporated at all the 5 ports of the router resulting in more area-overhead. In end-to-end the area-overhead with hamming, CRC and SP is 7.11%, 2.85% and 1.11% respectively. End-to-end encoding scheme do not detect or correct errors in the router, it is suitable for short network size with short link paths [21]. Strong error control codes are required in this case. Therefore, among the presented ECC codes, hamming or CRC are best suitable for end-to-end. In hop-to-hop the area-overhead with hamming, CRC and SP is 36.61%, 11.15%, and 6.45%. In hop-to-hop simple ECC codes are required, complex codes can increase circuit complexity and power consumption [19]. Therefore, from the presented ECC methods,SP or CRC are best suitable for hop-to-hop.

Concurrent online checkers are proposed to handle both transient and permanent faults occurring during the life-time of a system. The checkers are devised for different pseudo-combinational module of NoC routers and evaluated using Turbo Tester tool. The highest CEI of 99.98% and FC of 99.75% is obtained for LBDR. The proposed checkers for FIFO provided efficiency of 90.90%. It was possible to achieve FC of 92.2% for Arbiter which had area overhead of only 50% of the circuit-under-test. The checkers provide a constant area overhead in routers.

Chapter 8

Conclusion

This dissertation propose a methodology to partition design into control and data parts, each to be protected by checkers and error-control-codes respectively. Presented methods include error control codes implementation and online checkers designs, to handle the faults in both data-path and the control part of the NoC. The concurrent checkers monitor the control part, and error detection and/or correction code checks the data-path of the NoC. The results and analyses are provided to determine the appropriate error control scheme for the case study. For the implemented error control codes area was calculated and provided in terms of two types of encoding schemes, namely hop-to-hop and end-to-end. These codes are proposed for tolerating transient faults in data-path. The end-to-end encoding scheme shows less area overhead compared to hop-to-hop. This can be explained on the fact that end-to-end encoding scheme includes ECC in network interface of source and destination router. In hop-to-hop, ECC is incorporated at all the 5 ports of the router resulting in more area-overhead. In end-to-end the area-overhead with hamming, CRC and SP is 7.11%, 2.85% and 1.11% respectively. End-to-end encoding scheme do not detect or correct errors in the router, it is suitable for short network size with short link paths [21]. Strong error control codes are required in this case. Therefore, among the presented ECC codes, hamming or CRC are best suitable for end-to-end. In hop-to-hop the area-overhead with hamming, CRC and SP is 36.61%, 11.15%, and 6.45%. In hop-to-hop simple ECC codes are required, complex codes can increase circuit complexity and power consumption [19]. Therefore, from the presented ECC methods,SP or CRC are best suitable for hop-to-hop.

Concurrent online checkers are proposed to handle both transient and permanent faults occurring during the life-time of a system. The checkers are devised for different pseudo-combinational module of NoC routers and evaluated using Turbo Tester tool. The highest CEI of 99.98% and FC of 99.75% is obtained for LBDR. The proposed checkers for FIFO provided efficiency of 90.90%. It was possible to achieve FC of 92.2% for Arbiter which had area overhead of only 50% of the circuit-under-test. The checkers provide a constant area overhead in routers.

To summarize, an optimal fault detection/correction solution is a carefully selected set of methods.The set consists of methods targeted for transient and permanent faults that appear during the life time of the system. Furthermore, the selection of these methods for the case study is influenced by design choice of NoC as well as the desired fault coverage and efficiency for the system.

For future, this work can be extended by the optimization of the checkers using a greedy heuristic approach, which select checkers with maximum absolute weights and adding them one by one to the set of accepted checkers, until the maximum CEI and FC are achieved, while at the same time making sure that the total area of the obtained checkers does not exceed the area of module under test. For better efficiency and fault coverage in FIFO, It's architecture can be changed from one-hot. For end-to-end encoding schemes, other complex ECC like BCH, RS can be implemented and area can be provided for analysis.

References

[1] LU, R., CAO A., AND KOH, C. Samba-bus: "A high performance bus architecture for system-on-chips." IEEE Trans. VLSI syst. 15, 1 (Jan. 2007), 69-79.

[2] DALLY, W., AND TOWLES, B. "Route packets, not wires: On-chip interconnection networks." In Proc. Design Automation Conference (DAC) (June 2001), pp. 684-689.

[3] C. Constantinescu, "Trends and challenges in VLSI circuit reliability," IEEE Micro, vol. 23, no. 4, pp. 14?19, Jul. /Aug. 2003.

[4] C. Grecu, A. Ivanov, R. Saleh and P. P. Pande, "NoC interconnect yield improvement using crosspoint redundancy," Proc. IEEE Intl. Symp. on Defect and Fault Tolerance in VLSI Syst. (DFT), pp. 457-465, Oct. 2006.

[5] BJERREGAARD, T., AND MAHADEVAN, S. "A survey of research and practices of network-on-chip." ACM Computing Surveys 38, 1 (Mar. 2006), 1-51.

[6] Teijo Lehtonen (2009) On Fault Tolerance Methods for Networks-on-Chip: Published PhD Thesis, University of Turku

[7] Bo Fu (2010) "Crosstalk-Aware Multiple Error Control for Reliable On-Chip Interconnects" : Published PhD Thesis, University of Rochester.

[8] ARTERIS. "A comparison of network-on-chip and busses." White Paper, 2005.

[9] Philip Koopman, "Cyclic Redundancy Code (CRC) Polynomial Selection For Embedded Networks"

[10] Qiaoyan Yu, Paul Ampadu, Dual-Layer "Adaptive Error Control for Network-on-Chip Links"

[11] "Implementing CRCs in Altera Devices", October 2005 ver.2.1

[12] Srinivasan Murali, Luca Benini, Giovanni De Micheli, Theocharis Theocharides, N. Vijaykrishnan, and Mary Jane Irwin, "Analysis of Error Recovery Schemes for Networks on Chips"

[13] L. Benini and G. De Micheli, "Networks on chips: a new SoC paradigm," IEEE Computer, vol. 35, no. 1, pp. 70-78, Jan. 2002.

[14] B. Li, L.-S. Peh, and P. Patra, "Impact of process and temperature variations on network-on-chip design exploration," in Networks-on-Chip, 2008. NoCS 2008. Second ACM/IEEE International Symposium on, 2008, pp. 117-126.

[15] Y. Ogasahara, M. Hashimoto, and T. Onoye, "Quantitative prediction of on-chip capacitive and inductive crosstalk noise and discussion on wire cross-sectional area toward inductive crosstalk free interconnects," in Computer Design, 2006. ICCD 2006. International Conference on, 2006, pp. 70-75.

[16] B. Gill, N. Seifert, and V. Zia, "Comparison of alpha-particle and neutron-induced combinational and sequential logic error rates at the 32nm technology node," in Reliability Physics Symposium, 2009 IEEE International, 2009, pp. 199-205.

[17] C. Zhao, X. Bai and S. Dey, "Evaluating transient error effects in digital nanometer circuits," IEEE Trans. Reliab., vol. 56, no. 3, pp. 381-391, Sep. 2007.

[18] A. Dixit and A. Wood, "The impact of new technology on soft error rates," in Reliability Physics Symposium (IRPS), 2011 IEEE International, 2011, pp. 5B.4.1- 5B.4.7

[19] D. Bertozzi, L. Benini, and G. De Micheli, "Error control schemes for on-chip communication links: the energy-reliability tradeoff," IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst., vol. 24, no. 6, pp. 818-831, Jun. 2005.

[20] Simon Tam, " Single Error Correction and Double Error Detection", XAPP645 (v2.2) August 9, 2006.

[21] S. Murali, T. Theocharides, N. Vijaykrishnan, M. J. Irwin, L. Benini, and G. De Micheli, "Analysis of error recovery schemes for networkson chips," IEEE Design Test Comput., vol. 22, no. 5, pp. 434-442, Sep.-Oct. 2005

[22] D. Rossi, P. Angelini, and C. Metra, "Configurable error control scheme for NoC signal integrity," in Proc. IOLTS, 2007, pp. 43-48.

[23] M. Ali, M. Welzl, S. Hessler, and S. Hellebrand, "An efficient fault tolerant mechanism to deal with permanent and transient failures in a network-on-chip," Int. J. High Perform. Syst. Arch., vol. 1, no. 2, pp. 113-123, Jan. 2007.

[24] A. Sanusi and M. A. Bayoumi, "Smart-flooding: A novel scheme for fault-tolerant NoCs," in Proc. IEEE SoC Conf., 2009, pp. 259-262.

[25] Y.Qiaoyan and A. Paul, "Dual-Layer Adaptive Error Control for Network-on-Chip Links" in Proc. IEEE TRANSACTIONS ON VERY LARGE SCALE INTEGRATION (VLSI) SYSTEMS, VOL. 20, NO. 7, JULY 2012

[26] A. D. Tipton et al, "Multiple-bit upset in 130 nm CMOS technology," IEEE Trans. Nucl. Sci., vol. 53, no. 6, pp. 3259-3264, Dec. 2006.

[27] T. Lehtonen, P. Liljeberg and J. Plosila, "Analysis of forward error correction methods for nanoscale networks-on-chip," Proc. 2nd Intl. Conf. on Nano-Networks (Nano-Net), pp. 1-5, Sep. 2007.

[28] W. J. Townsend, J. A. Abraham, and P. K. Lala, "On-line error detecting constant delay adder," in 9th IEEE International On-Line Testing Symposium, IOLTS ?03, Kos Island, Greece, Jul. 2003, pp. 17-22.

[29] A. Ganguly, P. P. Pande, B. Belzer, and C Grecu, "Design of low power and reliable networks on chip through joint crosstalk avoidance and multiple error correction coding," Journal of Electronic Testing: Theory and Appl. (JETTA), Special Issue on Defect and Fault Tolerance, pp. 67-81, Jun. 2008.

[30] A. Ejlali, B.M. Al-Hashimi, P. Rosinger, and S.G. Miremadi, "Joint consideration of faulttolerance, energy-efficiency and performance in on-chip networks," Proc. of Design, Automation and Test in Europe (DATE), pp. 1-6, Apr. 2007.

[31] Suboh A. Suboh, Vikram K. Narayana, Mohamed Bakhouya, and Tarek El-Ghazawi, "A Scalability Study of Interconnect Architectures for System-on-Chip", IEEE International Conference on High Performance Computing and Simulation (HPCS), 2012.

[32] R. Ubar, J. Raik, A. Jutman, and M. Jenihhin, ?Diagnostic Modeling of Digital Systems with Multi-Level Decision Diagrams,?

[33] Suboh .A, Blahut. "Algebraic Codes for Data Transmission." Cambridge University Press, 2003.

[34] Kumar S. et. al. "A network-on-chip architecture and design methodology". In IEEE Computer Society Annual Symposium on VLSI, pages 117-124, Pittsburg, April 2002.

[35] P. K. Samudrala, J. Ramos, and S. Katkoori, "Selective triple modular redundancy (stmr) based single-event upset (seu) tolerant synthesis for fpgas," Nuclear Science, IEEE Transactions on, vol. 51, no. 5, pp. 2957-2969, 2004.

[36] R. Sharma and K. K. Saluja, "An implementation and analysis of a concurrent built-in self-test technique," in Fault-Tolerant Computing, 1988. FTCS-18, Digest of Papers., Eighteenth International Symposium on, pp. 164-169, IEEE, 1988.

[37] P. Drineas and Y. Makris, "Concurrent fault detection in random combinational logic," in Quality Electronic Design, 2003. Proceedings. Fourth International Symposium on, pp. 425-430, IEEE, 2003.

[38] K. Nepal, N. Alves, J. Dworak, and R. I. Bahar, "Using implications for online error detection," in Test Conference, 2008. ITC 2008. IEEE International, pp. 1-10, IEEE, 2008.

[39] M. Boule, J.-S. Chenard, and Z. Zilic, "Assertion checkers in verification, silicon debug and in-field diagnosis," in Quality Electronic Design, 2007. ISQED?07. 8th International Symposium on, pp. 613?620, IEEE, 2007.

[40] P.Saltarelli, B.Niazmand, J.Raik, R.Hariharan, G.Jervan and T.Hollstein,"A Framework for Comprehensive Automated Evaluation of Concurrent Online Checkers", Digital System Design (DSD), 2015 Euromicro Conference on, pp. 288 - 292, IEEE, 2015.

[41] J. Flich and J. Duato, ?Logic-based distributed routing for nocs,? Computer Architecture Letters, vol. 7, no. 1, pp. 13?16, 2008.

[42] Q. Yu, J. Cano, J. Flich, and P. Ampadu, "Transient and permanent error control for high-end multiprocessor systems-on-chip," in Networks on Chip (NoCS), 2012 Sixth IEEE/ACM International Symposium on, pp. 169-176, IEEE, 2012.

[43] J. Flich and J. Duato, "Logic-based distributed routing for nocs," Computer Architecture Letters, vol. 7, no. 1, pp. 13-16, 2008.

[44] A. Alaghi, N. Karimi, M. Sedghi, and Z. Navabi, "Online noc switch fault detection and diagnosis using a high level fault model," in Defect and Fault-Tolerance in VLSI Systems, 2007. DFT-7. 22nd IEEE International Symposium on, pp. 21-22, IEEE, 2007.

[45] R. Parikh and V. Bertacco, "Formally enhanced runtime verification to ensure noc functional correctness," in Proceedings of the 44th Annual IEEE/ACM International Symposium on Microarchitecture, pp. 410-419, ACM, 2011.

[46] A. Prodromou, A. Panteli, C. Nicopoulos, and Y. Sazeides, "Nocalert: An on-line and real-time fault detection mechanism for network-on-chip architectures," in Microarchitecture (MICRO), 2012 45th Annual IEEE/ACM International Symposium on, pp. 60-71, IEEE, 2012.

[47] A. Dalirsani, M. Kochte, H.-J.Wunderlich, et al., "Area-efficient synthesis of fault-secure noc switches," in On-Line Testing Symposium (IOLTS), 2014 IEEE 20th International, pp. 13-18, IEEE, 2014.

[48] B. W. Johnson, "Design and Analysis of Fault-Tolerant Digital Systems." Boston, MA: Addison-Wesley, 1989

[49] Siavoosh Payandeh Azad and Behrad Niazmand, (March 30, 2016) Network-on-Chip (NoC) Router Architecture(Online) Available: *https* : *//github.com/siavooshpayandehazad/NoC*$_F$*PGA*$_E$*mulator/wiki/Router*

[50] Siavoosh Payandeh Azad and Behrad Niazmand, (March 30, 2016) Network-on-Chip (NoC) Router Architecture(Online) Available:*https* : *//github.com/Project* − *Bonfire/EHA/wiki/Checkers*

[51] Turbo Tester Group. (2008, Jan.8) Turbo Tester Reference Manual, [Online]. Available : *http* : *//www.pld.ttu.ee/tt*

[52] P. K. Lala, "Self-Checking and Fault-Tolerant Digital Design." San Fransisco, CA: Morgan Kaufmann Publishers, 2001.

[53] M. Aarna, E. Ivask, A. Jutman, E. Orasson, J. Raik, R. Ubar, V. Vislogubov, and H. D. Wuttke, "Turbo Tester - Diagnostic Package for Research and Training," Scientific-Technical Journal, Radioelectronics and Informatics, vol. 3, pp. 69?73, 2003.

[54] Rong Pan, Nur A. Touba, and Edward J. McCluskey. 1993. The Effect of Fault Dropping on Fault Simulation Time. Technical Report. Stanford University, Stanford, CA, USA.

[55] A. Jutman, A. Peder, J. Raik, M. Tombak, and R. Ubar, "Structurally synthesized binary decision diagrams," in 6th International Workshop on Boolean Problems, pp. 271-278, 2004.

Appendix A

Simulation Results for SP, Hamming and CRC

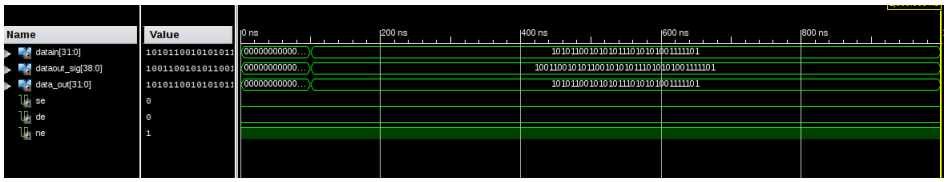

Figure A.1: Simulation report for Hamming encoder and decoder.

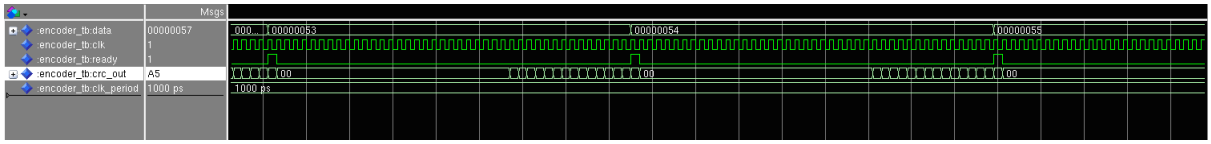

Figure A.2: Simulation Report for CRC encoder.

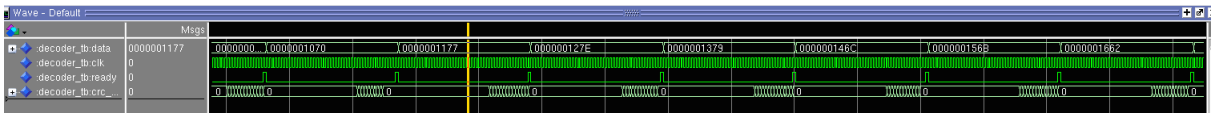

Figure A.3: Simulation Report for CRC decoder.

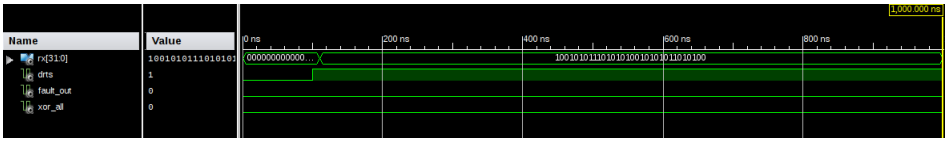

Figure A.4: Simulation Report for Single Parity.

Appendix B

Definitions

1. Fan-out: Fan-out of a logic gate output is the number of gate inputs it can feed or connect to. In this case logic gates are connected to form more complex circuits. It is a case, where one output is connected to several inputs of the logic gates.

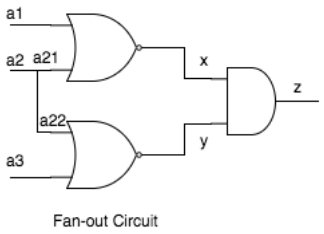

Figure B.1: Concept of Fan-out

2. Fan-out Free Regions : The circuits without fan-out points are the fan-out free circuits/regions.

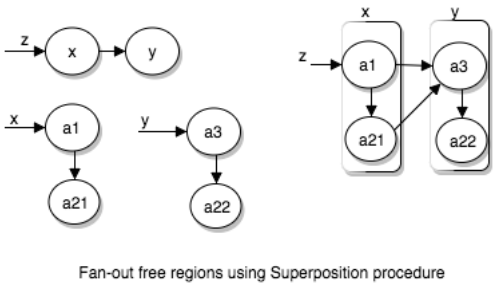

Figure B.2: Fan-out Free Regions Constructed using Superposition Algorithm

Appendix C

Evaluation Results for LBDR

```
.VECTORS 288


.PATTERNS

0000010010000
0000010010001
0000010010010
0000010010011
0000010010100
0000010010101
0000010010110
0000010010111
0000010011000
```

Figure C.1: Input test pattern

```
4
37
38
39
40
```

Figure C.2: Checkers output

```
STAT#    103 Nods,  46 Vars,  33 Grps,  13 Inps,  0 Cons,  9 Outs

MODE#   STRUCTURAL


VAR#   0:  (i_______)  "empty"
VAR#   1:  (i_______)  "Req_N_FF"
VAR#   2:  (i_______)  "Req_E_FF"
VAR#   3:  (i_______)  "Req_W_FF"
VAR#   4:  (i_______)  "Req_S_FF"
VAR#   5:  (i_______)  "Req_L_FF"
VAR#   6:  (i_______)  "flit_type[2]"
VAR#   7:  (i_______)  "flit_type[1]"
VAR#   8:  (i_______)  "flit_type[0]"
VAR#   9:  (i_______)  "dst_addr[3]"
VAR#   10:  (i_______)  "dst_addr[2]"
VAR#   11:  (i_______)  "dst_addr[1]"
VAR#   12:  (i_______)  "dst_addr[0]"

VAR#   13:  (________)  "empty"
GRP#    0: BEG =   0, LEN =   1 -----
   0   0:  (____)       (      0      0)  V = 0     "empty"

VAR#   14:  (________)  "flit_type[2]"
GRP#    1: BEG =   1, LEN =   1 -----
   1   0:  (____)       (      0      0)  V = 6     "flit_type[2]"

VAR#   15:  (________)  "flit_type[1]"
GRP#    2: BEG =   2, LEN =   1 -----
```

Figure C.3: SSBDD representation file