Winter 2015

Mitigation of Hardware Trojan Attacks on Networks-on-Chip

Jonathan Frey
University of New Hampshire, Durham

MITIGATION OF HARDWARE TROJAN ATTACKS ON NETWORKS-ON-CHIP

BY

Jonathan Frey
B.S., EE, University of New Hampshire, 2014

THESIS

Submitted to the University of New Hampshire
in Partial Fulfillment of
the Requirements for the Degree of

Master of Science
in
Electrical Engineering

December, 2015

This thesis has been examined and approved in partial fulfillment of the requirements for the degree of Master of Science in Electrical Engineering by:

Thesis Director, Qiaoyan Yu, Ph.D.
Assistant Professor
Department of Electrical & Computer Engineering

Michael J. Carter, Ph.D.
Associate Professor
Department of Electrical & Computer Engineering

W. Thomas Miller III, Ph.D.
Professor
Department of Electrical & Computer Engineering

On December 7, 2015

Original approval signatures are on file with the University of New Hampshire Graduate School.

ACKNOWLEDGEMENTS

First and foremost, I would like to thank my advisor, Dr. Qiaoyan Yu, who encouraged me to do my master's degree. Your guidance, friendship, and support have inspired me to strive for excellence. I would also like to thank Dr. Michael Carter and Dr. Thomas Miller for serving on my thesis committee, and providing their insights. I would like to thank the rest of the Electrical and Computer Engineering faculty for teaching me over the past 5 ½ years through both my bachelor's and master's degrees.

I would also like to thank my colleagues at the University of New Hampshire Reliable VLSI Systems Lab: Raashid Ansari, Drew Stock, Jaya Dofe, Hoda Pahlevanzadeh, and Patrick Nsengiyumva. They have collaborated with me on several projects that I worked on at UNH – their contributions have been invaluable. Many other friends, namely William Nitsch, Michael Joseph, Nicholas Frederico, and Rory O'Brien, have helped keep me on track over the last few years.

Finally, I would like to thank my family, Hans, Edna, and Nicole Frey, and my loving girlfriend of 5 years, Vivian Pham. All of you have endlessly supported my aspirations while also helping me grow personally.

TABLE OF CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ACRONYMS

HT	Hardware Trojan
IP	Intellectual Property
NoC	Network-on-Chip
NI	Network Interface
HDL	Hardware Description Language
FPGA	Field Programmable Gate Array
AES	Advanced Encryption Standard
Flit	Flow Unit
DoS	Denial of Service
PUF	Physical Unclonable Function
IC	Integrated Circuit
RTL	Register Transfer Level
SRC	Semiconductor Research Corporation
VLSI	Very-Large-Scale-Integration
ATPG	Automatic Test Pattern Generation
DPU	Data Protection Unit
FBR	Fault Bypass Rate

FIFO	First-In-First-Out
LUT	Look-Up-Table
SoC	System-on-Chip
ECC	Error Control Coding
MSB	Most Significant Bit
MUX	Multiplexer
RNG	Random Number Generator
CMOS	Complementary-Metal-Oxide-Semiconductor
IoT	Internet of Things
CPA	Correlation Power Analysis
NACK	Negative-Acknowledgement

ABSTRACT

MITIGATION OF HARDWARE TROJAN ATTACKS ON NETWORKS-ON-CHIP

by

Jonathan Frey

University of New Hampshire, December, 2015

The Integrated Circuit (IC) design flow follows a global business model. A global business means that the processes in the IC design flow could be outsourced, and consequently security threats have been introduced. Security threats on hardware include side channel analysis, reverse engineering, information leakage, counterfeit chips, and hardware Trojans (HTs).This work mainly focuses on HT attacks, which execute a malicious operation on the system when a trigger condition is met. Networks-on-Chip (NoCs) are a popular communications infrastructure for many-core systems, which have proved to be a more scalable option over the traditional bus interface. However, the high scalability and modularity provided by NoCs have introduced new vulnerabilities in the design, leading to hardware Trojans capable of causing several Denial of Service (DoS) attacks on the network.

A 4x4 Mesh-topology NoC with a more robust router microarchitecture is presented with several innovations relative to the baseline. A collaborative dynamic permutation and flow unit (flit) integrity check method is proposed to thwart an attacker from maliciously modifying the flit content in the routers of a NoC. Our method complements other HT detection approaches for the NoC network interfaces. Moreover, we exploit the Physical Unclonable Function (PUF) structure and the traffic routing history to generate a unique key vector for each router to select one of the multiple permutation configurations. Simulation and Field Programmable Gate Array (FPGA) results are compared between the proposed NoC microarchitecture and four other existing solutions found in literature, and it was shown that the proposed method outperforms all of the existing security methods.

Chapter 1. Introduction

1.1. The Security Issue in Hardware

The world is moving toward one that is more electronically driven, where a greater number of people are using computers and other electronics every day. The shift towards a world with more electronics leads to a greater demand for the production of the components that make up these devices. Most electronic devices use integrated circuits (ICs), which are single chip packages consisting of a collection of smaller circuits. ICs are used in practically every electronic device today, ranging from commercial applications such as cell phones and computers, to more critical applications like banking and military systems.

To address the supply demand of ICs, the semiconductor industry and has started to follow a globalized business model for the IC design flow. The IC design flow is comprised of several stages of design, fabrication, assembly, and test. The outsourcing of these processes has begun as a result of the global business model. Figure 1.1 introduces the IC design flow, and shows outlined vulnerabilities that are a result of outsourcing. The outsourcing of processes has led to an increase in hardware security threats as it hands over the design to 3rd party [1-6]. This limits the trust one can have in systems where the internal design was outsourced. Trusted hardware is especially needed for more critical applications where a single error may have a greater impact, such as military, government, and banking systems. Hardware security has therefore gained more attention recently.

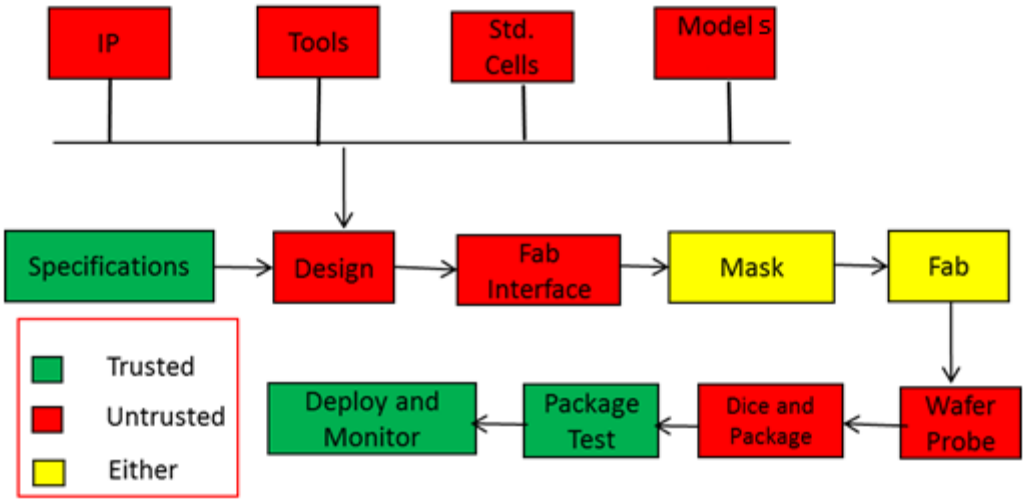


Figure 1.1. Outlined Vulnerabilities of IC Design Flow following Globalized Business Model.

1.2. Different Hardware Attacks

Hardware attacks pose an issue to the security of systems. Attacks aimed at a system's hardware can cause many adverse effects on the system. Internal memory values may be changed, secret keys can be leaked, or the circuit may degrade or even fail completely. Hardware attacks are different than software attacks, as the latter can always be removed. Software attacks that change the actual software may be reversed by overwriting the software with the original software again. However, this method cannot be applied to when a hardware attack on a system alters the actual hardware. Hardware attacks may linger with the system, or completely change the functionality of the system. Different hardware threats include fault injection, IC counterfeiting, 3rd party intellectual property (IP) piracy, and hardware Trojans (HTs).

1.2.1. Hardware Trojans

Hardware Trojans, which are malicious hardware modifications on the original chip, pose as a serious hardware security threat. This particular security issue arose directly as a result of the IC industry following a global business model. Figure 1.2 shows an understanding of the global business model, and gives an idea of how many countries may be involved in the design of a particular IC. HTs can be embedded during any stage of the IC design flow, and aim to deliver an attack on the system that can take on a variety of forms. HTs can be designed to deliver attacks such as Denial of Service, information leakage, data manipulation, and system degradation [7-11]. The malicious hardware is implemented in such a way to be hard to detect, either through being small enough to consume a negligible amount of power compared to the whole system, or by having a very small trigger probability. HTs implemented in Intellectual Property cores designed by third party vendors can introduce chip malfunctions and information leakage attacks on the system they are put into [11]. During the design stage, HTs can be embedded at the Register Transfer Level (RTL) in the design's Hardware Description Language (HDL), or straight into the gate level netlist, leading to logical attacks on the system [9]. At the fabrication stage, design layouts can be modified to include a HT, changing internal circuit characteristics such as delay [12].

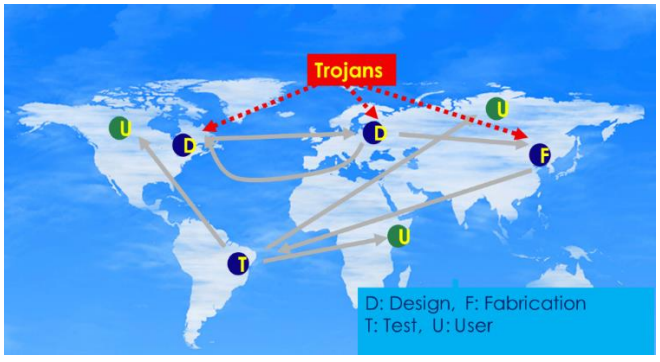


Figure 1.2. Global Business Model of IC Design Flow.

HT designs are becoming more complex and hard to detect, and have become capable of bypassing rigorous post-silicon tests. Due to the nature of HTs being triggered by a rare event, it is extremely difficult to detect HTs with conventional testing approaches [12]. HTs can activate through various types of trigger circuits, and deliver the respective attack via a payload circuit. HT trigger circuits are usually modeled as sequential [5] or combinational logic circuits [3] that activate after a specific condition is met. An example of how a HT with a sequential logic trigger circuit would become active is if the trigger circuit required a node to have a specific logic value X number of times. An example of how a combinational logic trigger circuit could activate would be if its trigger condition is to have X number of different nodes have specific logic values at the same time. There exist many hardware Trojan detection methods. Offline detection methods typically try to activate the trigger circuit to detect the existence of HTs. The switching probability of the trigger circuit is raised at the test time to increase the visibility of HTs [14, 15]. However, due to the hard to detect nature of HTs, trojaned ICs may still bypass offline detection methods, leading to residual HTs in designs. This is especially true for larger designs such as on-chip networks where complete coverage at test time is not practical.

1.2.2. Fault Attacks

Driven by malicious intent, attackers are impelled to extract the cipher key and thus compromise the cryptosystem through fault attacks. Fault attacks (FAs) compromise the cipher implementation and produce faulty ciphertexts for cryptanalysts to retrieve the secret key. FA can be executed through deliberate injection of faults into cryptographic devices by means of white light, laser beam, voltage/clock glitch, and temperature control. Existing fault-detection methods can effectively detect random faults in the cipher implementation, but yield a high fault bypass rate (FBR) under fault attacks.

1.2.3. Counterfeit ICs

Counterfeit ICs are used or lower-grade chips that are repackaged and sold as new or higher-grade chips. This security issue arose as a result of customers desiring ICs at cheaper prices, and the ramification is that systems become less reliable. The inferior chips may cause serious problems within the system where they are embedded. Typical problems that may arise from counterfeit ICs are not meeting timing constraints, or even total chip failure. If implemented in a critical electronic system, such as a military and or banking system, the likelihood that system will fail becomes greater.

1.2.4. Third Party IP Piracy

A third party foundry may produce an excessive amount of chips, corresponding to a particular IC design, than what was planned. The extra chips are then sold by the foundry over the gray market. This is done without the prior permission from the chip designer.

1.3. Thesis Contributions

1.3.1. Collaborative Dynamic Permutation and Flit Integrity Checking for NoCs

A new novel hardware Trojan mitigation method for Networks-on-Chip that works at run-time was proposed. The proposed method was compared against four other existing methods proposed in other IEEE conference and journal papers. The baseline design was considered against our design, which was [34, 35], and the other three methods were R-Term [38], NI-Term [36], and R-AddrFilter [45].

We propose a collaborative dynamic permutation and flit integrity check method to thwart an attacker from successfully modifying the flit content, especially the critical flit bits, during the flit transmission over the NoC routers. The proposed gate-level method provides multiple

permutation configurations in each router, rather than in NIs. Our work complements the existing methods [32-35, 39]. As the permutation configuration changes over time, the success rate of a HT insertion is significantly reduced even if the attacker knows the countermeasure.

To dynamically change the permutation configuration, we propose a PUF-based key vector generation method. Physical Unclonable Functions (PUFs) are designs that are mathematically the same, but differ in practice due to process variations. The proposed PUF structure exploits process variations and the random content from the router's round-robin register to generate a dynamic key vector for each flit permutation unit. As the process variations and the traffic conditions in each router are unique, key vectors for dynamic permutation units in different routers are different. Thus, the attacker is unlikely to trigger all of the HTs in the different routers, simultaneously. Consequently, a trojaned packet cannot be successfully transferred to the destination to complete the malicious task.

To the best of our knowledge, this is the first work that provides experimental results from the simulations of the gate-level NoC implementation. Quantitative throughput, link availability, traffic hotspot migration, bandwidth depletion, FPGA-based hardware overhead, and average packet latency of five methods are extensively compared. In addition to malicious destination addresses, this work further considers the loss of flit type information, which is induced by the inserted HTs.

1.4. Thesis Organization

The following chapters of the thesis are organized as described below:

Chapter 2 introduces background information on Hardware Security, Hardware Trojans and general IC security countermeasures, NoC security and countermeasures, and crypto-attacks and fault detection.

Chapter 3 introduces the basics of NoCs, as well as the building of the baseline NoC in the Verilog Hardware Description Language (HDL).

Chapter 4 introduces more on the proposed collaborative HT mitigation method and the Verilog coding of it.

Chapter 5 introduces the testing schemes used for the proposed collaborative HT mitigation method, as well as a comparison of results from the proposed method and the other existing solutions.

Chapter 6 implements the proposed method from Chapter 4 into an AES module, and it is shown whether the design can protect against crypto-attacks with a greater or worse success rate.

Chapter 7 discusses the main thesis contribution, the other relevant contributions, and finally the thesis is summarized.

Chapter 2. Background

2.1. Hardware Security

Unwanted changes and malicious additions to electronic circuit designs have become results of companies looking for cheaper ways to sell their high quality product, mainly due to companies looking for the cheapest foundries that will manufacture their product. These inclusions can be designed to be undetected and activate at almost random times or based on extremely rare events occurring in a particular order. The results of these malicious inclusions can be of a wide range, including resource hogging, system malfunctions, system degradation, data deletion, and even secret data leakage.

The five main stages of the Integrated Circuit (IC) design flow are specifications, design, fabrication, test, and package/assembly. Figure 2.1 shows the IC design flow and points out the stages that are considered trusted and those that are considered untrusted. The design stage is a viable option for insertion of malicious hardware, but this assumes an insider within the company designing the system. The fabrication stage is the most likely target to insert malicious hardware as the system's design is already out of the hands of the one who designed it. These problematic additions to hardware during the fabrication process have become known as Hardware Trojans. Hardware Trojans can be inserted into each of the different abstraction levels of designs. The different abstractions include, the system, register transfer, gate, transistor, and physical levels. Figure 2.2 shows a basis of what must be done at a particular abstraction level to insert a HT. At the system level, an extra system with HTs can be inserted between physical systems that are already designed and built. At the system level, HTs can cause information leakage and manipulation of data flowing through the extra system. HTs can be inserted into the register transfer level by the modification of logic. This can include an extra parameter in a conditional statement.

HTs inserted at the gate level require the addition of extra logic gates. An example of a gate-level HT can be seen in Figure 2.3, which follows in the next section. Transistor abstraction level HTs are achieved by modifying parameters of the switches/transistors. This includes the modification (thinning or widening) of the transistors channels, or width/length ratios. HTs at the transistor level can cause problems such as an imbalance in current and a change in delay characteristics. Delay characteristics in digital circuits are important as setup and hold time constraints can be violated easily. The modification of a design's layout, or physical abstraction, can cause HTs at this abstraction level. The modification of wire lengths/widths can cause several problems at the layout, including the widening of clock networks to change delay characteristics and combination/separation of physical connections.

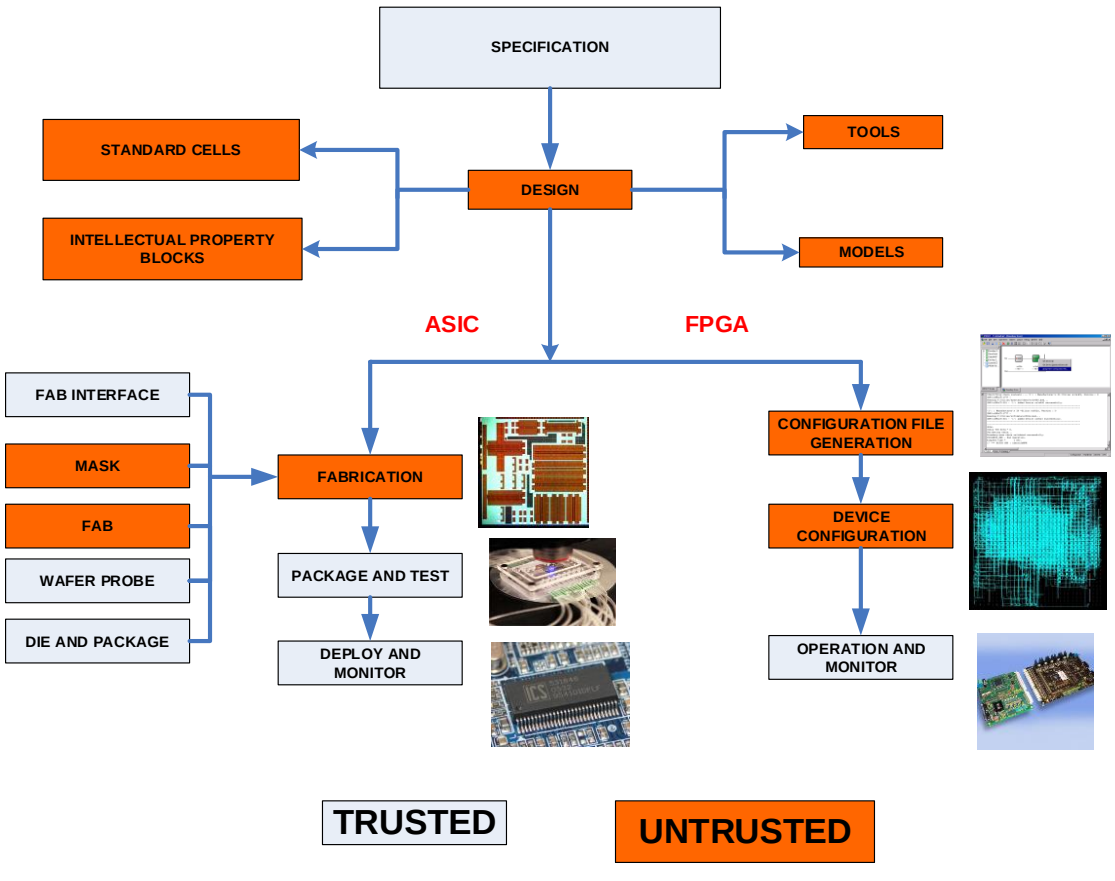


Figure 2.1. Integrated Circuit design flow with trusted and untrusted stages.

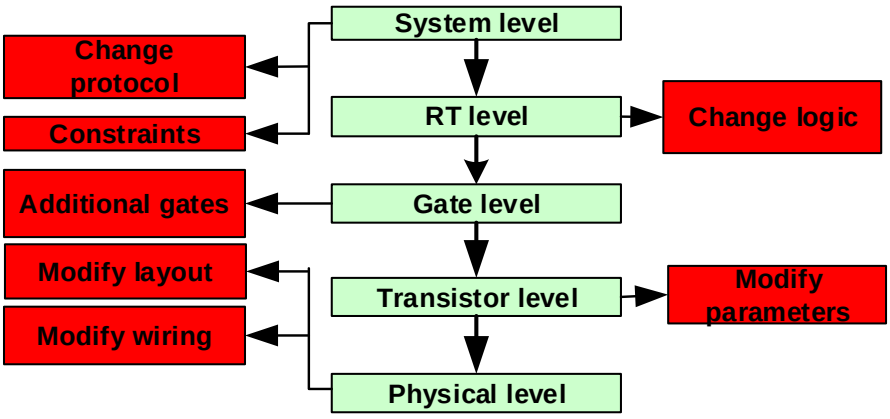


Figure 2.2. Insertion of HTs at different abstraction levels.

Hardware Trojans have become a serious issue in the ICs world today due to the wide usage of ICs in banking and military systems. If a banking system was to leak passwords or even fail, it is unknown what would happen to the currency supposedly safe to store there, meanwhile if a military system were to fail, unthinkable disasters could occur, leaving a level of damage done one would think is improbable.

2.2. Hardware Trojan Countermeasures

A lot of the previous work done on HT detection methods is done at the test-time. Offline detection methods presented in [14-16] aim to identify HTs by increasing the probability that the trigger circuit activates. The results of the offline tests are compared with a "golden model" to detect any abnormalities in the design. Figure 2.3 shows a basic understanding of a HT circuit model. If the HT model shown in Figure 2.3 is applied to a crypto-system, the end result may be look something like Figure 2.4. The HT payload in Figure 2.4 shows how a simple circuit can bypass a very complex system operation.

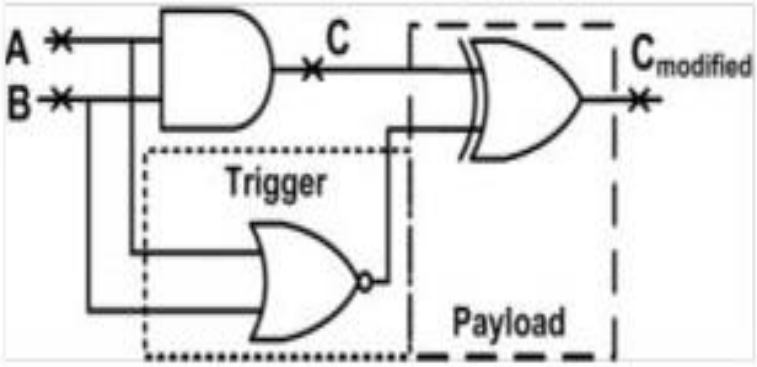


Figure 2.3. Basic Hardware Trojan circuit model.

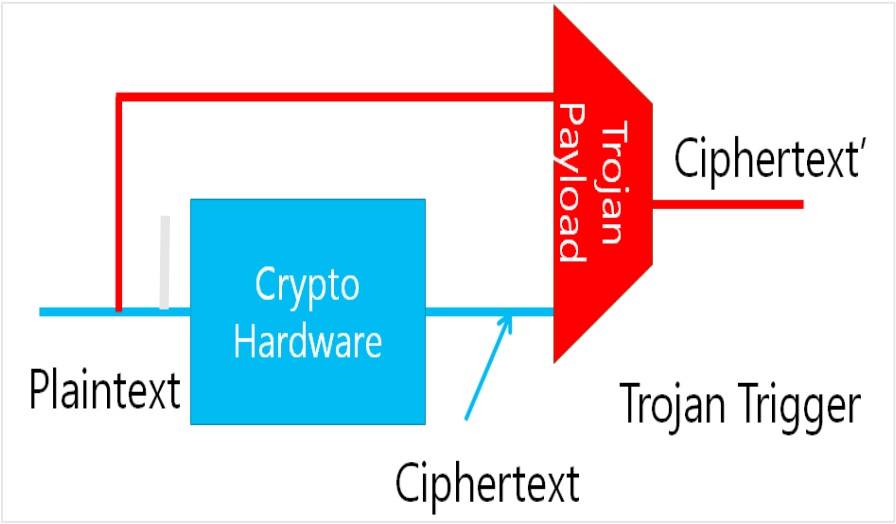


Figure 2.4. Hardware Trojan bypassing an example system.

Side Channel Analysis [17-21] is an effective offline method for the detection of HTs. This process works through the examination of side channel signals. Some side channel signals include electromagnetic radiation, delay characteristics, and power characteristics. The extracted side channel data is then compared to what it should look like via a golden model. As transistor size is constantly decreasing, it is becoming easier for side channel signals generated by HTs, such as delay

and power consumption, to be masked within the process variations of the system. This means that it will be harder to detect HTs during offline testing methods with the analysis of side-channel signals. The idea of a golden model is also unrealistic as process variations still exist and HTs can easily be masked within the deviation.

I/O ports and bit widths are also increasing in embedded systems. This leads to an increased testing time for system verification, making complete coverage of the system costly with respect to money and time. A typical attack would not include HTs in every IC, but to inject them into a random amount of ICs, to further reduce the detection probability. HTs that are difficult to trigger, or depend on large amount of combinational logical input conditions to be satisfied, are especially hard to detect with logical I/O testing. Automatic Test Pattern Generation (ATPG) [14, 15] is an effective method in detecting HTs that trigger with combinational logic conditions. ATPG is essentially an I/O test where, given certain input patterns, the output is logically compared to what it should look like given the respective input. ATPG becomes less effective when the logical trigger condition becomes more complex and depends on an increased amount of nodes.

IP core authentication can be performed by various techniques presented in [11, 27-29]. Recently, the Semiconductor Research Corporation (SRC) [27] concluded that one of the most challenging hardware security areas is the lack of HT detection methods at and before the Register Transfer Level (RTL). This requirement also applies to the Network-on-Chip (NoC) community. HTs maliciously inserted in a NoC can lead to information leakage, unauthorized memory accesses and denial of service (DoS) attacks, e.g. incorrect path routing, deadlock, and livelock [2, 10].

At run-time, the time and economic cost for replacing components is impracticable. Instead, run-time solutions can provide a last level of defense for systems to mitigate the effects of HTs. Run-time approaches focus more on the effects of the HT payload that delivers the attack, as the

trigger circuit has already bypassed offline methods. In this work, HTs that are placed in the design netlist are specifically addressed, as advocated by SRC in [27], and a run-time solution with low performance overhead is provided for NoCs.

2.3. Fault Countermeasures

Fault-detection methods are typically used to ensure the integrity of the cryptographic process. When fault attacks become a security concern, conventional fault-detection methods are first chosen [48]. In the evaluation stage, the existing works [47, 55-57, 60, 62-65] only consider the scenario that faults randomly happen in the cipher implementation. As fault attacks are intentional, we need to rigorously consider the scenario that the attacker may inject faults that have potential to bypass the fault-detection mechanism. For instance, the fault-detection methods shown in Fig. 2.4 cannot detect the faults injected in the input, i.e. intermediate state registers. This is because both data paths receive the same faulty input, and no difference will be observed at the comparison stage. As a result, a faulty ciphertext can bypass the fault detection.

Moreover, each method has its own limitation. In the Double Modular Redundancy (DMR) method in Fig. 2.5(a), if the fault attack is performed at the same time and location on the two F(·) functions, the DMR detection will fail. This kind of fault attack is feasible, as the two F(·) copies are identical. For a lightweight cipher like SIMON, the reverse function for fault detection in Fig. 2.5(b) is likely designed with a symmetric architecture. The reverse function may fail if the recovered intermediate state i is corrected by a symmetric fault before the comparison for fault detection. In the work [55, 56], the intermediate state is permuted before F(·), as shown in Fig. 2.5(c). As pointed out in [56], permutation can possibly fail because the permuted input bits may share the same logic value. Similar to DMR, the masking scheme shown in Fig. 2.5(d) cannot detect

faults injected at the same time and location in the two F(·) functions. This problem cannot be resolved even if the order of masking is extended [13].

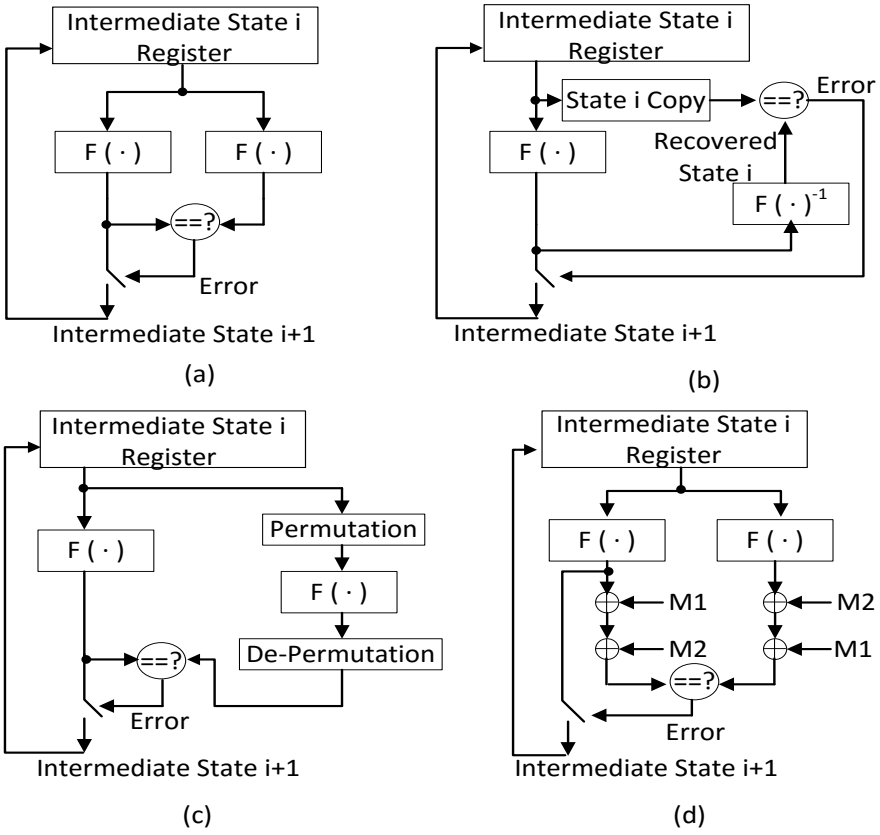


Figure 2.5. Typical fault-detection methods. (a) DMR [3], (b) reverse function [11], (c) permutation [9, 10], (d) masking. F(·) represents one round function for the cryptographic algorithm.

2.4. Introduction to the Network-on-Chip

As the number of transistors integrated on a single die grows, increasingly IP cores are connected with the Network-on-Chip (NoC) infrastructure. A basic understanding of a Network-on-Chip can be seen in Figure 2.6. NoCs were designed as an effective communications infrastructure for many-core systems. IP cores, such as processing and memory elements, need to

communicate with each other to execute instructions. Mesh NoCs, like the one seen in Figure 2.6, communicate on a packet basis. Each packet is organized into several flow units, or flits. Each flit contains a header to let the NoC know if it is the start, middle, or end of a packet. The header content of a flit contains other important information such as which IP core it was sent from, or the source address, and which IP core the data is being sent to, or the destination address. Each IP core is connected to a respective Network Interface (NI). The IP core's NI is responsible for packetizing the data that the IP core is sending, and depacketizing the data being received by the IP core. After packetization, the data is sent over the network, where it is moved by a series of routers, one at a time, toward its destination. Each NI is connected to one router, and the routers are connected in a mesh format. Routers interpret the header content of flits, and this is how they decide from what port the router must send the data out of. Each router has five input and five output ports corresponding to different port directions. A "local" port direction is assigned to the data being sent to and from the respective IP core. The other four port directions are cardinal north, south, east, and west. Source and destination addresses are assigned based on the router ID numbers. Each router in the network has its own ID number, which corresponds to where in the network it is. The routers' IDs are assigned based on their X and Y coordinates in the network.

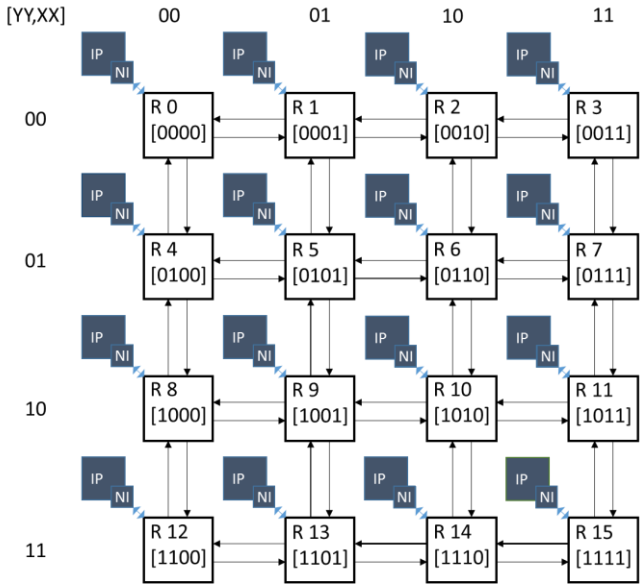


Figure 2.6. Basic understanding of 4x4 mesh-topology NoC.

Figure 2.6 shows the traditional router ID naming standard. For the 4x4 mesh NoC shown in Figure 2.6, there are a total of 16 routers, meaning that with Eq. 2.1, the ID must take up 4 bits. Two of the ID bits are assigned to the X direction, and the other two bits are assigned to the Y direction. The ID bits are formed for a 4x4 mesh NoC in the format {YY, XX}, which happens to agree in binary as to what the router's number is. The last part of the header content is the flit type.

$$bits_{ID} = \lceil \log_2(\#\ of\ Routers) \rceil \qquad \text{Equation 2.1}$$

Depending on the size of the data being transmitted, the packet size may vary, which translates to more flits per packet. There are four types of flits; empty, payload, header, tail. Flits are given two bits to determine their type; 00 for an empty flit, 01 for a header flit, 10 for a tail flit, and 11 for a payload flit. Packets are organized by starting with a header flit to denote the start of a

packet, then several payload flits depending on the size of the data being sent, and lastly a tail flit to denote the packet being transmitted has ended. Figure 2.7 shows a basic understanding of the NoC packet format. Figure 2.7 also defines the critical flit fields which are the destination address, source address, and the flit type fields.

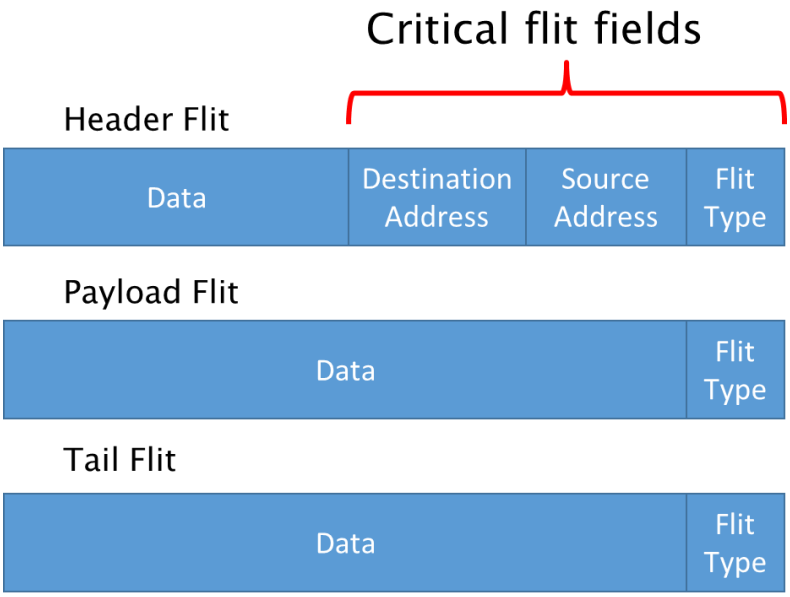


Figure 2.7. NoC packet format with critical flit fields labeled.

The path that flits take along the network depends on the routing method used. An example of a routing method is XY Routing, where the packet will move from its source X address to its destination X address first, then will move from its source Y address to its destination Y address. YX Routing is another approach, but moves the packet in the Y direction first, then the X direction. For simplification, it is further assumed that simulations are run with the XY Routing approach.

Flits received at a router port will be stored in that port's First-In-First-Out (FIFO) registers. Once the data leaves the FIFO, it enters the Full Crossbar, which interprets the header content of

the flit and determines flit type and the appropriate destination port from which to send this data out. Lastly, the data is stored in the output FIFO of the appropriate port to be sent out.

2.5. Countermeasures for On-chip Networks

Networks on Chip (NoCs) were introduced to be a solution for the scalability problem that Systems on Chip (SoCs) had with communications. Although an effective solution, these new communication structures came with their own problems, not least of which is security. Issues such as Denial of Service (DoS), illegal read accesses leading to extracted secret keys, and illegal write accesses leading to buffer overflow and overwritten data, have been introduced as security problems with NoCs. There exist general methods for the protection of Networks on Chip, just as is the same for Hardware Trojans, but no method exists to ensure complete security for the system. Some general methods for protection include the use of request verification through the use of look up tables (LUTs), exchanging secret information through the use of Virtual Channels, and dividing the network into secure IP core and non-secure IP core zones to allow only secure information to be sent to and from the secure zone, which is considered safe from malicious inclusions. Figure 2.8 shows two cases where HTs inserted in NoCs can cause hardware attacks.

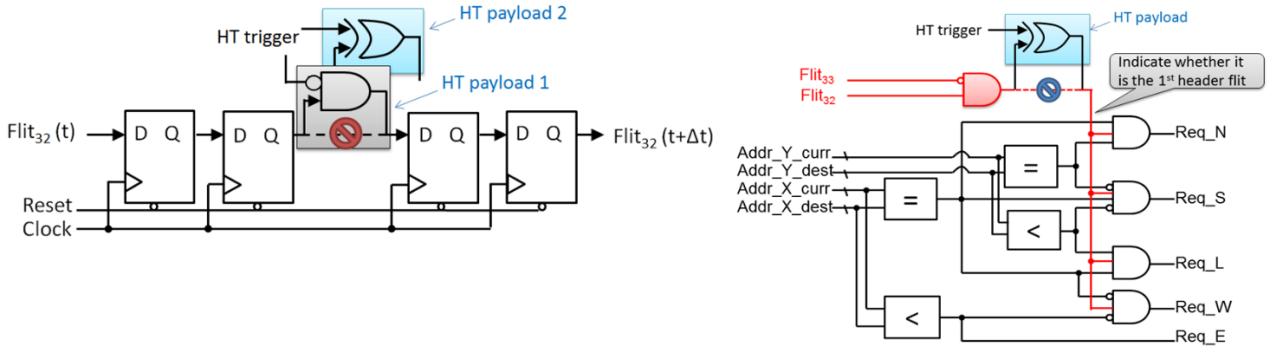


Figure 2.8. HT payloads affecting NoCs. On the left, a HT inserted in the FIFO will overwrite data, while on the right, HTs inserted in the Route Computation unit will cause incorrect path routing.

An architecture called a Data Protection Unit (DPU) is introduced in [34, 35]. The DPU is a hardware structure placed in the communication infrastructure of the NoC, particularly next to certain Network Interfaces (NIs). When a packet, the structure of data within NoC systems, is sent from an IP core, it enters its individual NI and is then transferred to the NI that belongs to the target IP core. The DPU will be placed within the NI of either the target or the sending IP core, in which it will read a part of the data within the packet and check its table to view the access rights of the packet's operation and either block the transmission or allow it to be sent. This approach also claims to offer no latency added as the DPU's operation is performed in parallel.

In [43], multiple design strategies of NoCs are presented to increase security. The division of a NoC into secure and non-secure areas increases security by separating crucial IP cores away from IP cores that may try to attack it. The first design strategy in strictly to prevent attacks that consume the bandwidth of the network. By implementing Virtual Channels, certain packets can be can be given a higher priority than other packets, allowing the higher priority packet to be allowed to complete its path before the lower priority packet. The non-secure areas of the NoC will generally send packets of lower priority and are allowed to enter the secure area, but will allow packets being sent in the secure area, which has higher priority, to complete their paths and not consume the secure area's paths.

The second design strategy in [43] presents three different methods to separate the secure and non-secure areas of the NoC. The first separation method only has non-secure IP cores in the non-secure area, and nothing else. A packet that is sent from the non-secure area enters the secure area and is forced to go through a NI. The NI will filter packets with irregular target paths, helping to counter unauthorized read/write attacks, and gives the packet a path that will limit path interruption with other packets, which helps to block DoS attacks like bandwidth consuming. The

second method has NIs and routers in the non-secure area to allow non-secure area IP cores to communicate with each other. The secure area contains a filter that only allows packets headed for the secure IP cores into the secure area. This prevents deadlock and livelock errors by preventing non-secure IP core packets headed for other non-secure IP cores from entering the secure area and interrupting secure packets. The third method uses path instructions to find incorrect or malicious data within packets. When a packet is sent, it is given a path instruction that it must follow in order to reach the target IP. Upon arrival at the target IP, the instruction must read "end of path," or else the packet is removed. Every router the packet passes checks the path instructions to ensure that the end-of-path instruction exists in order to avoid livelock and deadlock situations, and checks that the current instruction is not the end of path instruction. In either of those situations the router will remove the packet.

2.6. Chapter Summary

In this chapter, hardware security, Networks-on-Chip, hardware Trojans, and fault tolerance are discussed. Recent NoC security methods from IEEE conferences and journals were also presented. Although existing security methods address some attacks that can target NoCs, some more advanced attacks, such as Denial of Service attacks, can still hinder network performance.

Chapter 3. Networks-on-Chip

3.1. Baseline NoC Design

3.1.1. Design

To deploy our method, first we had to develop a Network-on-Chip testing environment. The testing environment was designed using the Hardware Description Language (HDL) Verilog. In Verilog, modules are made to represent hardware or logic. Lower modules can be interconnected within a top Module, to represent a larger circuit. A testbench is usually made that calls the top module and runs the design through given conditions preset by the user.

A 3x3 NoC was first designed given skeleton code provided by Dr. Qiaoyan Yu. The code had originally been for a 3x3 torus-topology NoC, but was ultimately modified to become a 4x4 mesh-topology NoC. Many of the following module names have been slightly altered to fit different tests. Therefore, a basic name will be given to each module to help the reader understand which module we are currently talking about. A top module where the 9 routers modules are instantiated and connected is the main goal. Each router was made within its own module named “five-port router.” The five-port router module is an individual router for the network, and is made up of several smaller submodules. A basic understanding of the five port router structure can be seen in Figure 3.1. The router should have 5 data inputs but it also needs a few other input signals to determine integrity, and also determine the status of the network. Write request signals, or wrreq in the code, are needed to be logic high when writing data to an input FIFO. CurrentID sets the source address for each flit, which helps determine how the packet will be routed by comparing CurrentID of the current router with the source address of an incoming packet. The source address of an incoming packet will be the CurrentID of the router connected to the NI the data came. Therefore,

comparing the source address and CurrentID, the direction in which the data should exit depending on the routing algorithm, can be computed easily. AlmostFull_in is another input signal to the five-port router. This signal allows the current router to see if it can send data to the surrounding routers based on knowing if the input FIFO of the router at the next hop is full or not. If this signal was not implemented, the network could be flooded and data could be overwritten before being sent out of the FIFO. The NoC routers are symmetric. Therefore, to have extra input signals being received at one router, this means routers must output more than just the data as well. Extra output signals include wrreq_out, which is connected to wrreq of a surrounding router and almost_full, which is connected to AlmostFull_in of a surrounding router. Lastly, every router is connected to the network's clock (CLK) and reset signals.

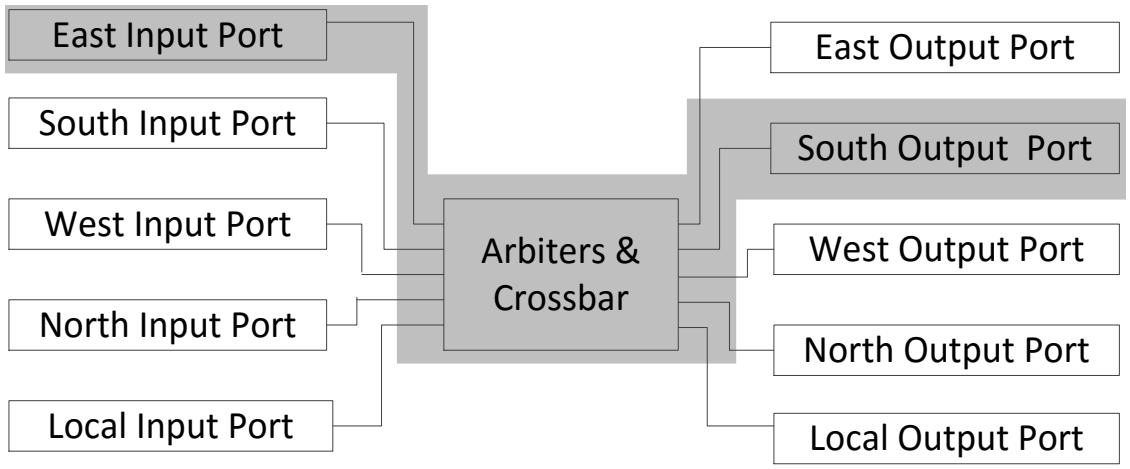


Figure 3.1. Baseline Five Port Router structure.

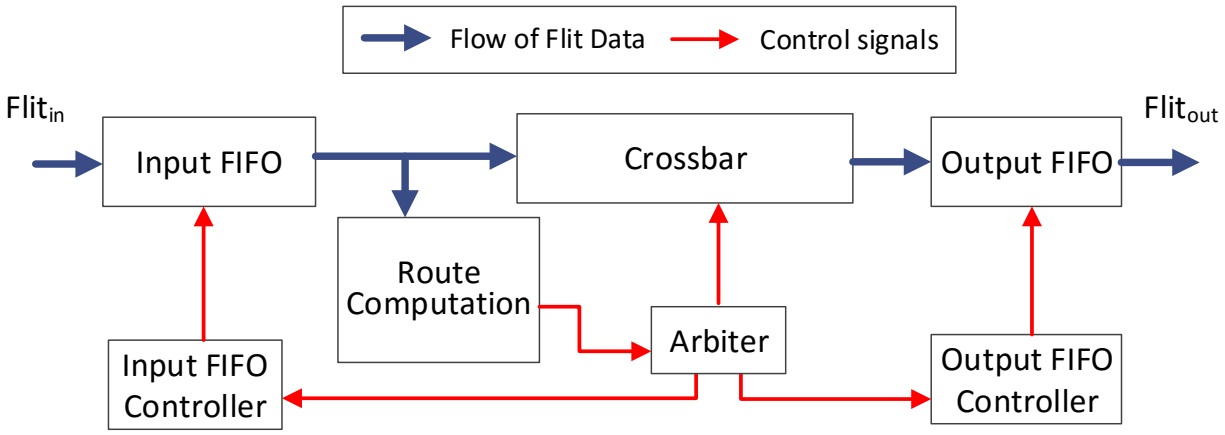


Figure 3.2. Flow of data from beginning to end of one input/output port pair.

Figure 3.2 shows the path of one flit from an input port of the five-port router to an output port of the five-port router. Basic signal flow and design submodules are also shown in Figure 3.2. The Verilog code for the hardware modules can also be translated to a schematic using Synopsys Design Vision. The design is first analyzed and elaborated by Design Vision, then a schematic of the design is displayed. The schematic, which can be found in the appendix section, shows several submodules in their basic I/O form for simplification, but examining this, it lets us see an image that looks similar to Figure 3.2. The submodules of five-port router include "Info Extraction," "Circular Output FIFO," "Input FIFO," "Round Robin," "Full Crossbar," and "Port Admission."

3.1.2. Submodules

3.1.2.1. Input FIFO

This module is the first that the flits enter. The input FIFO consists of 8 "stagereg31.v" submodules. Each stagereg31 module acts as 32 D-Flip-Flops (DFFs) in parallel. Therefore, the input FIFO has 8 "stages." In order to write to the FIFO, the write request signal "wrreq" must be logic high. The DFFs will store the 32 bit data in parallel until the data is ready to be moved out of

the FIFO. Each stage can effectively hold one 32 bit flit of a packet. If the FIFO depth is greater than the length of a packet, then the FIFO may hold more than one packet at a given time. The basic I/O and internal circuitry of the input FIFO module can be found in the appendix section.

3.1.2.2. Full Crossbar

After a flit leaves from the input FIFOs, it enters the Full Crossbar module. In addition to its own functionality, the Full Crossbar module also has two submodules. These two submodules are the Info Extraction and Port Admission modules. The submodules are responsible for passing signals to the Round Robin module, which is the arbiter that will determine output port priority, and for translating the critical flit data. The critical flit data translation will be passed to the Full Crossbar to grant an input port a connection to an output port. The submodules' basic I/O and internal circuitry for the Full Crossbar module can be found in the appendix section.

3.1.2.3. Info Extraction

The Info Extraction module looks at vital flit information and determines from which port the data should leave. Vital flit information are bits that, if altered, could cause adverse effects. The packet's destination address, source address, and flit type are all vital to the packet. If the destination address is changed, the data will be sent to the wrong address. This can especially clog networks, as they expect some sort of format in which data are sent, and not abiding by these rules may lead to congestion. If the flit type is changed, then the network may be congested while waiting for a specific flit type. If a tail flit is lost, the packet's transmission may become endless [45]. This is because another submodule's signal grants permission for the Full Crossbar to dedicate a single input-port/output-port pair to a packet once a header flit is detected. This will cause the input-

port/output-port connection to be locked, and it cannot be released to allow other packets to use that output port. The basic I/O and internal circuitry of this submodule can be found in the appendix section.

3.1.2.4. Port Admission

The Port Admission module grants permission for the Full Crossbar to connect an input data port to its respective output data port. Port Admission passes router input signals and Info Extraction output signals to its submodule Round Robin. The Round Robin module grants priority based on direction to the incoming/outgoing flits. The basic I/O and internal circuitry of the module can be found in the appendix section.

3.1.2.5. Round Robin

The Round Robin module is the router's arbiter. This module grants priority to certain ports over other ports in race conditions. This module calculates the Port_Ad internal signal, which is required for the crossbar to route data. Port_Ad determines the output port to which each input port with incoming data gets connected. This will make the Full Crossbar lock the connection between the input port and output port until a tail flit is detected. The basic I/O and internal wiring for the Round Robin module can be found in the appendix section.

3.1.2.6. Circular Output FIFO

The Circular Output FIFO module is used for the output buffers of the router. This module is responsible for reading the tail flit and determining if it should set the wrreq_out bit to. Wrreq_out

will act as the input wrreq to an adjacent router. Similar to the input FIFO module, inside the Circular FIFO module there are 8 parallel 32 bit shift register stages made using the stagereg31 element. The difference between the output and input FIFOs is that the output FIFO has a circular design element. The circular element of this FIFO is for NACK signals. The basic external I/O schematic and the internal schematic of the Circular FIFO module can be found in the appendix.

3.1.3. Top Module

Once the five-port router module and all of its submodules were verified to work, a top module was made to connect the 3x3 Mesh NoC. The top module should consist of 9 five-port router submodules. The routers should be connected appropriately to the surrounding routers. A basic router node can be seen in Figure 3.3. For simplicity, just the incoming and outgoing data are shown in Figure 3.3. The data for each router was named in the fashion shown in Figure 3.3. The data's appropriate name takes the form of "data_{[Router#],[Direction]}." The router number is just the ID number of the router in decimal. The direction bits are given as follows: leaving east = 0, leaving south = 1, leaving west = 2, leaving north =3, and leaving local = 4. Therefore, for router R0, the data out of the east port is named data_00.

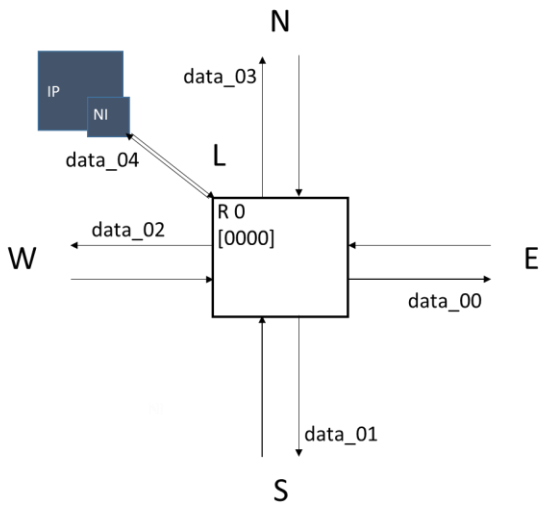


Figure 3.3. Basic I/O ports of a five-port router

To connect multiple routers, the I/O ports were connected based on Figure 3.4. The west port output of router R1 becomes the input of the east port of router R0, and the east port out of router R0 becomes the input of the west port of router R1. Data_12 will be a new input into router R0, and other input signals such as wrreq will need to be connected in this same way. Other inputs and outputs that are vectors of bits are named in a similar fashion as incoming and outgoing data. Therefore, input signals such as wrreq and data_in into the east port of router R0 will be named wrreq_12 and data_12. Output signals such as wrreq_out and dataout are named this way as well. The rest of the routers can be connected in a similar way, with an end result looking like Figure 3.5.

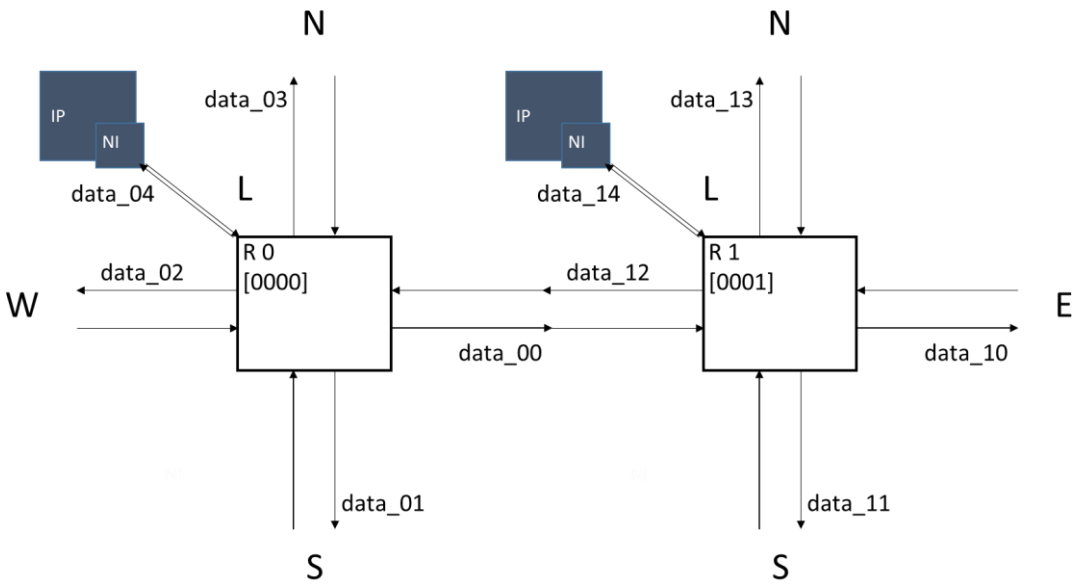


Figure 3.4. Configuring routers together to create the NoC topology.

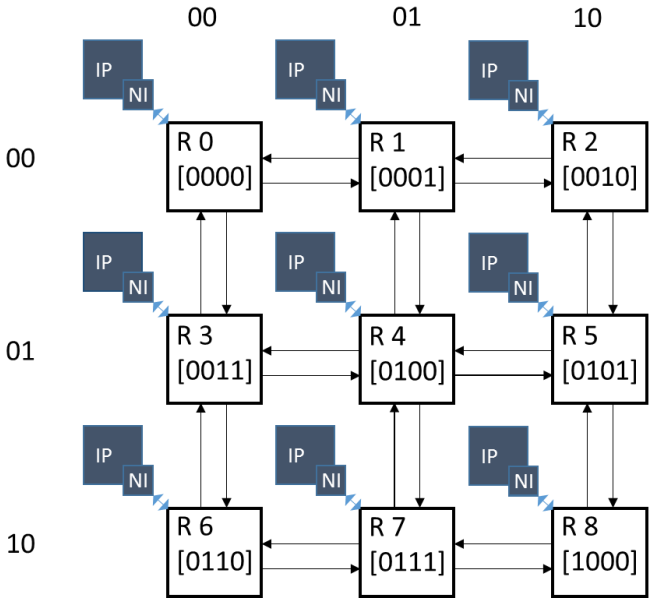


Figure 3.5. Final 3x3 NoC that was designed.

3.2. FPGA Implementation for 3x3 NoC

The 3x3 mesh NoC was synthesized to a Nexys 4 Artix-7 Field Programmable Gate Array (FPGA) to report the hardware cost of the design. A picture of the FPGA used for synthesis can be seen in Figure 3.6. Table 3.1 shows the FPGA slice/register utilization numbers. FPGA synthesis of designs is important, as it directly shows if the design can be implemented into hardware or not, and utilization numbers provide a designer with knowing how a design compares with others. Comparison of slice and register utilization will directly lead to the knowledge of area overhead, which can also give rise to knowledge of power consumption and possibly delay characteristics.

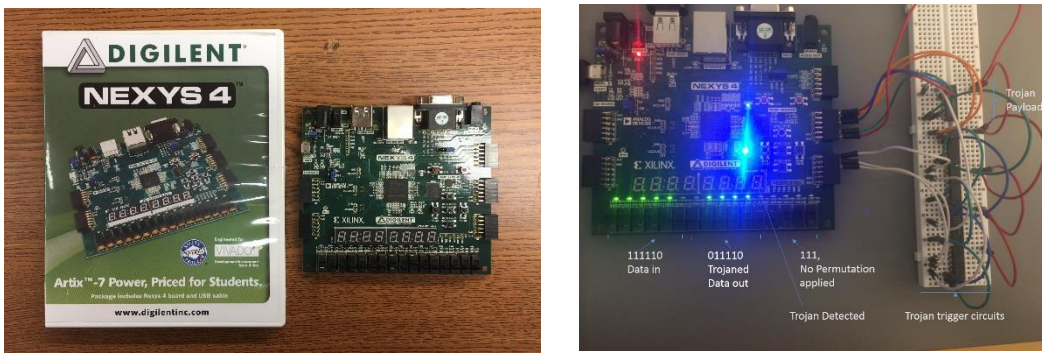


Figure 3.6. FPGA used for logic synthesis and measurement of area overhead.

Table 3.1. FPGA Logic Slice Utilization of 3x3 NoC design.

	Utilization	Available	Utilization (%)
Slice LUTs	11150	63400	17.59
Slice Registers	8950	126800	7.06

3.3. Testing the Baseline NoC

Once the 3x3 NoC was finished, both Mentor Graphics ModelSim and Cadence NCVerilog were used to compile and elaborate the design. NCSim was used to generate timing diagrams of each signal of interest. The testing and functionality of the design was verified using the signal waveforms generated during the simulation time. Next, the ability to send packets was tested by sending pseudo packets manually through the network and verifying that the packets reached the destination. This was done by manually injecting 32 bit flit vectors into the local port of a specific router. An initial statement of the testbench was used to inject the flits at certain times using delays. Initial statements execute once during a simulation time and are usually used to set initial conditions of signals. Initial statements are also not synthesizable. To generate a packet, first a header flit was injected, then several payloads, and then lastly a tail flit. Each flit is injected a clock cycle apart, as this is the minimum time that can be used to inject a single flit. An “Always" statement in Verilog, which activates whenever the value of its statement toggles, was added to the testbench to detect the presence of a received flit at the local output port of all routers. Any time the value of a local output port register was greater than 0 (any other 32 bit output flit besides all zeros), would trigger the “always” statements. The statement then extracted information such as flit type, source address, destination address, and how many cycles it took to get to the local port from when it was latched into the input FIFO from the source router’s NI. This information was then written to a dedicated .txt file for a router receiving the flit. This allowed one to observe if a sent packet was received at the target router or not. This would prove especially helpful when inserting hardware Trojans into the design later, where we visibly see packets arriving at incorrect destination addresses. Many conditions were tested, such as multiple packets arriving at a destination router at the same time to test priority, and multiple packets crossing a single router at the same time.

As the end goal of this research was to use a 4x4 mesh NoC, the number of routers was increased to 16. The final 4x4 NoC takes the form of the one seen earlier in Figure 2.4. As the number of nodes increased, it was found that it would be too difficult to check waveforms manually to observe outcomes. A lot of automation was built into the testbench to read and write data to and from text files. Text files were eventually used to read traffic trace files to determine input packets. Traffic trace files are with respect to the input local port of a router, and were generated using a custom built MATLAB script that would randomly pick destination addresses of other routers. A second clock was used to perform read operations from the trace files to read a destination address. The read destination addresses were of the form in Figure 3.7, where a read of 100 would correspond to a destination address of router R0, while a read of 1600 would correspond to a destination address of router R15.

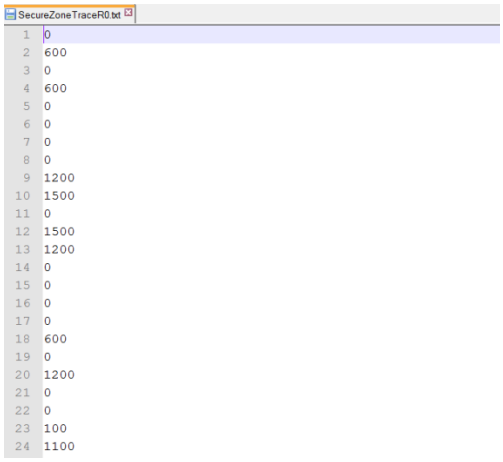


Figure 3.7. Format for trace file reads.

Once a destination address was read, a loop would be triggered that would latch the read destination address into the header flit of a new packet. The header flit is then sent out and the loop

would properly inject a predetermined amount of payload flits, and then finally a tail flit. Each flit is injected one clock cycle apart from one another. The clock period of the clock that sends out the packets has to be a fraction of the primary clock's period in order not to cut off packets midway through generating them. For N flits in a packet, the clock period relation can be seen in Eq. 3.1.

$$ClockPeriod_{(injection)} = ClockPeriod_{(primary)} \cdot \frac{1}{(N+1)}$$

Equation 3.1

There is an offset of 1 in the denominator that can be attributed to an empty flit (all zeros) placed after every tail flit of a packet. This was necessary to reset the write request signal from the local input port FIFO between packets to ensure the control signals operated properly. Equation 3.1 can be rearranged as seen in Eq. 3.2, where lambda is defined as the packet injection rate.

$$\lambda = \frac{\left(ClockPeriod_{(primary)}\right)}{\left(ClockPeriod_{(injection)}\right)} = \frac{1}{(N+1)}$$

Equation 3.2

Different packet injection rates were used to vary the amount of stress on the network due to traffic. The packet length N was chosen to be constant at 4 flits to ease the verification processes of using the waveforms. In testing, we used 9 different injection rates that varied between 0.0769 and 0.2 were used.

NoC traffic was monitored by recording the number of flits being transmitted over the links between the routers. This value once during each clock cycle and the value was written to a .txt file with each injection rate having a different file. This was done with a method similar to the one used to detect flits at the local output port. At the negative edge of every clock cycle, the testbench would

record how many links had values that were greater than 0, denoting there is currently a flit being transmitted over those links. This helped define the metric link availability, which is defined as the number of links not transmitting data during a clock cycle.

3.4. Chapter Summary

The basics of on chip networks, coding of the baseline NoC, FPGA synthesis of the NoC, and basic testing methods for the NoC were presented in this chapter.

Chapter 4. Proposed Collaborative Dynamic Permutation and Flit Integrity Checking

4.1. Big Picture of Proposed Method

The proposed HT mitigation method aims to detect and mitigate the HT attacks that (1) modify the flit type, (2) change the legal destination address to an unauthorized destination, and (3) sabotage the integrity of a packet.

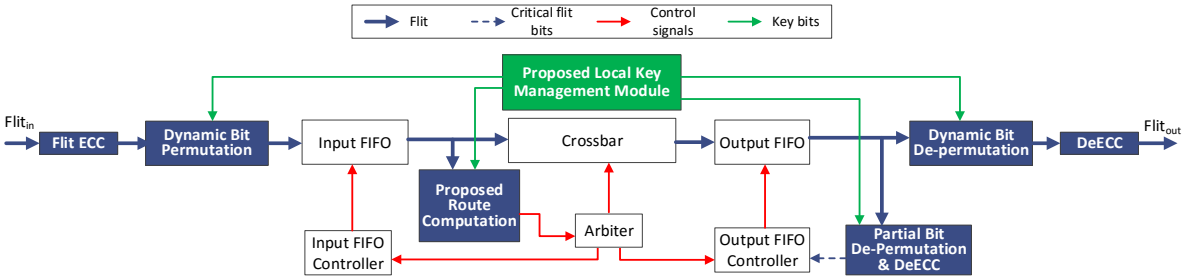


Figure 4.1. One proposed input and output port pair for hardware Trojan detection and mitigation. Solid blocks highlight our innovations over the generic router architecture.

A generic mesh-NoC composed of five-port routers is assumed, as shown in Fig. 3.18. One pair of input and output ports (shadowed area in Fig. 3.2) is used as an example to present the novelty of the router design developed for this thesis. As highlighted in the solid blocks in Fig. 4.1, seven changes were made to the generic router shown. To direct a packet to its original destination, the critical flit fields (i.e. source/destination address and flit type bits) cannot be altered. Hence, it is proposed to first encode the critical flit fields with a 1-bit Error Control Code (**ECC**) to address the HT-induced flit integrity loss. Next, a **Dynamic Bit Permutation** technique is applied to the

packet buffers so that critical information bits in a packet are '*randomly'* stored in the buffer from the adversary's point of view. The randomness is achieved by adding a key vector, which is dynamically generated by the **Local Key Generation Module.** As the key vector varies over time, the permutation configuration is uncertain at the static analysis time and thus the adversary cannot easily place HTs to perform successful attacks at the design time. Symmetrically, the **Dynamic Bit De-Permutation** is applied at the output port. After de-permutation, an ECC decoder (**DeECC**) facilitates the router to detect and correct all 1 bit errors on the critical flit content. The **Proposed Route Computation** unit is designed to interpret multiple versions of the permutated flit content into a request to use the correct output port for the next routing hop. To reduce the hardware cost, it is proposed to use ***Partial* Bit De-Permutation and DeECC** to correctly interpret important internal and external signals such as write request and FIFO buffer status signals.

4.2. Innovations over the Baseline

It is proposed to integrate a key-based dynamic permutation mechanism with an error control codec to address HTs inserted in the router. The overview of the proposed collaborative method is shown in Fig. 4.1. To minimize the hardware cost and maximize the error correction strength, only the critical flit bits (i.e. header, tail and destination address bits) are encoded with the flit ECC encoder. Together with the non-critical flit bits, the codeword from the ECC encoder is permuted before being stored in the input FIFO. The proposed route computation unit processes the critical flit bits by partially de-permuting and partially error control decoding before the normal route computation. This is because the partial de-permutation and decoder outputs are only used to generate a valid output-channel request. If the loss of flit integrity is detected, the flit, if correctable, is corrected or the packet is dropped. An uncorrectable bit error will trigger a permutation configuration update.

The flit permutation technique obfuscates the flit content to a certain degree, so that the attacker cannot easily implement a HT to modify the flit content. In addition, the permuting mechanism may distance the positions of the target bits of a DoS attack; as a result, not all critical flit bits are modified by the HT. The distributed error bits can allow the use of a simple error correction code to reconstruct the critical flit bits and prevent the depletion of bandwidth accordingly. The error detection and correction capability available in our router provides a second line of defense to thwart HTs if the attacker successfully bypasses the dynamic permutation.

As shown in Fig. 4.1, the local key applied to the flit permutation (FP), flit de-permutation (FDP) and partial flit de-permutation (PFDP) is the same. To further reduce the overhead induced by the key utilization, one can share one key with five pairs of input and output ports per each router. The proposed method can be extended to dynamically change the permutation configuration once an error is detected in the critical flit field bits.

4.2.1. Permutation

It is proposed to permute the flit bit positions before storing the flit to the input buffer, and de-permute the flit after the output buffer. The principle of this permutation is as expressed in Eq. 4.1.

$$flit_{dp}(t,i,j) = \Phi_{dp}\left[flit(t,i,j), key(t,i,j)\right] \qquad \text{Equation 4.1}$$

The variables *i* and *j* in Eqs. 4.1 and 4.2 are the router ID and the input port ID, respectively. Variables *flit(t)* and $flit_{dp}(t)$ are the original flit and the flit after dynamic permutation at the time t, respectively. The function $\Phi_{dp}[]$ stands for our proposed dynamic permutation algorithm, which is

a function of the incoming flit(t) and the key vector generated at run-time. Like other data permutation methods, the permutation algorithm itself is a linear mapping process. Figures 4.2 and 4.3 show two example cases of permuted bits given different key vectors.

To add a non-linear property to the permutation, we configure the permutation mapping with an unfixed key vector, *Key(i, j, t)*. The key vector is the output of a PUF-based structure and varies with process variations and the dynamic challenge vector X, as expressed in Eq. 4.2. The challenge vector *X(t, i, j)* depends on the routing history in each router's output port. More about the PUF-based controller follows in section 4.2.2.

$$Key(t, i, j) = f\left(Process\ Variation, X\left(t, i, j\right)\right) \qquad \text{Equation 4.2}$$

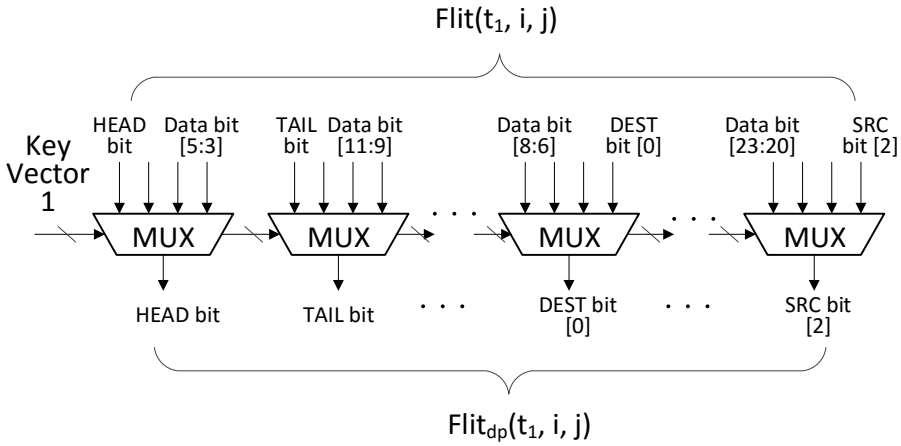


Figure 4.2. One example of using a multiplexer network to permute bits, which is decided by a configuration based on key vector 1.

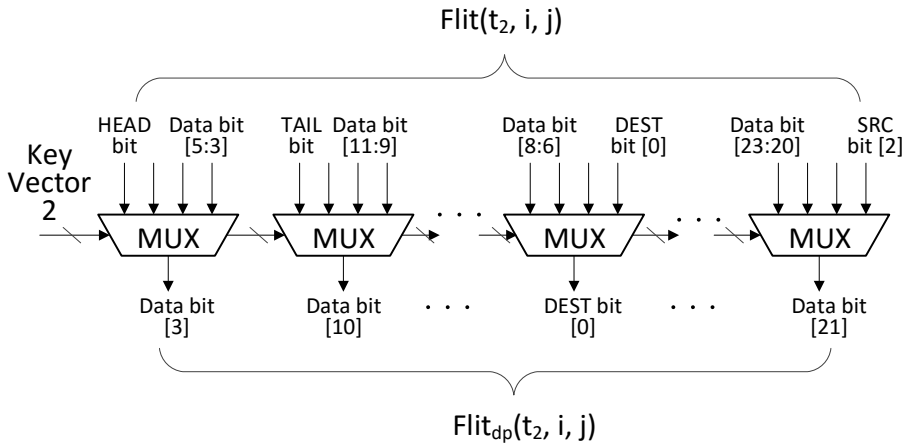


Figure 4.3. Another example of permuted bits given a different key vector.

4.2.2. PUF-based Local Key Generation

The key bits determine which permutation algorithm for the flit permuting and de-permuting modules to use at the current time. To prevent the attacker from attacking the permutation mechanisms, we proposed to dynamically change the key for the permutation algorithm at run-time. The key is generated locally in each router. As shown in Fig. 4.1, each pair of input and output ports needs to have one proposed local key management module. No central key management is needed. This further hardens our method against the adversary as the local router keys will differ at run-time. As the flit permutation configuration in each router is different, cracking one router would not guarantee the malicious packet to have full network access to successfully reach its destination.

We exploit the PUF-like structure [41, 42] and the traffic-pattern-dependent content saved in the round-robin register to generate a random key vector. As shown in Fig. 4.4, the PUF-like network is made up of multiplexers that propagate a flit bit along a path determined by the multiplexer challenge vector. Due to process variations, the propagation delays between multiplexer

inputs are slightly different. Thus, each multiplexer will have two different inputs. The challenge vector controls which multiplexer inputs propagate to their respective outputs. The output of the multiplexer is latched by the key registers at the key clock edge (*KeyCLK* in Fig. 4.4). The period of the user-defined key clock pulse should be longer than the time needed for all packets left in the input and output buffers to move to the next hop. If eight permutation algorithms are sufficient to obfuscate the flit, a 3-bit key vector is needed for run-time dynamic permutation.

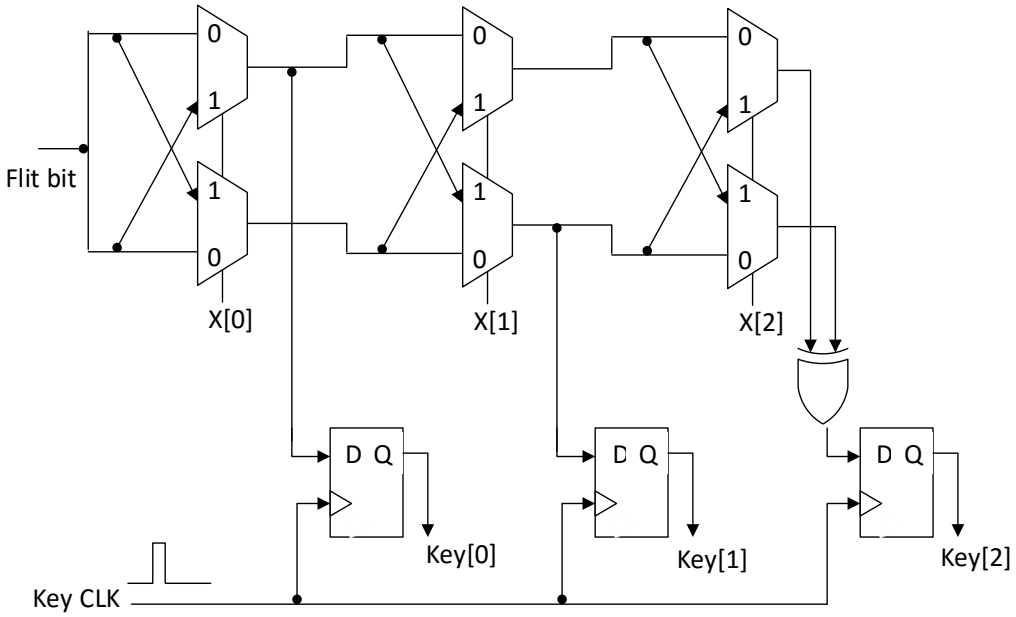


Figure 4.4. PUF-like structure used for key generation at run-time.

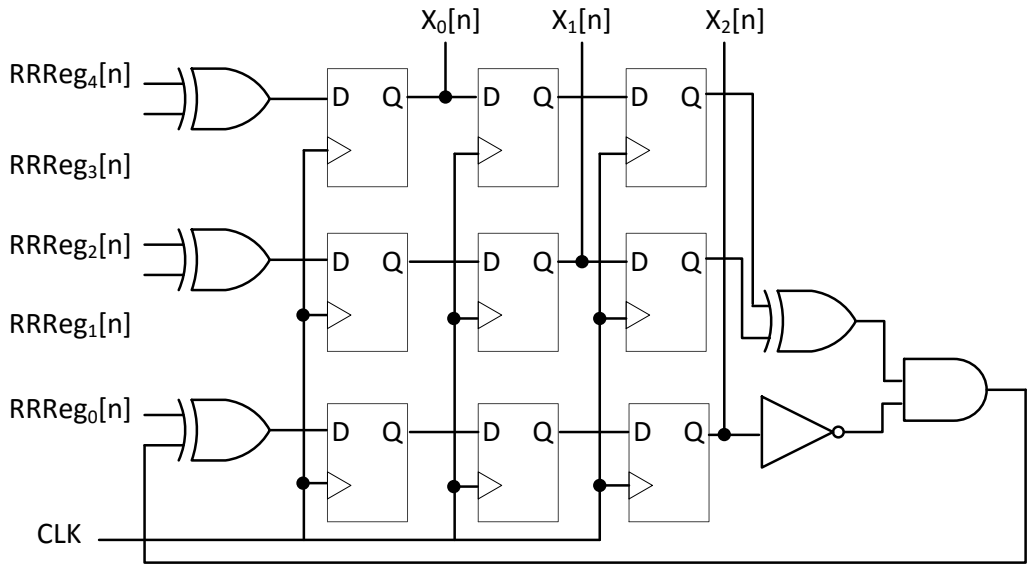


Figure 4.5. Challenge vector generation circuit.

Besides the process variation, the proposed method further dynamically changes the challenge vector using the internal value of the Round-Robin Registers (RRReg) in each router. As the RRReg value is set based on the output port requests from the real-time packet traffic, different routers will have unique RRReg values at a given time. Moreover, the RRReg value varies with different traffic traces extracted from the NoC application. It is proposed to exploit the time differential concept to design the RRReg-based challenge vector generator. Equations 4.3-4.5 express how the RRReg content at different clock cycles is used to generate the challenge vector X, where $RRReg_0[n]$ is the value of the 0th RRReg bit at clock cycle n, and $RRReg_0[n-2]$ is the 0^{th} RRReg bit stored at clock cycle (n-2). D Flip-Flops were used to create shift registers that store the RRReg history, as shown in Fig. 4.5. In addition, the key depends on the incoming flit, thus it cannot be predicted at the design time. This unpredictability further increases the difficulty to execute a successful HT attack at run-time.

$$X_2[n] = (RRReg_0[n-3]) \oplus ((!X_2[n-3]) \,\&\, ((RRReg_4[n-6]) \oplus (RRReg_3[n-6]) \oplus (RRReg_2[n-6]) \oplus (RRReg_1[n-6])))$$ Equation 4.3

$$X_1[n] = (RRReg_2[n-2]) \oplus (RRReg_1[n-2])$$ Equation 4.4

$$X_0[n] = (RRReg_4[n-1]) \oplus (RRReg_3[n-1])$$ Equation 4.5

4.2.2.1. Dynamic Key Update

The time relation of the key vector generation and internal router signals is shown in Fig. 4.6. When it is time to change the key, we wait until a tail flit arrives, signifying that a full packet has been received. As shown in Fig. 4.6, upon arrival of the tail flit (1), the input buffer controller forces the input buffer full signal to be high, in order to stop more flits from entering the input buffers. Once the input buffer full signal is forced to high, transition (2) happens, and the key generation network is turned on and allows an incoming flit bit to enter the key generation unit. As soon as all of the router's output buffers are emptied, transition (3) occurs. Transition (3) leads to a clock pulse for the key generation unit that is generated to latch the random key vector at the current time (4). The negative edge of the clock pulse leads to transitions (5) and (6), which turn off the key generation network, and sets the buffer signals back to normal, respectively.

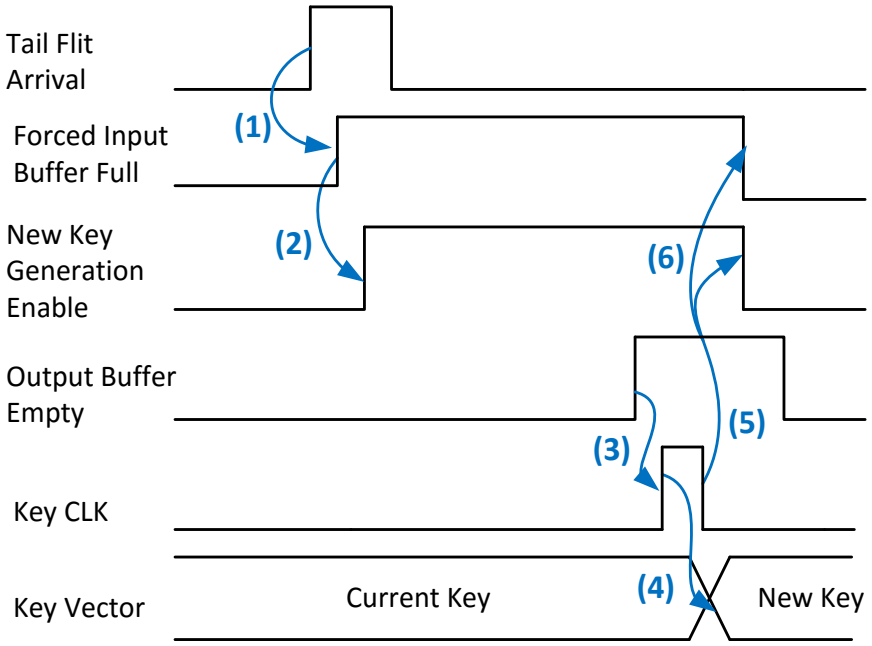


Figure 4.6. Example timing diagram of how dynamic key updates itself.

4.3. Proposed NoC Design

The proposed method mainly differs from the baseline design in the five-port router module. A system block diagram for the proposed five-port router module can be seen in Figure 4.7. The flit encoding, flit decoding, flit bit permutation, flit bit de-permutation, local key generation, and a configurable route computation module are the additions we made to the router.

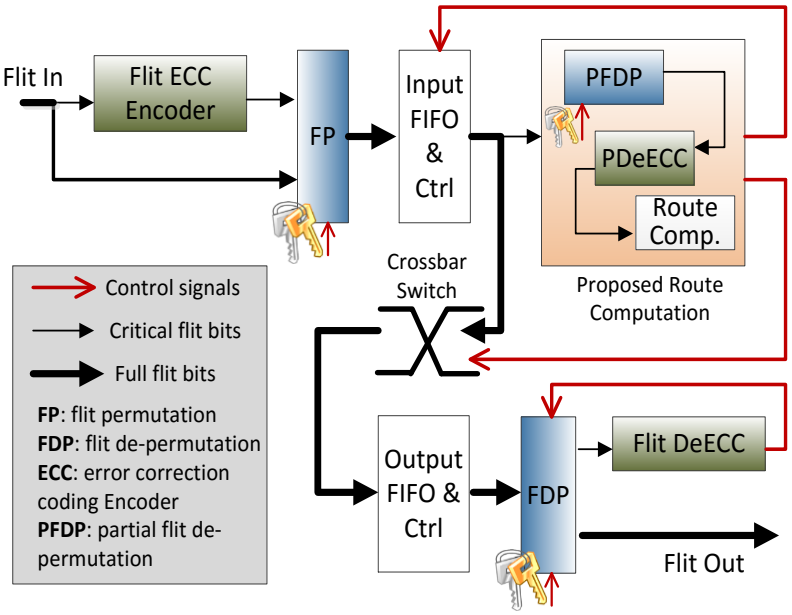


Figure 4.7. Block level schematic of the proposed five-port router, with flit flow.

4.3.1. Proposed Submodules

The addition of several submodules was needed in order to represent the hardware additions made over the baseline design. The five-port router submodules for the proposed method include, "Permutation," "De-Permutation," "Partial Flit De-Permutation," "Hamming(15, 11) Encoder," "Hamming(15, 11) Decoder." Many of the baseline submodules had to be edited as well, as an Error Correction Code (ECC) is part of our proposed method, and the 4 ECC parity bits increase the bit width from 32 to 36. As the ECC is on both ends of the router, all bit widths inside the router are 36 bits, and the bit widths over the external links remain at 32 bits.

4.3.1.1. Hamming(15, 11) Encoder

This module is responsible for applying the ECC to the incoming data. The ECC used was a Hamming (15, 11) code, which can correct all one bit errors.

When data is transmitted, the zone where the data is transmitted, between the transmitter and receiver, is called the channel. Channels differ and can be modeled in several ways, but usually the point of modeling a channel is to express the rate of error to which the transmitted data is exposed. A simple data communications scheme can be seen in Figure 4.8, where the message *u* is sent over the channel, and the message $\hat{u}$ is received on the other side. A hat is used over the received message to denote that there may be a change with respect to the input *u*.

Figure 4.8. Basic understanding of data communications over a channel.

With an original message without coding applied, upon arrival at the receiver there is no way to tell if that received message is actually what was sent. In channel coding, the goal is to detect and correct errors in the received message to increase the integrity of the data. This is done by concatenating additional bits into the message to create a codeword. The additional bits are formed from the parity subsets of the message bits. For a Hamming (n,k) code, where *k* is the message length, and *n* is the codeword length, the amount of parity bits *m* that are concatenated is equal to *n* – *k*, as seen in Figure 4.9.

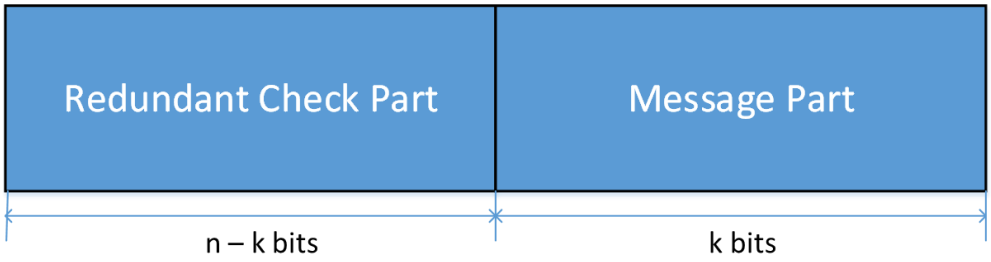


Figure 4.9. Codeword length (n) vs. Message length.

For a Hamming code with *m* parity bits, the message length and codeword length can be found in the equalities seen in Eqs. 4.6 and 4.7:

$$n = 2^m - 1$$ Equation 4.6

$$k = 2^m - m - 1$$ Equation 4.7

The addition of redundancy to the message lowers the effective rate of the code, as seen in Eq. 4.8.

$$Rate = \frac{k}{n} = 1 - \frac{m}{(2^m - 1)}$$ Equation 4.8

The m parity bits of a codeword are calculated using the generator matrix of the particular code. The generator matrix *G*, is a *k* x *n* matrix that encodes any message *u* into a codeword *v*. Eq. 4.9 expresses that any message *u* multiplied with the generator matrix *G*, will result in a corresponding codeword *v*.

$$v = u \cdot G$$ Equation 4.9

The generator matrix consists of two submatrices, a *k* x *k* identity matrix and a *k* x *(n – k)* submatrix formed by the code's parity check equations. This deduces that there are *m* parity check equations. The first submatrix places the original message into the codeword, and the second submatrix is responsible for tagging on the redundancy bits to the codeword. Eq. 4.10 shows the systematic form of a generator matrix.

$$G = [P|I_k] \quad \text{Equation 4.10}$$

Given a Hamming(15, 11) code, where codeword length is 15, and message length of 11, we therefore have 15-11 = 4 bits of redundancy. With m = 4, we have four parity check equations, which, for a Hamming(15, 11) code, are expressed in Eqs. 4.11-4.14 below:

$$P_3 = m_0 \oplus m_3 \oplus m_4 \oplus m_6 \oplus m_8 \oplus m_9 \oplus m_{10} \quad \text{Equation 4.11}$$

$$P_2 = m_0 \oplus m_1 \oplus m_3 \oplus m_5 \oplus m_6 \oplus m_7 \oplus m_8 \quad \text{Equation 4.12}$$

$$P_1 = m_1 \oplus m_2 \oplus m_4 \oplus m_6 \oplus m_7 \oplus m_8 \oplus m_9 \quad \text{Equation 4.13}$$

$$P_0 = m_2 \oplus m_3 \oplus m_5 \oplus m_7 \oplus m_8 \oplus m_9 \oplus m_{10} \quad \text{Equation 4.14}$$

Finally, the generator matrix for a Hamming(15, 11) code can be seen below in Eq. 4.15. To implement the encoder in hardware, only XOR gates are needed because they act perfectly as a bitwise modulo-2 addition operation. An equivalent circuit that generates one of the redundancy bits, made out of 2-input XOR gates, can be seen in Figure 4.10.

$$G = \begin{bmatrix} 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 0 & 1 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 \\ 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 1 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 & 0 \\ 1 & 0 & 1 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 & 0 \\ 1 & 0 & 0 & 1 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix}$$

Equation 4.15

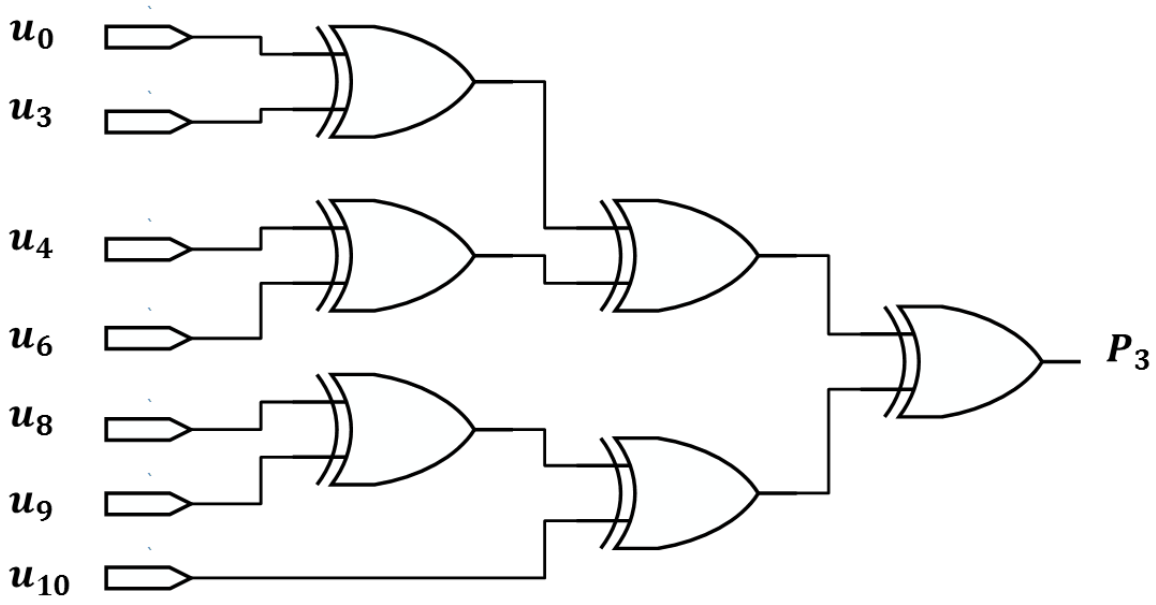


Figure 4.10. Equivalent circuit for redundancy bit calculation using XOR2 instead of XOR7.

A length of 11 un-coded bits was chosen, as we care mostly about the 10 bits in the critical flit field of each header flit. For our 4x4 implementation, the critical flit field bits consist of the 4 bit destination address, 4 bit source address, and 2 bit flit type. The flit structure for header, payload, and tail flits can be seen in Figure 4.11. The LSBs in this case are the critical field flit bits, meanwhile in real applications the MSBs correspond to the critical field flit bits.

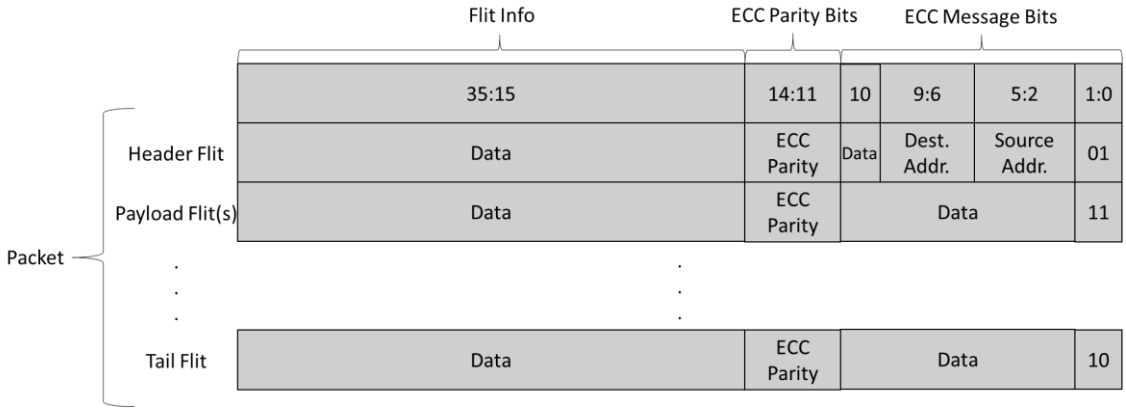


Figure 4.11. New 36 bit Flit Structures.

4.3.1.2. Hamming(15, 11) Decoder

This module removes the ECC redundancy bits that were concatenated into the flit, and also corrects one bit errors in the information bits. To correct errors, the module uses correction circuits based off of each error vector's syndrome. As any codeword can be made up by linearly adding other codewords, for a given error vector the set of all of the possible received codewords is made up of the Galois Field 2 (GF(2)) addition (also known as modulo-2 addition) of the error vector with the set of all of the possible generated codewords. Due to this linearity, all received codewords with a given error vector will exhibit the same syndrome. This means that each output bit has a dedicated correction circuit that will always flip and correct the corresponding output bit when the

specific syndrome is present that corresponds to an error on that bit. This decoding method is next examined in detail.

From the generator matrix seen in Eq. 4.10, one can obtain another matrix which is defined as the parity check matrix. The parity check matrix definition can be seen in Eq. 4.16:

$$H = \left[I_{(n-k)} \middle| P^T \right] \quad \text{Equation 4.16}$$

$$G \cdot H^T = 0 \quad \text{Equation 4.17}$$

$$e = r + v \quad \text{Equation 4.18}$$

$$r = v + e \quad \text{Equation 4.19}$$

$$s = \left(s_0, s_1, \ldots, s_{(n-k-1)} \right) = r \cdot H^T \quad \text{Equation 4.20}$$

The expressions seen in Eqs. 4.17-4.20 are also true for Hamming codes. The variables r, v, e, and s, in Eqs. 4.18-4.20, correspond to the following: v is a valid codeword generated by Eq. 4.9, e is an error vector, r is the received codeword which is a modulo-2 sum of a valid codeword and an error vector, and finally s is the syndrome. The H matrix used in this work can be seen in Eq. 4.21. As Hamming codes are both perfect and linear, rather than equating v to a specific codeword and e to a specific error vector, one can instead generalize both of these variables to a set of all possible codewords and error vectors respectively. Another interesting property of linear codes is that by combining two valid codewords, one can create another valid codeword, and this is how a set of all possible codewords can be made. This linearity means that in Eq. 4.20, all valid codewords will give the same syndrome, and all received codewords that incur a specific error vector will generate the same syndrome. Generalizing, all one bit errors on the same bit position will trigger the same syndrome, and detection of this syndrome can lead to the detection of an error on that

position, and then the bit error can be corrected. To correct a bit error on a given position, the corresponding syndrome can be used to create an error-correcting circuit for that one bit position. An example error-correcting circuit for a Hamming(15, 11) code is shown in Figure 4.12.

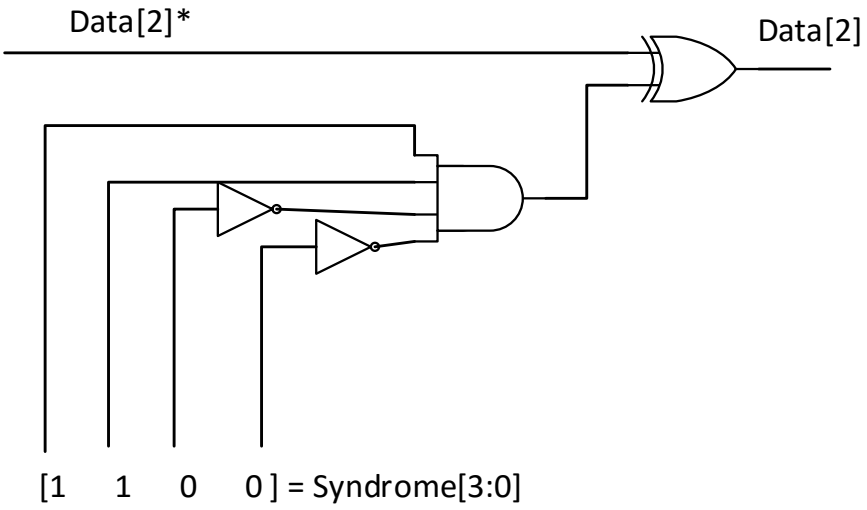


Figure 4.12. Example of error correction circuit. The AND gate will only have an output of one when the corresponding syndrome is met. When that syndrome is met, the output of 1 from the AND gate, will flip the respective bit using with the XOR gate.

$$H = \begin{bmatrix} 1 & 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 \\ 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 1 & 1 & 0 & 1 & 0 & 1 & 1 & 1 & 1 \end{bmatrix}$$ Equation 4.21

4.3.1.3. Permutation

The Permutation module is responsible for permuting the input data. For the method presented here, 8 different permutation configurations were used as this provided “good enough”

flit bit obfuscation. The permutation method works as explained in section 4.2.1. The permutation module was originally designed for 32 bits, but was extended to 36 due to the ECC parity bits.

4.3.1.4. De-Permutation

The De-Permutation module is responsible for de-permuting the output data, prior to decoding. For the method proposed here, 8 different permutation configurations are used and therefore 8 de-permutation configurations were needed. The de-permutation module works as the dual of the permutation module, and acts to permute the data back to its original position before permutation.

4.3.1.5. Partial Flit De-Permutation (PFDP)

The Partial Flit De-Permutation module is a submodule within in the Circular FIFO and Info Extraction modules. This module provides permutation for an output of shorter length than the length of the input. The number of MUXes needed is equal to the amount of outputs. Still, the 36 bit (extended because of parity) flit data is needed as an input to the module due to the flit content being able to be permuted to any other position in the flit.

4.3.1.6. PUF

The PUF module is responsible for dynamically updating the key vector that controls the permuting modules. The PUF network accepts a bitstream, and that bitstream propagates along MUXes with different delays on the lines. The MUX lines are controlled by the challenge bits generated by the hardware seen in Figure 4.5.

4.4. Chapter Summary

In this chapter, the main design of the thesis was presented. Submodules that build upon the baseline NoC to create the proposed method were also introduced. In the next chapter, the proposed

method presented here will be tested and compared to existing NoC security methods. Several metrics will be used to measure the integrity and security of each method under the effects of certain hardware attacks.

Chapter 5. Testing of Multiple NoC Methods

5.1. Attack Model and Experimental Setup

The primary focus of this thesis work is at the router level, therefore the techniques proposed here address the hardware Trojans that perform Denial of Service (DoS) type attacks in NoC routers. The consequence of DoS attacks includes bandwidth depletion, incorrect path routing, deadlock and livelock [7]. More specifically, hardware Trojans in routers can maliciously change the source/destination address or flit type information of a packet that has left the transmitter NI. If a Trojan payload modifies the destination address of a packet, that packet could be directed to an unauthorized IP core. A drop of the header flit or tail flit will result in the incomplete packet being retained in the router until some operation arrives to reset the router. In the rest of this section, all methods above suffering those HTs which are injected between the Input FIFO and the route computation module, are evaluated. Three HT types—destination address (i.e. DEST HT), header bits (i.e. HEAD HT), and tail bits (i.e. TAIL HT) — were considered in the experiments presented in this section. The HT injection spots were on routers center routers R5 and R9, and corner router R0. HTs were injected into the middle routers as the incurred effects should be more detrimental for routers with more traffic passing through. A corner router was chosen as another injection spot to view the effects of a HT inside a router with fewer I/O ports.

The proposed method was compared with existing methods [34, 36, 38, 43]. The work in [34] presents a data protection unit (DPU) in detail to address the potential hardware Trojans attacking the NoC NI. Saponara et al [44] follow a similar idea to detect if the memory access is out of the secure memory range in the NI. It is assumed that address filtering should always be done at the network interface. Hereafter, DPU is referred to as *Baseline* in the rest of this work. In [36], a counter-based traffic adjustment NI is presented to address the packet loss and thus bandwidth

depletion due to hardware Trojans. Here to this method is refrred to as *NI-Term*. When a packet without a tail flit is left in the NoC network, that packet cannot move forward and will block future packets entering the network, thus resulting in the network bandwidth depletion. Timeout mechanisms [45, 46] are typically utilized to resolve the bandwidth depletion induced by deadlock or link faults. Similarly, the work [38] uses a counter to terminate the current packet transmission if malicious behavior is detected. In this thesis he timeout mechanism is applied to the router by manually inserting a tail flit after a pre-determined time in order to terminate the transmission of the current packet. This method is referred to as *R-Term*. The data protection unit [34] and the wrapper method in [36] are not designed for router level hardware Trojan detection. The work [45] proposes to investigate the feasibility of applying address filters to NoC routers. Despite minor differences, the key principle for the DPU and Wrapper methods is to check whether the pair of source and destination addresses carried in the header flit matches to the table of legal source-destination pairs. If the address pair is not legal, that packet is filtered out by the router. Hereafter, this method is referred to as *R-AddrFilter*.

5.2. Generation of Test Scripts, Traffic Patterns, and the other NoC Designs

A lot of time and effort was put into recreating the NoC designs to which the proposed methods of this thesis are compared with. The baseline design presented in Chapter 3 was used as the DPU design. This is fair in the sense that this design's protection method lies in the NI and anything that happens over the NoC fabric will not be observed until the receiver NI. Therefore, all of the designs, including those proposed here, use the DPU method in the NI. Again, this is fair because it is NI-level protection, and does not directly affect NoC-level protection, as will be shown in section 5.3. The NI-Term, R-Term, and R-AddrFilter methods were all built on top of the baseline method to create separate testbenches.

As packet injection rate, the type of HT, and the number of HTs injected into the system, there were many tests involved. This is because each of the possible cases should be tested for each design. A total of 108 simulations of each design were generated, as this equates to 3x4x9, which represent HT cases (DEST, HEAD, TAIL), injected HT numbers (0-3), and the 9 packet injection rates used. As there are a total of 5 different designs, a minimum of 540 simulations were needed to generate the data presented in 5.3. Each simulation could take several minutes to do, and parameters in the code would have to be changed manually, so it was unrealistic to manually record the data of all 108 cases. Therefore, a bash script was developed to test multiple Verilog designs using a bash script. The bash script is also able to compile modules and change internal parameters, which would prove helpful. The use of Define functions in Verilog enabled the scripts to change internal parameters of the module with external commands. Each script contains 108 sections, with each section having many lines and a small change compared to the others. Python was used to generate the testing scripts for each design, as this would prove more efficient than manually writing the thousand plus lines in each script.

To be consistent with the other methods, the new NoC designs were tested in the same way as the reference designs. It was decided to split the NoC into two zones as the other methods do; secure and non-secure zones. A basic understanding of the two zones can be seen in Figure 5.1. Figure 5.1 also shows the injection sites of the HTs. These two zones are based on the integrity of the IP cores to which the routers are connected. There is no functional difference between any of the routers in either the secure or non-secure zones. A new traffic trace had to be generated to allow only the secure zones to send packets to each other and vice versa for the non-secure zones. MATLAB's random number generator (RNG) was used to generate all of the traffic traces needed for both debugging and testing the designs. This would allow for a proper, yet random, distribution

of the destination routers of each traffic trace file. For instance, router R0 has the same probability to send a packet to router R1, R4, R5, R10, R11, R14, or R15.

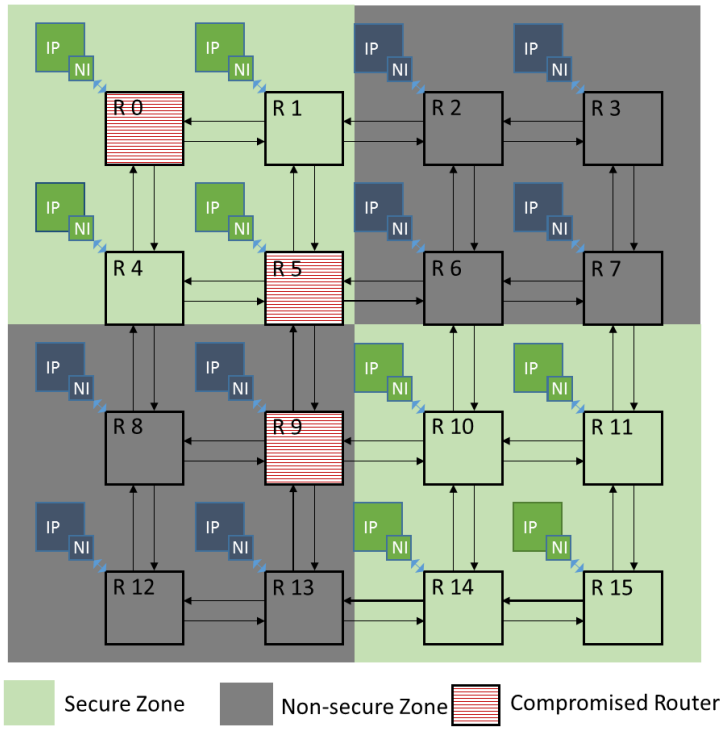


Figure 5.1. NoC split into secure and non-secure zones for testing.

5.3. Results

5.3.1. Throughput

The throughput of different methods was compared by examining the total number of valid packets received by the NoC NIs in the given simulation time. A valid packet was considered as one that eventually reaches its original destination address rather than a modified destination. By comparing the performance shown in Figs. 5.2 (a), (b), and (c) one can see that the proposed method receives the same amount of valid packets for the same traffic injection rate, regardless of how many HTs are inserted. In contrast, other methods drop significantly in the number of received valid

packets as more HTs are inserted. The traffic injection rate, λ, was 0.2 packets per cycle per node. For the single DEST HT scenario, the proposed method improves the number of received packets by up to 6.4%, 70.1%, 14.3%, and 11.5% over the baseline, NI-Term, R-Term, and R-AddrFilter, respectively. For the three DEST HTs scenarios, the proposed method achieves even better throughput performance than other methods. The proposed approach can successfully receive 43.1% more packets than R-AddrFilter. The proposed method is better than the other methods for the DEST HT scenario because it is able detect and correct a 1 bit change to the destination address.

In Fig. 5.3 the impact of HEAD HTs on the number of received packets is shown. Again, our method outperforms other approaches. This is because when a header flit is dropped, the other methods cannot recover that flit, and the packet is lost. The throughput of the proposed method remains unchanged with the introduction of HTs. The other methods, however, receive 43.3% fewer packets in the three HEAD HTs case than the zero HEAD HT case. In addition, the throughput of the R-Term, R-AddrFilter and baseline methods will remain equal, as no protection is provided by those methods against HEAD HTs. This is different than the throughput performance shown in Fig. 5.2.

When the TAIL HT causes tail flit losses, a more radical change in the number of received packets is observed than in the prior HT cases. As shown in Fig. 5.4, the proposed method outperforms the other methods by receiving more valid packets when TAIL HTs are introduced. The proposed method achieves the same throughput in the TAIL HT cases as in the HEAD HT cases. The impact of the different HT types on the throughput of the other methods is more significant. For instance, as shown in Fig. 5.4(b), the R-AddrFilter method receives 94% fewer packets than the proposed method. This performance drop is more than what was observed in the HEAD HT case. The performance drop is more detrimental because the absence of tail flits will not

allow all routing paths to release, resulting in the network bandwidth depleting rapidly. The R-Term method is designed to handle TAIL HTs after a pre-defined waiting period. As the number of inserted HTs increases, multiple routers waiting for the timeout to insert tail flits will block new packets from being injected to the network, thus resulting in a significant decrease in the number of received packets.

Compared to the baseline method, the proposed method improves the throughput by 6.4% (for 1 DEST HT case) and 43.12% (for 3 DEST HT case) at the greatest injection rate. When the HT controls the header flit bits of the NoC packet, the proposed method provided 11.6% and 43.67% increases in throughput over the baseline method for 1 HEAD HT and 3 HEAD HTs, respectively. If the HT compromises the tail bits of the NoC packet, the proposed method achieves up to a 98.8% improvement in throughput for the 3 TAIL HTs scenario over the baseline method.

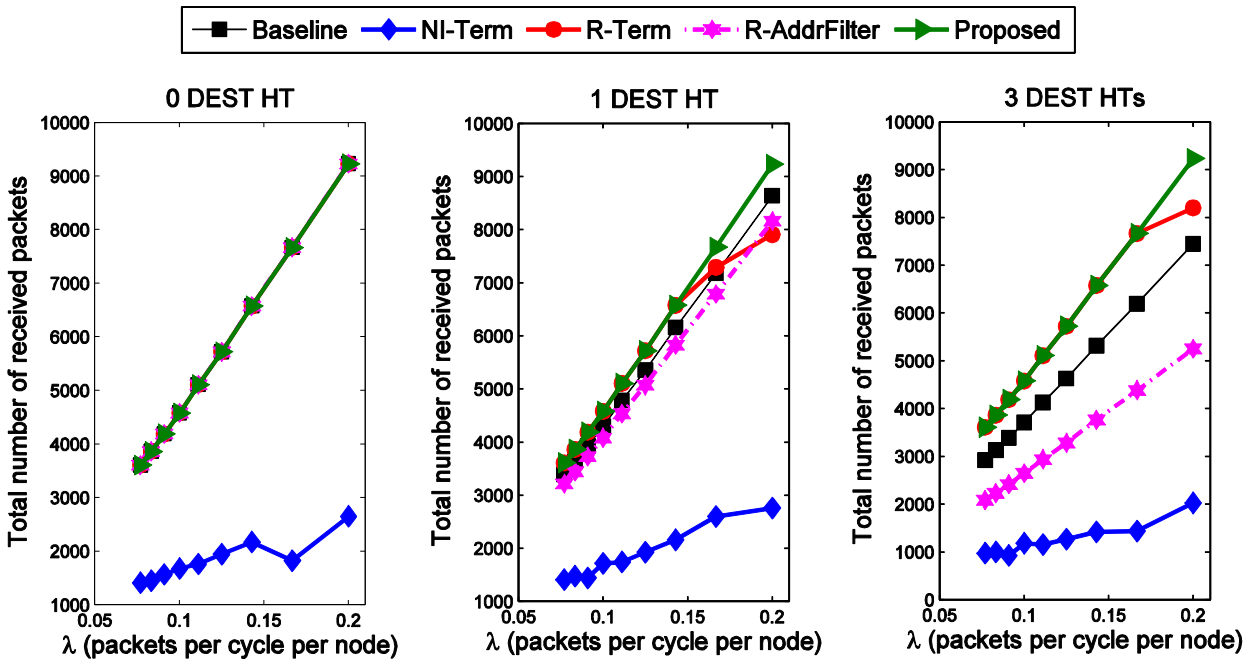


Figure 5.2. Number of valid packets received. (a) No HT, (b) 1 DEST HT, and (c) 3 DEST HTs.

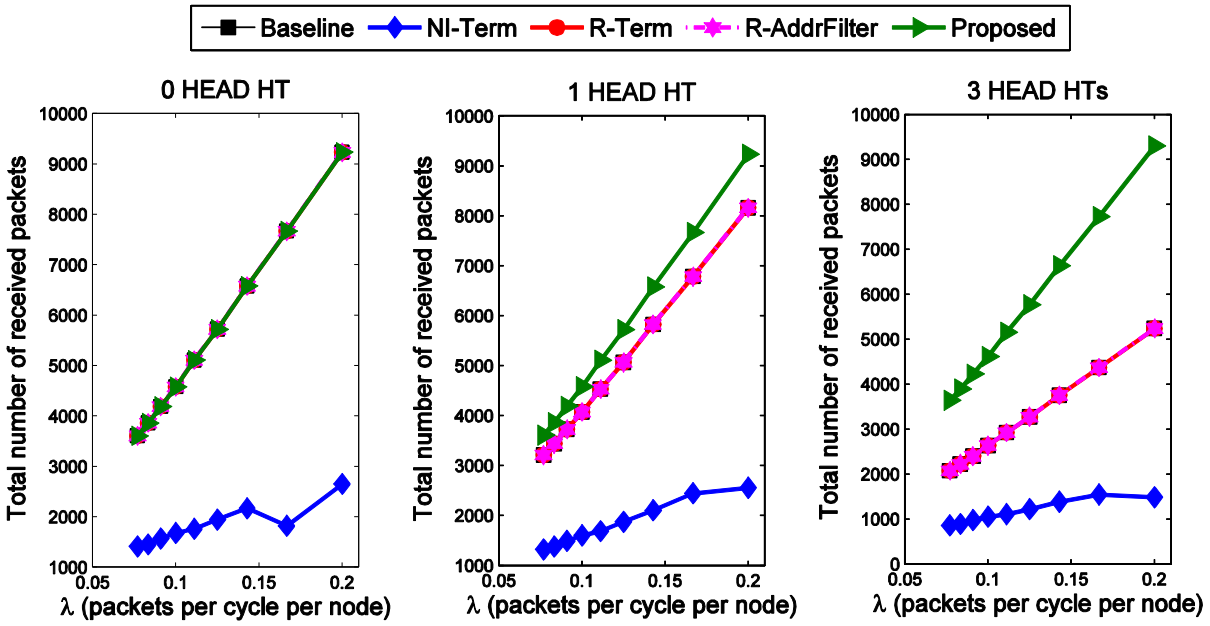


Figure 5.3. Number of valid packets received. (a) No HT, (b) 1 HEAD HT, and (c) 3 HEAD HTs.

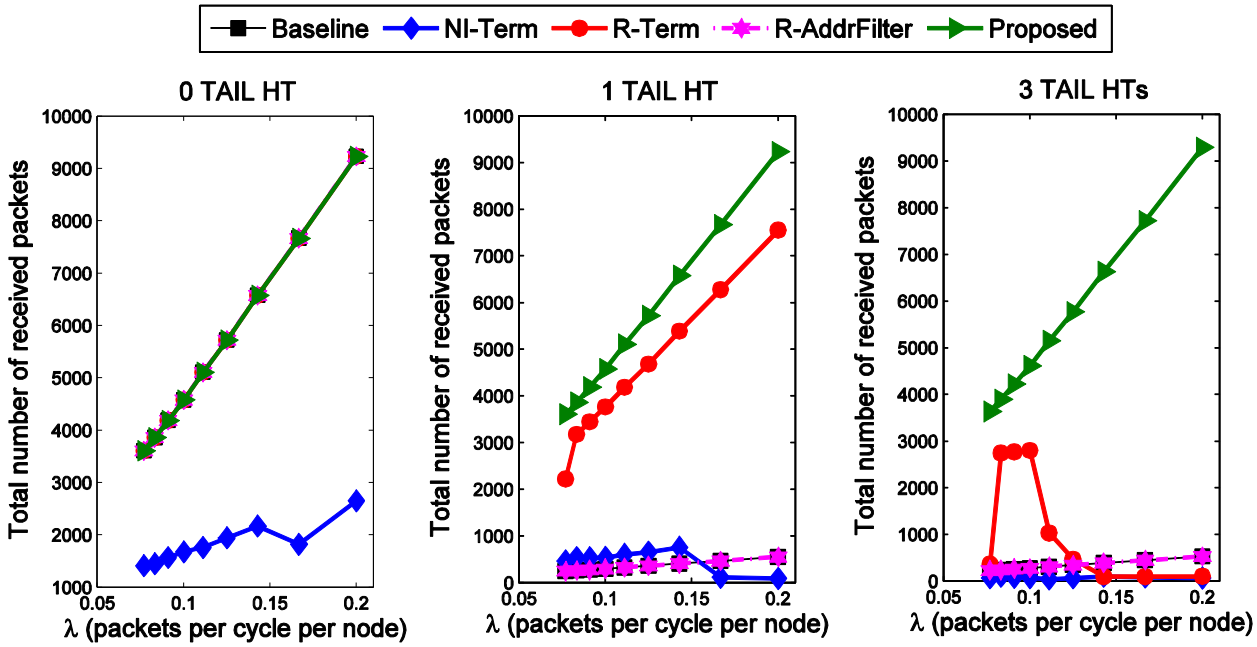


Figure 5.4. Number of valid packets received. (a) No HT, (b) 1 TAIL HT, and (c) 3 TAIL HTs.

5.3.2. Link Availability

Variation in the number of received packets shown in Figs. 5.2-5.4 demonstrates, from a throughput perspective, how different methods are intended to mitigate bandwidth depletion if a HT is present. In this section the effectiveness of each HT mitigation method from a link availability point of view is examined. The link switching activity of each NoC link is recorded throughout the simulation, and the total number of idle links at each clock cycle is accumulated. Given a traffic trace, a smaller number of idle links indicates a more severe level of bandwidth depletion.

In equation 5.1, the network bandwidth availability as the ratio of the number of cycles that *n* links are idle over the total number of simulation cycles.

$$Availability_{n\ idle\ links} = \frac{Cycles\ \ for\ n\ Idle\ Links}{Total\ Simulation\ \ Cycles} \qquad \text{Equation 5.1}$$

The subplot (a) depicted in each of Figs. 5.5-5.7 shows the link availability of the baseline NoC without a HT insertion. Figure 5.5 shows the bandwidth availability of the NoC given a specific HT mitigation method against three DEST type HTs injected into the NoC router. If the histogram plot displays a peak towards more idle links for a higher availability, then the corresponding HT mitigation method successfully manages the presence of HTs. In contrast, if the peak is shifted more toward a smaller amount of idle links, then the method may be suffering more from the depleted link bandwidth. As shown in Fig. 5.5, the R-AddrFilter and NI-Term methods exhibit less link availability than the baseline, R-Term, and proposed methods if the DEST HTs appear in the routers. The proposed method achieves a 19.1% and 22.5% higher average link availability than the R-AddrFilter and NI-Term methods, respectively.

The impact of three HEAD HTs on the link availability achieved by different methods is next examined. Comparing Figures 5.5 and 5.6, one can see that only the proposed method obtains

the same link availability with that of the baseline without any HTs. This is because the proposed method detects the HEAD HT and reconstructs the packet header flit in the same clock cycle, while the other methods lack protection against HEAD HTs. With three HEAD HTs injected in the NoC routers, the average number of idle links for the baseline, R-Term, and R-AddrFilter methods is reduced by 10% over the baseline NoC with the 0 HT case. The NI-Term method suffers from a 29.8% decrease in the average number of available links over the 0 HT case. The decreases in the link availability of the other methods are a result of the extra payload and tail flits propagating into the router FIFOs, hindering the network performance as those flits will waste the FIFO buffer space.

If three TAIL HTs are injected to the NoC routers, the proposed method is the only one that follows the trend of the baseline without HTs case, as shown in Fig. 5.7. Due to the HT payload effects, the other methods show signs of bandwidth depletion as certain links have data stuck on them for extended periods of time. The proposed method improves the average link availability by 33.7%, 33.7%, 26.4%, and 43.7% over the baseline, R-AddrFilter, NI-Term, and R-Term methods, respectively.

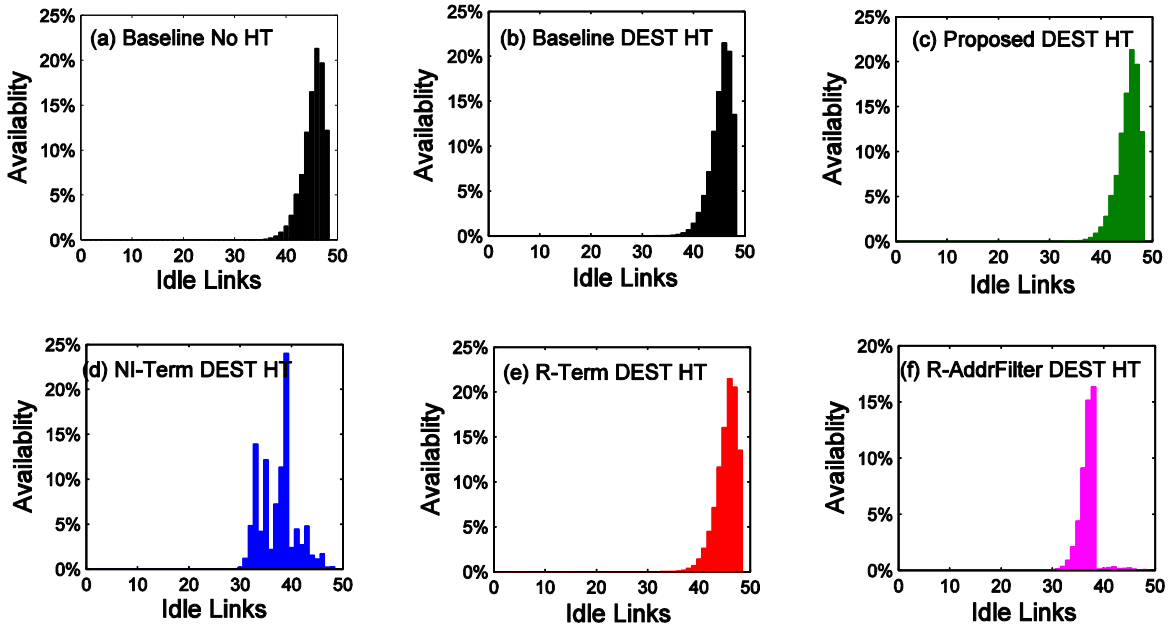


Figure 5.5. Link availability comparison among methods that suffer from DEST HTs.

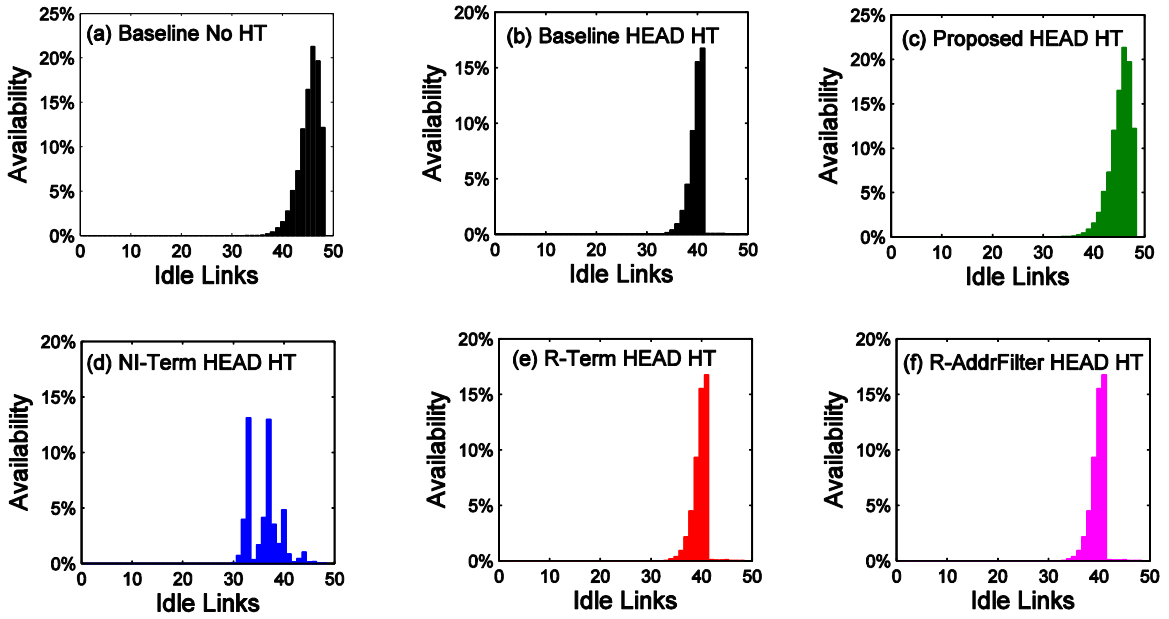


Figure 5.6. Link availability comparison among methods that among methods that suffer from HEAD HTs.

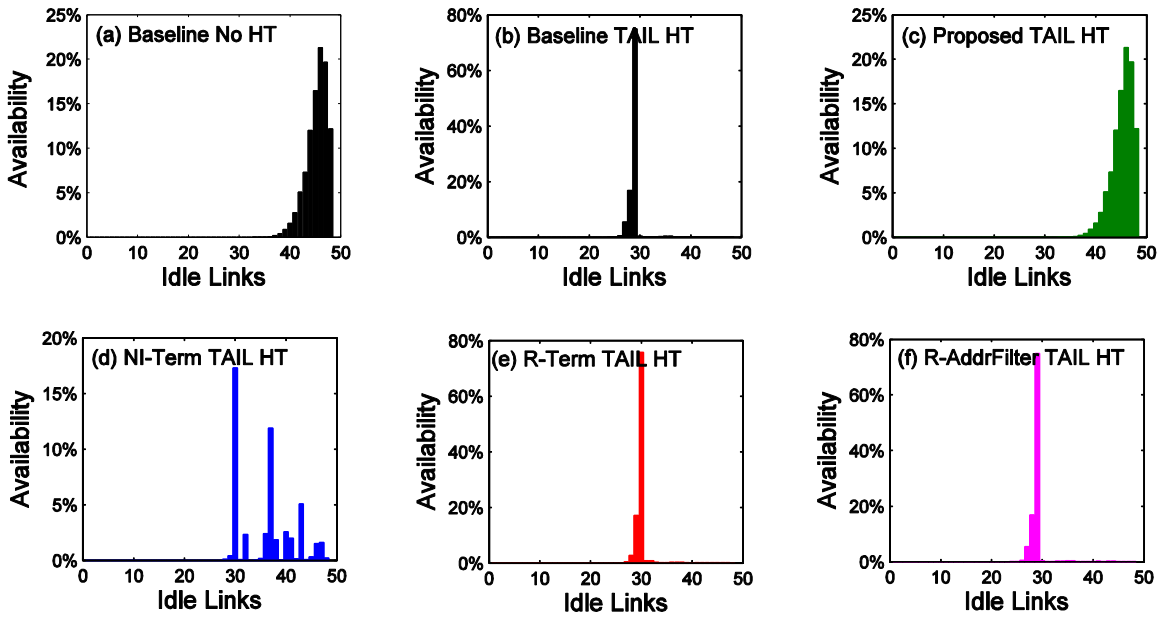


Figure 5.7. Link availability comparison among methods that among methods that suffer from TAIL HTs.

5.3.3. Traffic Hotspot/Coldspot Migration and Bandwidth Depletion

The arrival activity of new flits at each output port of the routers was monitored to estimate the switching activity (thus power consumption) of each router throughout the entire simulation. The total number of different flits transferred through each router is accumulated and then plotted in a node-type plot, as shown in Figs. 5.8-5.10. Each square in the figure represents the switching activities of one router with the corresponding coordinates in the physical NoC implementation. The color of each square indicates the intensity of the number of output port switching operations occurring over the entire simulation period. Figures 5.8-5.10 show the impact of two DEST HTs, two HEAD HTs, and two TAIL HTs on the router switching activities, respectively. The tile plots of each method were compared to the baseline NoC without HTs in the network, which is subplot (a) in each figure.

Compared to the tile plots in Fig. 5.8, the proposed method exhibits the closest activity compared to the baseline with 0 HTs. The proposed method has more intense switching on two routers over the baseline due to a measurement error, which whenever correction occurred it counted as two switching operations instead of one. Less switching of the routers in the other methods is either because the contaminated packets are filtered out by the HT mitigation method (e.g. R-AddrFilter), or because the packets are transmitted based on a counter (e.g. NI-Term). The baseline NoC with 2 DEST HTs differs from the 0 HT case at a few routers due to data being routed to the wrong local port. Another example is router R1 (X=2 and Y=1) in the baseline NoC, which exhibits more switching activity with 2 HTs than with 0 HTs. This is because R1's neighbor routers R0 (X=1 and Y=1) and R5 (X=2 and Y=2) are the HT injection sites. Traffic mitigation due to the DEST HTs can also be seen on router R4 (X=1, Y=2).

The tile plots for the HEAD HT case are shown in Fig. 5.9. As all of the methods, excluding the proposed method, cannot prevent the effects of the HEAD HTs, those methods will have less link usage at the affected routers. This is an indication that the R-Term, Baseline, R-AddrFilter, and NI-Term methods receive fewer packets than the baseline NoC without the HT case, which is confirmed by the results in Fig. 5.3. Note that the NI-Term method receives the least number of packets as the total traffic for the simulation is less than that of the other methods due to the counter-based traffic adjustment.

Figure 5.10 shows the tile plots for the traffic condition of the NoC with different HT mitigation against two TAIL HTs. Only the R-Term and proposed methods can mitigate TAIL HTs, and therefore the other methods will suffer from bandwidth depletion. The proposed method corrects the tail flit if a HT is detected. On the contrary, the R-Term method depends on a counter to release the suspended packet transmission. The counter-based waiting process eventually makes

the network halt. This is why the overall switching activity for the R-Term method is low for this case, even though this method provides some protection against TAIL attacks.

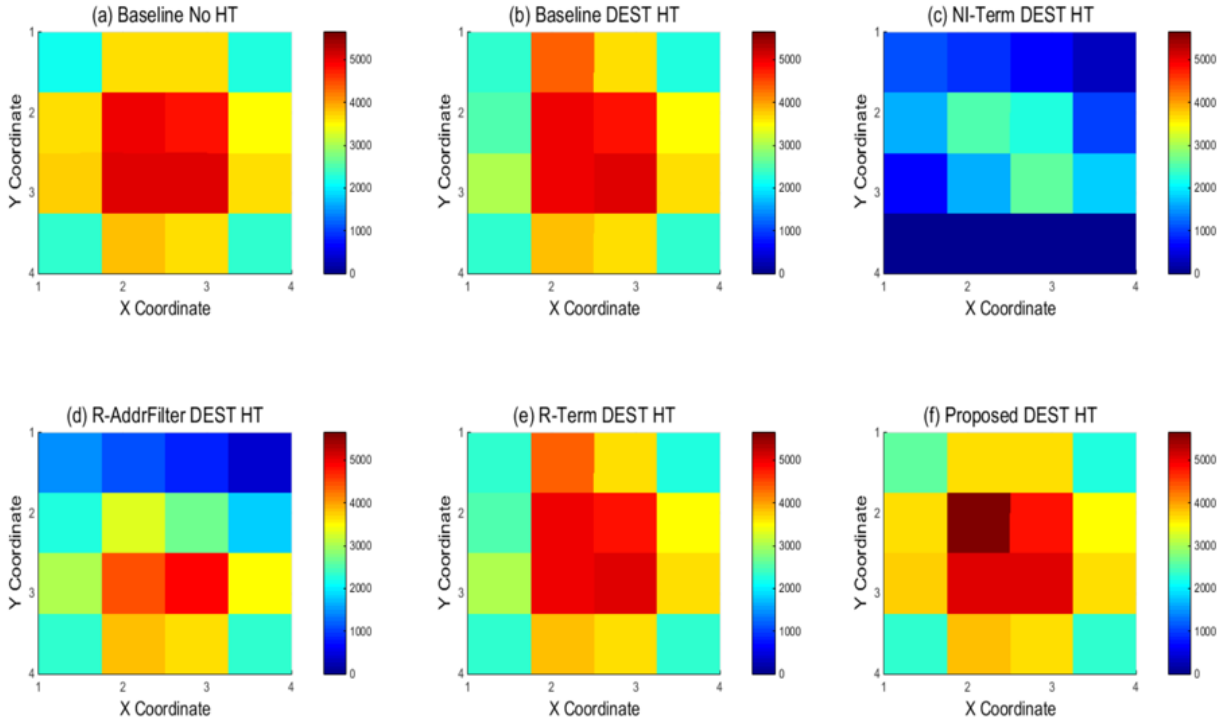


Figure 5.8. Traffic hotspot migration and bandwidth depletion induced by DEST HTs.

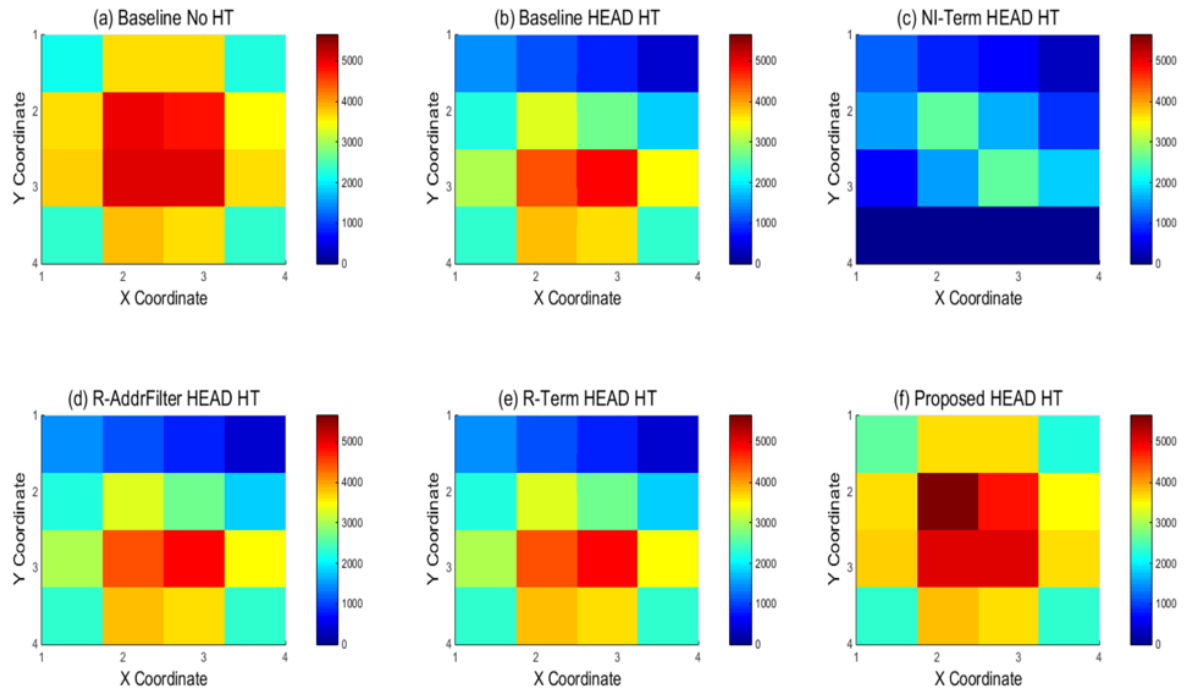


Figure 5.9. Traffic hotspot migration and bandwidth depletion induced by HEAD HTs.

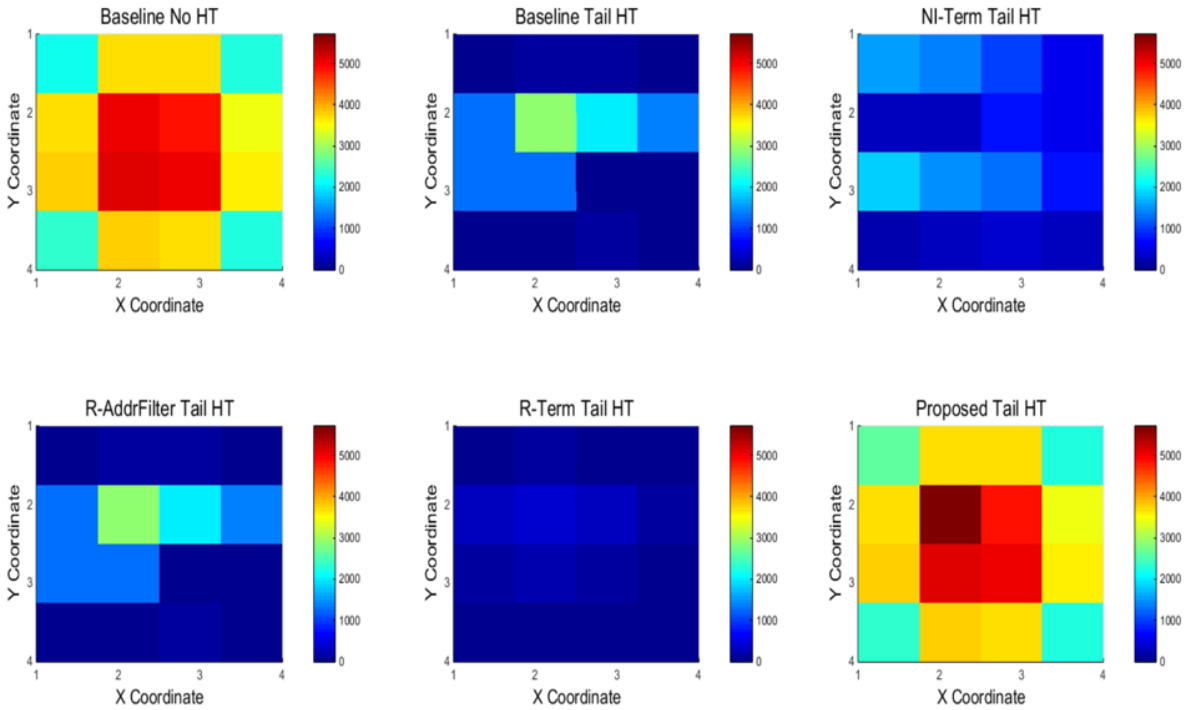


Figure 5.10. Traffic hotspot migration and bandwidth depletion induced by TAIL HTs.

5.3.4. Effective Average Packet Latency

The packet latency is the time interval from the moment when the packet is completely injected into the network to the time when the tail flit of that packet arrives in the packet destination. Figure 5.11 shows the average packet latency of the proposed method, which does not increase with an increasing number of DEST HTs. When the traffic injection rate increases, the average latency variation is within only 2.2%.

As some HTs in the other methods can make designs have erratic latency results (for instance, infinite latency), the average latency would become meaningless. Therefore, an effective average latency metric is used to compare the latency performance among different methods. As expressed in Eq. 5.2, the effective average latency (i.e. $Avg.\ Latency_{Effective}$) is the product of the real average latency for all valid packets and the ratio of the valid packets received by the baseline without HTs over those collected by the method under test.

$$Avg.Latency_{Effective} = Avg.Latency * \frac{Valid\ Packets_{Baseline\ w/o\ HT}}{Valid\ Packets_{method\ under\ test}} \qquad \text{Equation 5.2}$$

Using this effective latency, we are able to consider the impact of HTs on latency fairly, given that some methods will never receive the packets affected by HTs. As shown in Fig. 5.12, the proposed method improves the effective average latency by up to 63.4%, 68.2%, and 98.9% over the existing methods for the single DEST HT, single HEAD HT, and single TAIL HT scenarios, respectively.

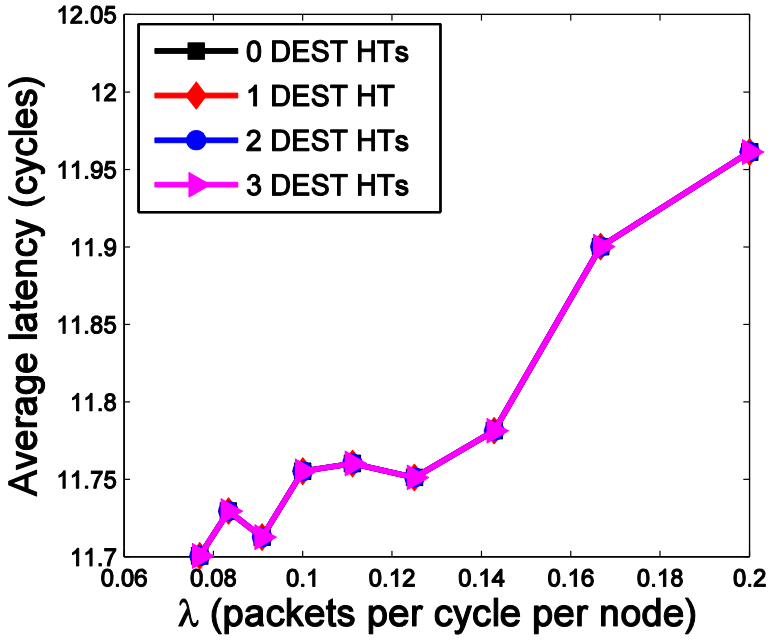


Figure 5.11. Average packet latency for a wide range of traffic injection rate and different number of HTs.

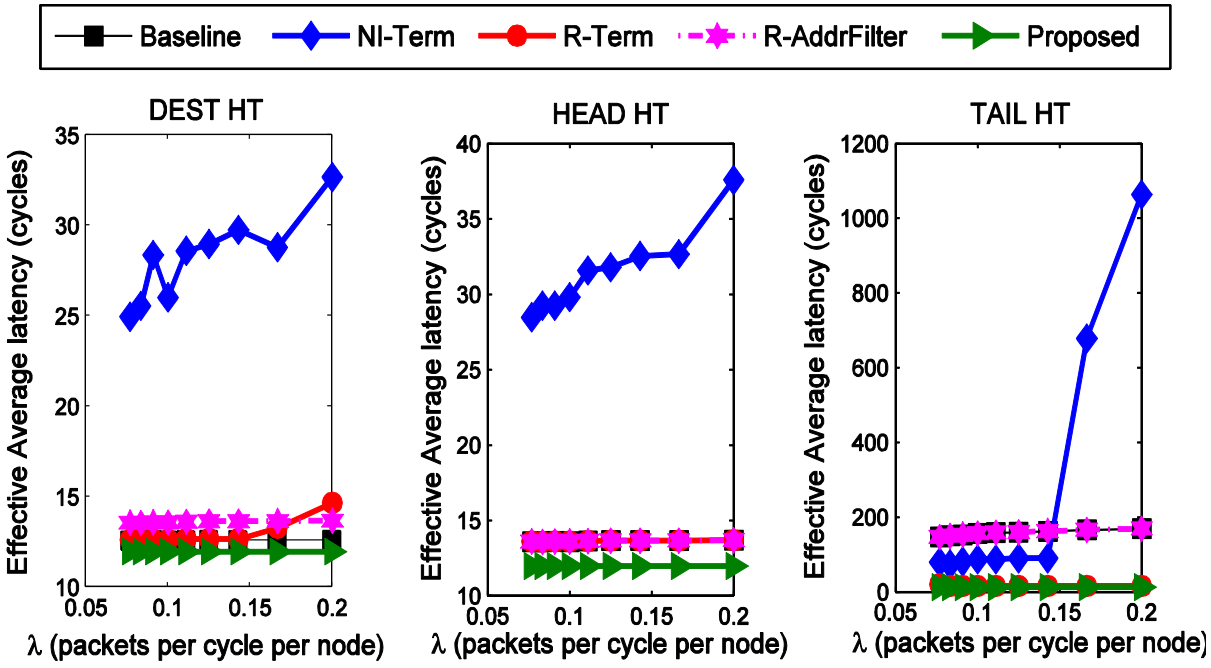


Figure 5.12. Effective average packet latency.

5.4. Probability of HT Attack Success (P_{AS})

5.4.1. P_{AS} for the Case of Countermeasure Unknown

The probability of attack success (P_{AS}) was used as a metric to evaluate the obfuscation efficiency of the proposed bit permuting mechanism. P_{AS} is defined as the ratio of the number of cases that the hardware Trojan successfully selects the bits belonging to the flit field of interest over the total number of test cases. If the attackers do not have sufficient knowledge of the specific permutation configuration, they will simply overwrite the interested field at the position that they blindly guessed.

Figure 5.13 shows the P_{AS} achieved by the proposed permutation method against tail bit attacks for NoCs with flit sizes of 32-bits, 64-bits and 128-bits. Without the permutation, it is assumed that the hardware Trojan can always successfully control the tail bit, i.e. P_{AS} will be 1. Since the permutation configuration is dynamically chosen by the key vector, in the proposed method P_{AS} is less than 1, as shown in Fig. 5.13. The impact of the flit width on the P_{AS} for the tail bit attack. Simulation results show that the average P_{AS} of the proposed method are 0.0206, 0.0104, and 0.0053 for 32-bit flits, 64-bit flits, and 128-bit flits, respectively.

Figure 5.14 shows the probability of *partially* or *completely* identifying and replacing the exact flit-field positions with intended malicious content. The flit field of interest has a binary vector that may have overlapped bits with the malicious intention bits. Even if the adversary does not correctly guess all of the target bits, they still may change the target field to a different malicious vector. As shown in Fig. 5.14, the attacker can successfully perform an attack by changing one of the destination address bits with a probability of 82.2%. The successful change of 3 destination address bits occurs with probability of 1.37%, which is over one order of magnitude lower than the 1-bit cases. The probability for 6-bit changes on the destination address is zero among one million

test cases. Figure 5.14 indicates that the proposed error control code can obtain a P_{AS} below 10^{-6} even if a 5-bit malicious vector is injected by the attacker.

5.4.2. P_{AS} for When the Adversary Knows the Countermeasure

The goal of the HT countermeasures in [36, 38, 39] and the proposed approaches is to minimize the HT attack success rate (P_{AS}). A lower P_{AS} means a more secure design. Previous work [36, 38, 39] assumes that the attacker does not have the knowledge of the HT countermeasure mechanisms used in the design. Unfortunately, this assumption is not always valid if the attacker has access to the design details. The methods in [36, 38, 39] yield a P_{AS} of 1 when the applied HT countermeasure is known to the adversary. This is because the adversary may be able to change the restricted memory range and mute the counter for HT detection. In contrast, the proposed method assumes that the adversary may know the bit permuting mechanism in the design, but they do not know the exact key vector. This is because the key vector is uniquely generated at each router at run-time. As shown in Fig. 5.15, PAS significantly decreases with the increasing number of permutation configurations used. The nonlinear trend line is predicted by a power function y=0.4887x-1.402. As predicted, if the number of permutation configurations available in the router is equal to 16, the proposed method achieves a P_{AS} 1.04*10-2 in the case that the countermeasure is known to the adversary. The proposed method P_{AS} is two orders of magnitude less than that of previous work [36, 38, 39].

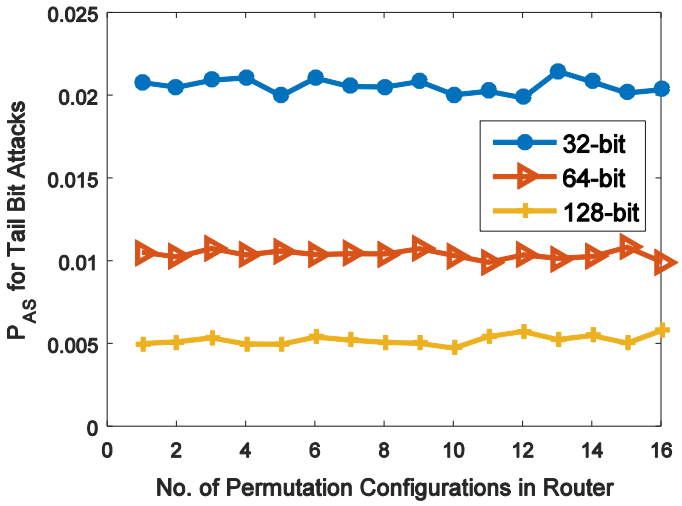


Figure 5.13. Success rate PAS of tail bit attacks for different number of permutation configurations available in a router.

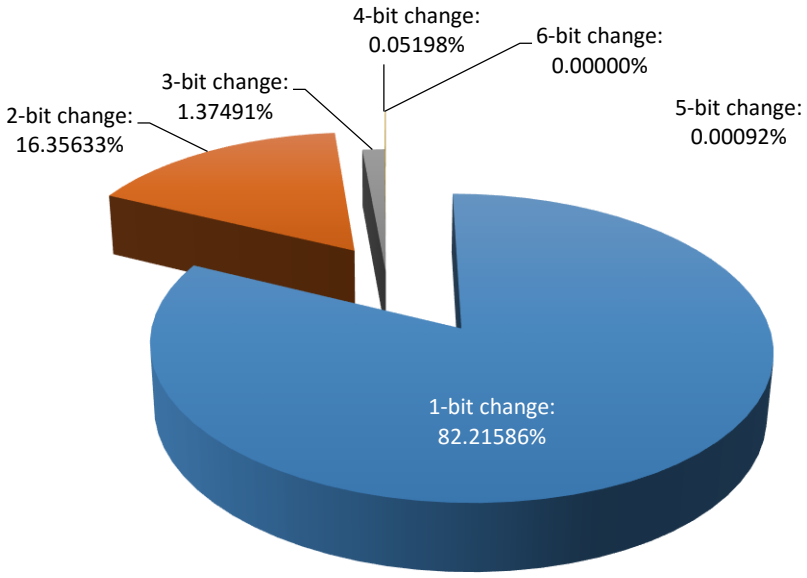


Figure 5.14. Statistical pie chart for the number of valid bits that an adversary modifies on destination address to perform a successful attack.

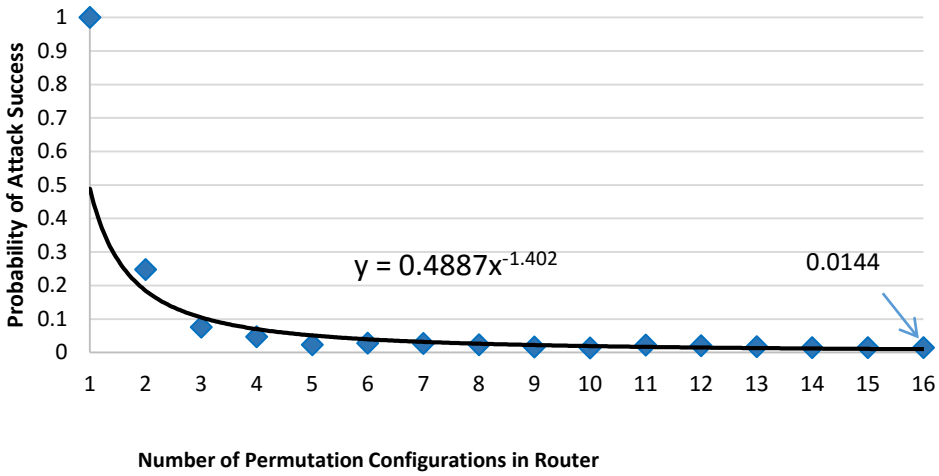


Figure 5.15. Probability of attack success PAS of the proposed method if the adversary knows the permutation configurations applied in the router.

5.5. Randomness of Challenge Bits

The randomness of the dynamic permutation configuration is achieved by the following two factors: (1) the PUF structure implemented in each router produces different select bits due to process variations, and (2) the challenge bits from the round-robin table depend on the real-time traffic injection rate and traffic pattern. As a result, the selection of the dynamic permutation configuration is unknown at the design time. In each NoC router, eight configurations of dynamic bit permutation were implemented. Figure 5.16 shows the occurrences of different PUF challenge vectors over 21,000 clock cycles. As can be seen, each router demonstrates a unique characteristic of the occurrence frequency for the permutation configurations. Figure 5.16 clearly indicates that the round-robin table based dynamic permutation can make the permutation configuration unpredictable, thus raising the bar for the attacker to insert a meaningful hardware Trojan.

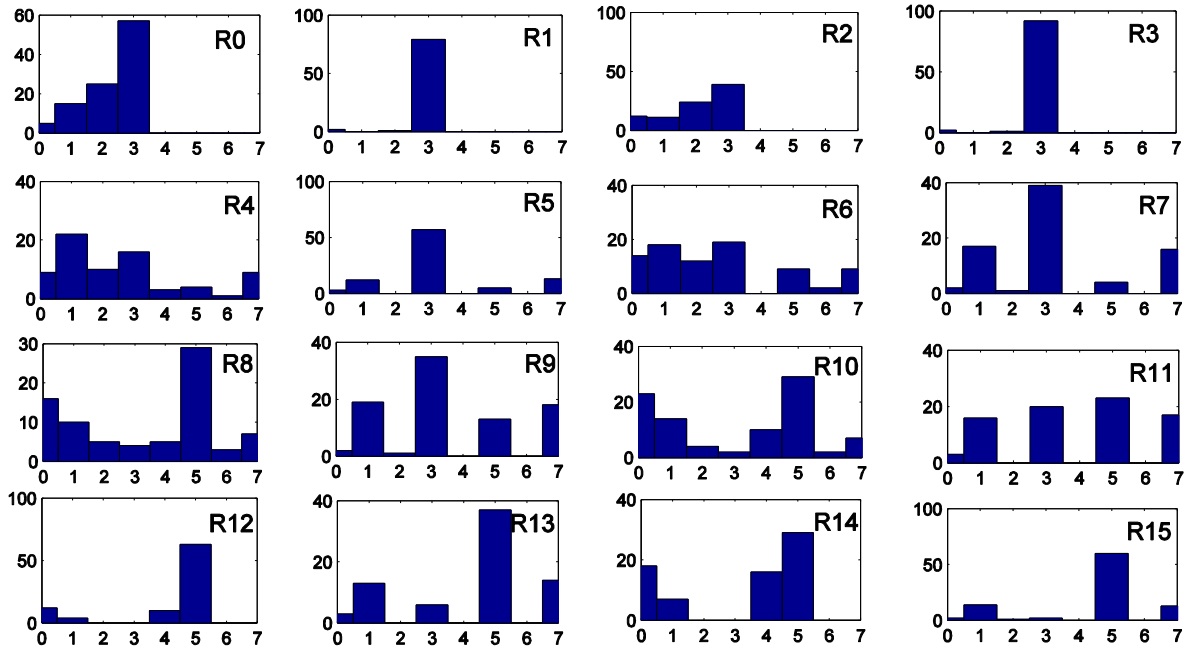


Figure 5.16. Occurrence numbers of PUF challenge vectors for different NoC routers. The X-axis is the decimal number of each dynamic permutation configuration ID. The Y-axis is the number of occurrence times of that challenge vector in the total simulation period.

The traffic traces for Figure 5.17 were the secure and non-secure zone cases shown in Fig. 5.1, with a traffic injection rate of 0.143 packets per cycle per node. The impact of different traffic injection rates and traffic traces on the occurrences of different challenge vectors were also examined. As shown in Fig. 5.17, different traffic injection rates for the same traffic trace can lead to significant differences on the occurrence frequency of different challenge vectors. For instance, the configurations 0 and 5 yield -73.9% and +141.4% occurrence changes, respectively, when the traffic injection rate λ varies from 0.143 to 0.2 packets per cycle per node. As the uniform traffic trace is injected into the network, the frequencies of some challenge vectors (e.g. 1 and 7) are more than doubled.

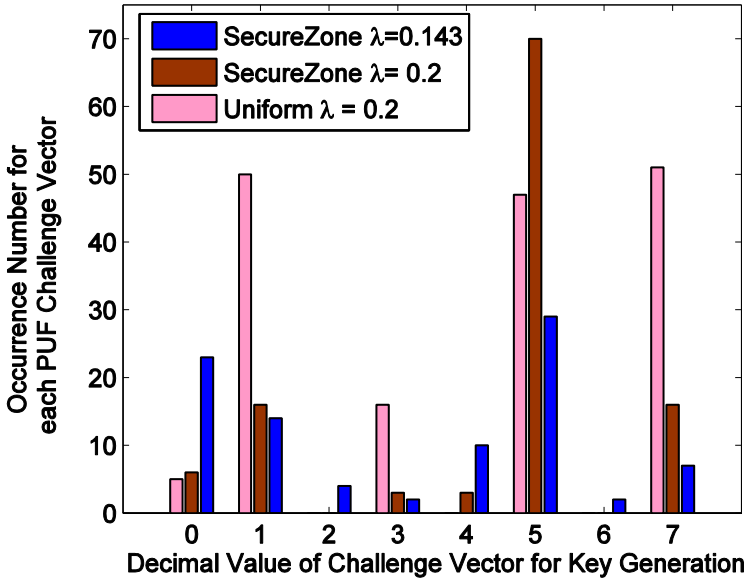


Figure 5.17. Impact of different traffic injection rates and traffic traces on the occurrence of different PUF challenge vectors for the dynamic permutation configuration.

5.6. Area Overhead/FPGA Synthesis

Each of the five designs examined here was synthesized to the FPGA seen in Fig. 3.21. The resource utilization (which relates to the area of the device) was measured for each design, and can be seen in Table 5.1. The biggest impact on area most likely comes from the permuting mechanisms, and the decoders.

Table 5.1. Device Resource Utilization.

Design	FPGA Resource	FPGA Utilization Rate
Baseline	Slice LUTs	42.77%
	Slice Registers	16.99%
NI-Term	Slice LUTs	52.00%
	Slice Registers	17.37%
R-Term	Slice LUTs	48.59%
	Slice Registers	18.03%
R-AddrFilter	Slice LUTs	57.24%
	Slice Registers	17.80%
Proposed	Slice LUTs	65.74%
	Slice Registers	19.11%

5.7. Chapter Summary

Results obtained through Verilog and MATLAB simulations were presented in this chapter. The Verilog results were compared between the proposed new method and four other methods. It was found that the proposed method outperformed the other methods when HTs were introduced into the system.

Chapter 6. AES Power Obfuscation with Dynamic Permutation

6.1. Introduction

In an Internet of Things (IoT) application, sensors, actuators, and simple data collection devices are connected through the internet. Among those connected devices, at least one data computation chip is needed to analyze the massive amount of data and make a corresponding decision. Due to the important role that the computation chip plays, it is assumed that some cryptographic algorithm (at least a lightweight one) will be applied to that chip. It is proposed to exploit the dynamic permutation method, which was presented in Chapter 4, to protect the processing unit from hardware attacks, more specifically hardware Trojan insertions and side-channel attacks simultaneously.

Dynamic permutation changes the original order of the information received from the sensor. As the processing unit handles a dynamically permuted message, the attacker cannot precisely execute a hardware Trojan attack at a pre-defined condition. Multiple possibilities of permutation patterns increase the search space for the attacker to successfully reverse engineer the processing unit, or execute a meaningful attack. Even if the attacker has certain knowledge of the processing unit design, the dynamic aspect of the permutation method makes it harder to precisely launch an attack. A random number generator or the cryptographic module can control the dynamicity. If the cryptographic module detects an invalid message, a new permutation configuration will be requested. With this framework, the processing node is obfuscated by the dynamic feature.

6.2. Experimental Setup

Data collection and transfer devices are also typical IoT devices. As a case study, we use a router for a Network-on-Chip (NoC) is used to explain in detail how to design the dynamic permutation to thwart hardware Trojan insertions and also make side-channel attacks harder to execute.

The main function of a router is to interpret the header information of a data packet from other NoC nodes and then direct that packet to the next routing hop. In a generic NoC router, each pair of input and output ports is composed of input and output FIFOs, FIFO controllers, a crossbar, and an arbiter. As shown in Fig. 4.1, the proposed dynamic bit permutation is placed before the input FIFO, while the dynamic bit de-permutation is appended after the output FIFO. To detect whether any hardware Trojan changes the packet integrity, an error control coding method can be applied to the packet. The route computation is executed only when the packet integrity is sustained.

The critical component in this router design is the permutation configuration module. The dynamic aspect of the permutation method is achieved by a random number generator followed with modulo operation, as opposed to the PUF method used in chapter 4. It is assumed that the router has a unique identifier, R_{ID}, and the dynamic selection bits for the permutation unit are defined as DP_{SEL}. A random DP_{SEL} is obtained through Eq. 6.1 below.

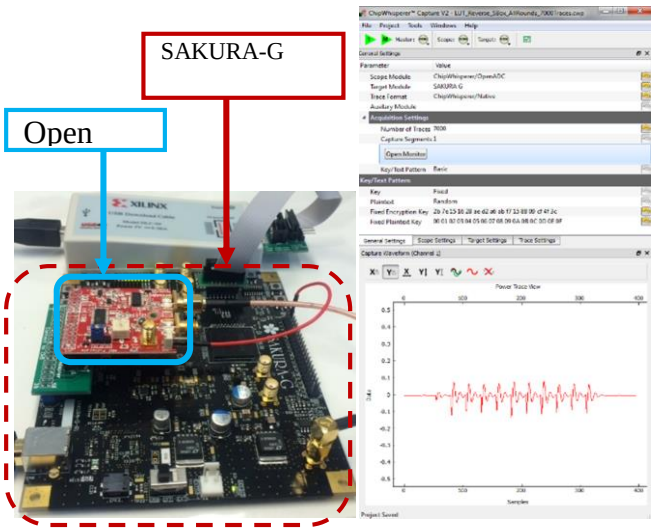
$$\mathrm{DPSEL}\,(\mathrm{t}) \;=\; \mathrm{RNG}\,(\mathrm{t})\;\%\;(\beta \;*\; \mathrm{RID}) \qquad \text{Equation 6.1}$$

where *RNG(t)* is the random number generated at the moment *t*, which typically is an integer times the minimum clock period. The symbol '%' stands for modulo operation. If the range of the random

number generator is sufficiently large (compared with R_{ID}), the choice of DP_{SEL} is evenly distributed in the range of [0, β * R_{ID}-1]. The parameter β is the bit width ratio between DP_{SEL} and R_{ID}. Normally, the DP_{SEL} is updated after a given period. Once the alarm signal from a cryptographic module arrives, the DP_{SEL} is promptly modified to mitigate the hardware Trojan attack. As a hardware Trojan is triggered by a specific condition, a new DP_{SEL} is very likely to lead that Trojan triggering condition to vanish.

6.3. Results

An advanced encryption standard (AES) module with a 128-bit key and the proposed router were implemented in Verilog HDL. The HDL codes were synthesized in Xilinx ISE 14.6 and then downloaded to the SAKURA-G FPGA board, as shown in Fig. 6.1. With the assistance of Python-based *ChipWhisperer* [85] software, the power traces were captured for side-channel analysis attacks.



(a) (b)

Figure 6.1. Experimental setup for SCA attack. (a) SAKURA-G board with an OpenADC mounted. (b) ChipWhisperer interface capturing one power trace.

6.3.1. Crypto Key Retrieval Speed

The goal of the power-based side-channel attack is to retrieve the cipher key used in the cryptographic module. The leak of a cipher key leads to the loss of security assurance provided by the cryptographic module. As dynamic permutation potentially changes the power consumption profile for the processing unit, the proposed method makes it more difficult to retrieve the key. The increased difficulty is due to the obfuscation of the power waveform due to the dynamically changing permutation configuration. The average partial guessing entropy (APGE) is used as a metric to evaluate the speed of the cipher key retrieval [80]. It is assumed that all subkeys are retrieved when APGE falls below 10 [81].

6.3.1.1. Impact of Processing Unit Area on Speed of Key Retrieval

A side-channel analysis attack is able to retrieve the crypto key if one has a sufficient number of power traces. As shown in Fig. 6.2(a), the APGE for the AES reaches below 6, and this indicates that all of the subkeys for AES have been found. The combination of the AES module and a processing unit, router or NoC in our experiment results in the noticeable increase in the corresponding APGE of the processing unit with AES protection. The baseline router and the corresponding NoC are for a generic 4x4 mesh NoC. The proposed method is built in the generic NoC with dynamic permutation and its run-time configuration unit. Fig. 6.2(a) shows that the proposed processing unit design increases the APGE by factors of 82.3, 5.8, and 3.6 over AES, AES-Router-Baseline, and AES-NoC-Baseline, respectively. This means, if the proposed method is applied, more power traces are needed to find the complete key vector. This can also be interpreted as the proposed method makes it more difficult to retrieve the subkey bytes through correlation power analysis (CPA). Fig. 6.2(b) shows the speed of subkeys retrieval over the number

of the power traces. As can be seen, the proposed method yields fewer subkey bytes than the AES one given 7000 power traces.

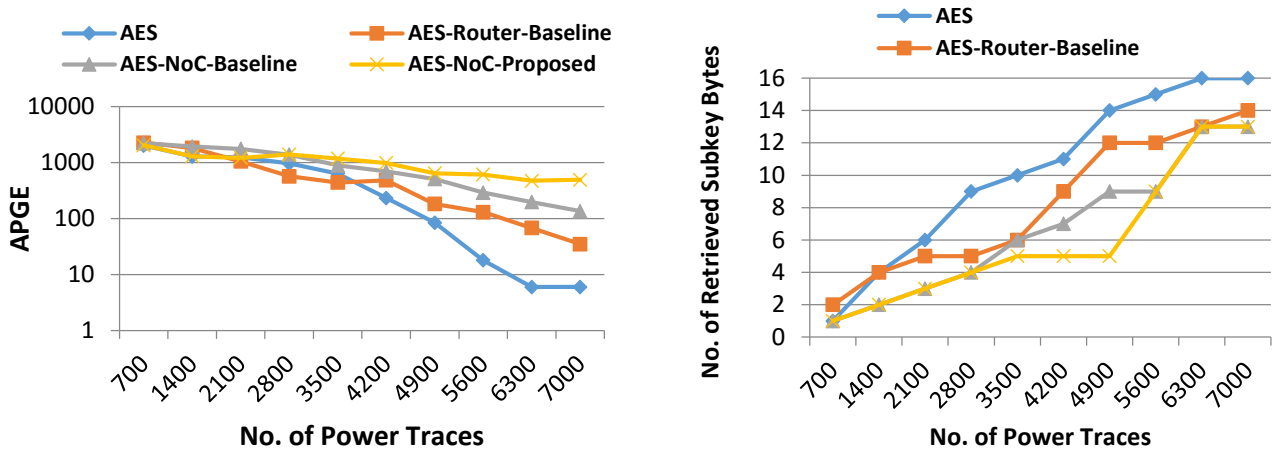


Figure 6.2. Impact of the size of processing unit on the key retrieving speed. (a) APGE and (b) the number of retrieved subkey bytes.

6.3.1.2. Impact of Selection Randomness on the Speed of Retrieving Keys

To reduce the hardware cost, a modular counter (CNT) was implemented to mimic RNG(t). The result of CNT(t) % ($\beta * R_{ID}$) allows for some randomness in the run-time selection of the bit permutation configuration. As the width of R_{ID} is as same as DP_{SEL} in this experiment, β is equal to 1. The results in this subsection are based on the processing unit including an AES module and a proposed router. The width of the modular counter was varied from 0 to 10 bits. The 0-bit CNT means a fixed permutation pattern. A modular counter with more width generates a larger data range, and thus more random options for permutation.

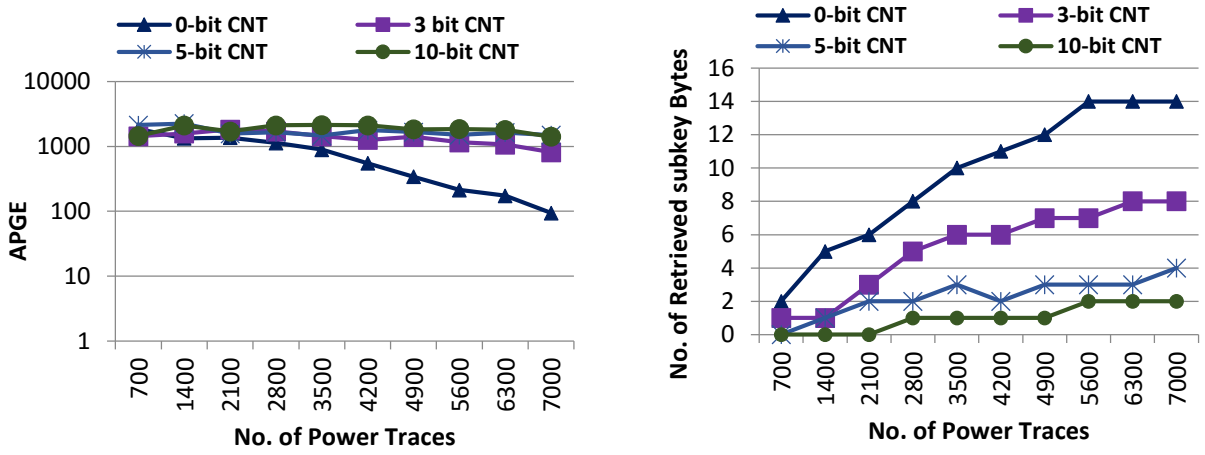


Figure 6.3. Impact of the selection randomness on the key retrieving speed. (a) APGE and (b) the number of retrieved subkey bytes.

As shown in Fig. 6.3(a), the 10-bit counter successfully makes the crypto key almost irretrievable, as the APGE remains nearly flat with the increasing number of power traces. The number of retrieved subkey bytes, shown in Fig. 6.3(b), also confirms that only two bytes of subkey are found for the 10-bit CNT case. However, the obtained benefit on key retrieval does infinitely increase with the size of the counter. As shown in Fig. 6.3(a), the variation of APGE for 3- to 10-bit CNT cases is not significant. This indicates that a small variation on the selection of different permutation patterns can effectively thwart power-based side-channel attacks.

6.3.2. Area Overhead (FPGA Results)

The hardware cost of the XC6SL75 FPGA implementation is shown in Table 6.1. The AES-Router-Proposed (i.e. static permutation) method increases the number of occupied slices by 33.9% over the AES-Router-Baseline. If one introduces a 5-bit counter to dynamically permute the bit order, this method costs 88.5% more FPGA area. For security-critical application, this amount of hardware overhead is acceptable.

Table 6.1. Hardware Cost in FPGA Implementation.

Design	AES	AES-Router-Baseline	AES-NoC-Baseline	AES-NoC-Proposed	AES-Router-Proposed	AES-Router-Proposed-3-bit CNT	AES-Router-Proposed-5-bit CNT	AES-Router-Proposed-10-bit CNT
Number of Slice Registers	769	1184	5,620	11,263	1,497	2,059	2,688	2,656
Number of Slice LUTs	2663	3584	14,182	22,201	4,546	4,583	5,990	6,544
Number of Occupied Slices	828	1034	4,934	6,912	1,385	1,366	1,949	1,957

6.4. Chapter Summary

To protect the processing unit in AES applications used for the IoT, it was proposed to use the collaborative method presented in chapter 4 to provide obfuscation of the data being processed. It was seen that the proposed method can effectively help protect the processing unit from HT insertions and side-channel attacks simultaneously. The dynamic aspect of the permutation method helped protect the processing units by delaying the time it takes to retrieve a given key for AES by increasing the number of power traces needed to do so.

Chapter 7. Discussion

7.1. Summary

The globalization of the business model used by the IC industry has helped reduce the costs of supply. Outsourcing of the fabrication, assembly, and manufacturing stages have proved to reduce the production cost of chips, but have also introduced several security issues in electronic hardware. Hardware attacks that aim to change design functionality, steal design secrets, and cause information leakage have all been introduced as a result of the globalized business model.

A prevalent type of hardware attack that was introduced directly from outsourcing, is that of the Hardware Trojan. General methods exist to detect the presence of Hardware Trojans in designs at test time, but with the complexity of these malicious circuits increasing, it is becoming harder to detect them with conventional methods. Although general detection methods exist, there is a lack of detection methods for on-chip networks. Trojans in on-chip networks can cause Denial of Service, information leakage, and design trade secret leakage. To thwart Hardware Trojans in Networks-on-Chip, a run-time method, as opposed to conventional offline methods, is proposed and provides active mitigation of Hardware Trojans that cause Denial of Service attacks.

In chapter 4, a novel method to mitigate residual HT attacks is introduced. This method is different than existing offline approaches as it is assumed that residual HTs have already bypassed the offline detection methods. The basis of the proposed method is that it provides integrity to packets traveling over the NoC, and it also lowers the chance that an attack affects the targeted area. To provide packet integrity, Error Control Coding, in the form of a Hamming Code, is used not only to detect errors present in the packet flits, but to also correct vital instructions that may have been maliciously changed. To lower the probability of a successful attack, a bit permutation method

is proposed that rearranges the bits going into a NoC router. Within the router, extra hardware is added so instructions can still be interpreted correctly. A collaborative method combines the prior two methods to detect multiple bit errors and trigger a change in the bit arrangement to effectively spread out the multiple bit error.

In chapter 5, the proposed method is compared to four other methods that aim to increase the security of NoCs. Each of these methods is from recent IEEE literature. Results are presented that compare the packet throughput, effective average latency, link availability, and link usage of the five different methods. For packet throughput, the number of valid packets received by a particular design is compared with that of the others. A *valid packet*, is defined as one that is received at the target IP core to which it was originally sent. Any other packet that arrives at a local port that is illegal or incorrect with regard to the original destination is considered an *invalid packet*. The results presented in 5.3.1 show that the proposed method outperforms the other methods with regard to throughput, providing an increase of 6.4-43.12%, 11.6-43.67%, and up to 98.8% for DEST, HEAD, and TAIL HTs respectively. The interval of an increase in throughput depends on the number of HTs that were injected into the system.

For link availability, the amount of data on the NoC links between routers was measured at every clock cycle of the simulation time. This enabled a view of the amount of traffic on the links at a given simulation time, and this variable was plotted this data over the entire simulation time in the form of histogram plots, seen in section 5.3.2. It was seen that that the proposed method outperforms the rest for link availability, and an in-depth analysis of the improvement over the other methods can be found in section 5.3.2.

Results for link usage, which can be found in section 5.3.3, show of a graphical representation of the traffic flow in the network over the entire simulation time. Migrations of hot

spots in the network, compared to an ideal “no HT” model, mean that the network has been rerouting traffic the wrong way due to HTs that affect the destination address of a header flit. Cold spot migration shows that a zone of the network is undergoing depleted bandwidth, due to extra flits probably missing their header flit. Dropping the tail flit will also cause bandwidth depletion, but in this case the depletion is due to paths not being freed up to other packets requesting permission to use that path. It was seen that the proposed method exhibits the most similar plot to the ideal case.

In section 5.3.4 the effective latency was examined for each HT mitigation method. The proposed method achieved an average effective latency between 63.4-98.9% better (smaller) than the existing approaches.

In chapter 6, the proposed collaborative method was exploited to protect the processing units in an AES application against HT insertions and side-channel attacks simultaneously. The permutation method, and the dynamicity provided by it, helps to obfuscate the data within the processing unit. This will help delay the time it takes to retrieve a given key for AES by increasing the amount of power traces needed.

The limitations of the proposed method lies in its area overhead. In Tables 5.1 and 6.1, it is shown that the proposed method incurs the largest area overhead of the designs with which it was compared. This is most likely due to the large networks of MUXes that comprise the permuting modules, as well as the increase in internal bit widths in routers for the ECC parity bits. The increase in bit width also means that extra registers are needed in the FIFOs, and other modules require extra hardware to process and send these bits through the router.

7.2. Other work

While working at the Reliable VLSI systems lab at UNH, several other papers were worked on, including the strengthening of crypto-systems by creating a more robust fault detection method, and by using an emerging technology (memristors), in place of traditional CMOS gates.

7.2.1. Crypto-system Security

Security is a crucial concern for the development of trustworthy embedded systems. Cryptographic algorithms are commonly used to ascertain security of the system. Driven by malicious intent, attackers are impelled to extract the cipher key and thus compromise the embedded system. As embedded systems are typically hardware constrained and are used for critical and potentially sensitive contexts, lightweight ciphers are preferred for security. Hence it is imperative to explore effective fault-detection methods for lightweight ciphers.

We propose a new microarchitecture to thwart fault attacks that place symmetric faults on the two encryption data paths. To further reduce the FBR for SIMON, we propose a new fault-detection algorithm that integrates operand permutation and masking techniques. Closed-form expressions for de-permutation and de-masking in SIMON are provided in this letter. Our method is assessed under two fault attack scenarios (random and symmetric fault injections) with bit-flip, stuck-at-0, and stuck-at-1 fault models. Simulation results show that our method minimizes the FBR to zero with the fault attack scenarios of symmetric fault location and stuck-at-0 fault injections. Overall, the proposed method outperforms the existing fault-detection methods in multiple fault attack conditions, at the cost of 5% more area overhead than the most hardware-efficient fault detection method.

7.2.2. Memristor-based SIMON Cipher

The fourth fundamental circuit element, memristor, attracts increasing attention because of its memory characteristic. The special memory behavior of the memristor has been exploited to design control systems, memory arrays, logic gate, and security primitives in previous work. However, the power characteristics of the memristor have not been widely studied yet. In this work, we used a memristor model that is suitable for circuit simulation to investigate the power characteristics of the memristor itself and memristor-based logic gates. Our simulation results indicate that memristor has different power characteristic compared with CMOS devices. The peak power of memristor-based gates does not monotonically increases with input voltage amplitude; instead, the combination of input period length and voltage amplitude determines the occurrence of power peak. The reason is that the power consumption of memristors depends on the effective memristor width, which is controlled by the input. We further examine the feasibility of utilizing memristors to implement a new block cipher, SIMON. Our studies show that the unique power characteristic of memristor based SIMON may add extra challenges for extraction of the secret key from the cipher as it introduces 94% power deviation while power sampling.

References

[1] S. King, et al. “Designing and implementing malicious hardware,” in *Proc. the 1st Usenix Workshop on Large-Scale Exploits and Emergent Threats (LEE’08)*, Article 5, 2008.

[2] N. Potlapally, “Hardware Security in Practice: Challenges and Opportunities”, in *Proc. HOST’11*, pp. 93–98, 2011.

[3] R. Chakraborty, S. Narasimhan and S. Bhunia, “Hardware Trojan: Threats and emerging solutions,” in *Proc. HLDTV*, pp. 166–171. 2009.

[4] R. Karri, J. Rajendran, K. Rosenfeld and M. Tehranipoor, “Trustworthy Hardware: Identifying and Classifying Hardware Trojans,” *IEEE Computer*, vol. 43, no.10, pp. 39–46, Oct. 2010.

[5] H. Liu, H. Luo, and L. Wang, “Design of Hardware Trojan Horse Based on Counter,” in *Proc. ICQR2MSE’11*, pp. 1007–1009, June 2011.

[6] X. Wang, M. Tehranipoor, J. Plusquellic, “Detecting malicious inclusions in secure hardware: Challenges and solutions,” in *Proc. HOST’08*, pp. 15–19, 2008.

[7] J.-P. Diguet, et al., “NOC-centric Security of Reconfigurable SoC,” in *Proc. NOCS’07*, pp. 223–232, May 2007.

[8] H. Wassel, et al. “Networks on chip with provable security properties,” *IEEE Micro*, vol. 34, no. 3, pp. 57–68, 2014.

[9] Y. Jin, N. Kupp and Y. Makris, “Experiences in Hardware Trojan design and implementation,” in *Proc. HOST’09*, pp. 50–57, Jul. 2009.

[10] S. Adee, [online] The hunt for the kill switch, http://www.spectrum.ieee.org.

[11] X. Zhang and M. Tehranipoor, "Case study: Detecting Hardware Trojans in third-party digital IP cores," in *Proc. HOST'11*, pp. 67–70, June 2011.

[12] M.Tehranipoor and F. Koushanfar, "A Survey of Hardware Trojan Taxonomy and Detection," *IEEE Design & Test of Computers*, vol. 27, no. 1, pp. 10–25, Jan.-Feb. 2010.

[13] H. Salmani, M. Tehranipoor, and J. Plusquellic, "New Design Strategy for Improving Hardware Trojan Detection and Reducing Trojan Activation Time," in *Proc. HOST'09*, pp. 66–73, July 2009.

[14] S. Bhunia, et al, "Hardware Trojan Attacks: Threat Analysis and Countermeasures," in *Proc. of the IEEE*, vol.102, no. 8, pp. 1229–1247, July 2014.

[15] J. Rajendran, O. Sinanoglu, and R. Karri, "Regaining Trust in VLSI Design: Design-for-Trust Techniques," in *Proc. of the IEEE*, vol. 102, no. 8, pp. 1266–1282, July 2014.

[16] M. Banga and M. Hsiao, "VITAMIN: voltage inversion technique to ascertain malicous insertions in ICs," in *Proc. HOST'09*, pp. 104–107, 2009.

[17] S. Narasimhan, et al., "Improving IC Security Against Trojan Attacks Through Integration of Security Monitors," *IEEE Design & Test of Computers*, vol. 29, no. 5, pp.37–46, Oct. 2012.

[18] Y. Jin, and Y. Makris, "Hardware trojan detection using path delay fingerprint," in *Proc. HOST'08*, pp. 51–57, 2008.

[19] Y. Jin, and Y. Makris, "Hardware trojans in wireless cryptographic integrated circuits," *IEEE Design Test of Computers*, vol. PP, no. 99, 2013.

[20] J. Rajendran, E. Gavas, J. Jimenez, V. Padman, R. Karri, "Towards a comprehensive and systematic classification of hardware trojans," in *Proc. ISCAS'10*, pp. 1871–1874, 2010.

[21] L. Lin, W. Burleson, and C. Paar, “Moles: malicious off-chip leakage enabled by sidechannels,” in *Proc. ICCAD ’09*, pp. 117–122, 2009.

[22] R. Torrance and D. James, “The State-of-the-Art in IC Reverse Engineering,” in *Proc. CHES*, vol. 5747 of LNCS, pp. 363–381, Sept. 2009.

[23] D. Agrawal, S. Baktir, D. Karakoyunlu, P. Rohatgi, and B. Sunar, “Trojan detectionusing ic fingerprinting,” in *Proc. IEEE Symposium on Security and Privacy*, pp. 296–310, 2007.

[24] S. Vangal, et al., “An 80-tile sub-100-W TeraFLOPS processor in 65-nm CMOS,” *IEEE Journal of Solid-State Circuits*, vol. 43, no. 1, pp.29–41, 2008.

[25] W. J. Dally and B. Towles, “Route packets, not wires: On-chip interconnection networks,” in *Proc. DAC’01*, pp. 684–689, Jun. 2001.

[26] L. Benini and G. De Micheli, “Networks on Chips: A new SoC paradigm,” *Computer*, pp. 70–78, Jan. 2002.

[27] R. S. Chakraborty, and S. Bhunia, “HARPOON: An Obfuscation-Based SoC Design Methodology for Hardware Protection,” *IEEE TCAD*, vol. 28, no. 10, pp. 1493–1502, 2009.

[28] J. Rajendran, Y. Pino, O. Sinanoglu, and R. Karri, “Logic encryption: a fault analysis perspective,” in *Proc. DATE'12*, pp. 953–958, 2012.

[29] G. K. Contreras, M. T. Rahman and M. Tehranipoor, “Secure Split-Test for Preventing IC Piracy by Untrusted Foundry and Assembly”, in *Proc. DFT’13*, pp. 196–203, 2013.

[30] H. Wassel, Y. Gao, J. Oberg, T. Huffmire, R. Kasterner, F. Chong, and T. Sherwood, “SurfNoC: A low latency and provably non-intetfering approach to secure Networks-on-Chip, “ in *Proc. ISCA’13*, pp.583–594, 2013.

[31] L. Fiorin, S. Lukovic, G. Palermo, “Implementation of a reconfigurable data protection module for NoC-based MPSoCs,” in *Proc. IEEE. Intl. Symp. Parallel and Distributed Processing*, pp. 1–8, Apr. 2008.

[32] J. Porquet, A. Greiner, and C. Schwarz, “NoC-MPU: a secure architecture for flexible co-hosting on shared memory MPSoCs,” in *Proc. DATE’11*, pp. 1–4, 2011.

[33] J. Sepulveda, et al. “Hybrid-on-chip communication architecture for dynamic MP-SoC protection,” in *Proc. SBCCI*, pp. 1–6, 2012.

[34] L. Fiorin, G. Palermo, S. Lukovic, V. Catalano and C. Silvano, “Secure Memory Accesses on Networks-on-Chip,” *IEEE Trans. on Computers*, vol. 57, no. 9, pp. 1216–1229, Sept. 2008.

[35] L. Fiorin, G. Palermo, S. Lukovic, and C. Silvano, “A Data Protection Unit for NoC-Based Architectures,” in *Proc. CODES+ISSS*, pp. 167–172, Sept.-Oct. 2007.

[36] S. Baron, M.Wangham, C. Zeferino, “Security mechanisms to improve the availability of a Network-on-chip,” in *Proc. ICECS’13*, pp. 609–612, 2013.

[37] G. Gebotys, and R. Gebotys, “A framework for security on NoC technologies,” in *Proc. ISVLSI*, pp. 113–117, 2003.

[38] Y. Wang and G. Suh, “Efficient timing channel protection for on-chip networks,” in *Proc. NOCS’12*, pp. 142–151, 2012.

[39] L. Kim, J. Villasenor, and C. Koc, “A Trojan-resistant system-on-chip bus architecture,” in *Proc. Military Comm. Conf*. pp. 1–6, 2009.

[40] Semiconductor Research Corporation, “Reliable needs for secure, trustworthy, and reliable semiconductors,” [Online]: https://www.src.org/calendar/e004965/sa-ts-workshop-report-final.pdf.

[41] G.E. Suh and S. Devadas, “Physical Unclonable Functions for Device Authentication and Secret Key Generation,” in *Proc. DAC’07,* pp.9–14, June 2007.

[42] Herder, C.; Meng-Day Yu; Koushanfar, F.; Devadas, S. “Physical Unclonable Functions and Applications: A Tutorial,” *Proceedings of the IEEE,* vol. 102, no. 8, pp. 1126–1141, 2014.

[43] S. Evain, and J.-P. Diguet, “From NoC security analysis to design solutions,” in Proc. *IEEE Workshop on Signal Processing Systems Design and Implementation*, pp.166–171, Nov. 2005.

[44] S. Saponara,T. Bacchillone, E. Petr, L. Fanucci, R. Locatelli, and M. Coppola, “Design of an NoC Interface Macrocell with Hardware Support of Advanced Networking Functionalities,” *IEEE Trans. On Computers*, vol. 63, no. 3, pp. 609–621, Mar. 2014.

[45] Q. Yu and P. Ampadu, “Dual-Layer Adaptive Error Control for Network-on-Chip Links,” *IEEE Trans. on Very Large Scale Integration (VLSI) Systems*, vol. 20, no. 7, pp. 1304–1317, 2012.

[46] N. Matveeva, Y. Sheynin, and E. Suvorova, “QoS Support in Embedded Networks and NoC,” in *Proc. 16th Conference of Open Innovations Association FRUCT*, pp. 51–59, 2014.

[47] A. Moradi, M. T. M. Shalmani, and M. Salmasizadeh, “A generalized method of differential fault attack against AES cryptosystem,” in *Proc. CHES*, 2006, pp. 91–100.

[48] H. Bar-El, et al., "The Sorcerer's apprentice guide to fault attacks," *Cryptology ePrint Archive,* Report 2004/10, 2004.

[49] G. Di Natale, et al., "A reliable architecture for parallel implementations of the Advanced Encryption Standard," *J. Electronic Testing,* vol. 25, no. 4, 2009, pp. 269-278.

[50] A. Aysu, E. Gulcan, and P. Schaumont, "SIMON Says: break Area Records of Block Ciphers on FPGAs," *Embedded Systems Letters,* vol. 6, no. 2, 2014, pp. 37-40.

[51] J. Blomer and J. P. Seifert, "Fault based cryptanalysis of the Advanced Encryption Standard (AES)," *FC* 2003,LNCS 2742,pp.162-181, 2003

[52] T. Korak, M. Hoefler, "On the effects of clock and power supply tampering on two microcontroller platforms," in *Proc. Workshop on Fault Diagnosis and Tolerance in Cryptography (FDTC),* 2014, pp. 8-7.

[53] G. Bertoni, et al., "Error analysis and detection procedures for a hardware implementation of the Advanced Encryption Standard," *IEEE Trans. Computers*, vol. 52, no. 4, pp. 492–505, 2003.

[54] A. Barenghi, et al., "Fault injection attacks on cryptographic devices: Theory, practice, and countermeasures," in *Proc. IEEE*, vol. 100, no. 11, 2012, pp. 3056–3076.

[55] M. Mozaffari-Kermani, et al., "Fault-resilient lightweight cryptographic block ciphers for secure embedded systems," *IEEE Embedded Syst. Lett.*, vol. 6, no. 4, Dec. 2014, pp. 89–92.

[56] X. Guo and R. Karri, "Recomputing with permuted operands: A concurrent error detection approach," *IEEE Trans. Computer-Aided Design Integr. Circuits Syst.*, vol. 32, no. 10, Oct. 2013, pp. 1595–1608.

[57] Ramesh Karri et al., “Concurrent Error Detection of Fault-Based Side-Channel Cryptanalysis of 128-Bit Symmetric Block Ciphers,” In *Proc. DAC*, pages 579–585, 2001.

[58] M. Joye, P. Manet, and J. Rigaud, “Strengthening hardware AES implementations against fault attacks,” *IET Info Security*, vol. 1, no. 3, pp. 106-110, 2007.

[59] V. Lomne, T. Roche, and A. Thillard, “On the need of randomness in fault attack countermeasures—Application to AES,” in *Proc. FDTC*, pp. 85–94, Sep. 2012.

[60] Xiaofei Guo, Debdeep Mukhopadhyay, Chenglu Jin, Ramesh Karri,” NREPO: Normal basis Recomputing with Permuted Operands,” in *Proc. HOST*, pp. 118-123, 2014

[61] H. Tupsamudre, S. Bisht, and D. Mukhopadhyay, “Differential fault analysis on the families of SIMON and SPECK ciphers”, in *Proc. FDTC*, pp. 40–48, 2014.

[62] X. Guo, D. Mukhopadhyay, C. Jin, and R. Karri, “Security analysis of concurrent error detection against differential fault analysis,” in Proc. *J. Cryptogr. Eng.*, vol. 5, no. 3, pp. 153–169, 2015.

[63] M. Mozaffari Kermani and A. Reyhani-Masoleh, "Parity-based fault detection architecture of S-box for Advanced Encryption Standard," in Proc. *DFT* , pp. 572-580, October, 2006.

[64] M. Mozaffari Kermani and R. Azarderakhsh, “Efficient fault diagnosis schemes for reliable lightweight cryptographic ISO/IEC standard CLEFIA benchmarked on ASIC and FPGA,” IEEE Trans. *Ind. Electron.*, vol. 60, no. 12, pp. 5925-5932, Dec. 2013.

[65] M. Mozaffari Kermani and A. Reyhani-Masoleh, “Concurrent structure-independent fault detection schemes for the Advanced Encryption Standard,” *IEEE Trans. Comput.*, vol. 59, no. 5, pp. 608–622, May 2010.

[66] http://www.mckinsey.com/insights/high_tech_telecoms_internet/the_internet_of_things_sizing_up_the_opportunity.

[67] Hardware Security in the IoT, [URL] http://embedded-computing.com/articles/hardware-security-in-the-iot.

[68] http://research.ijcaonline.org/volume102/number2/pxc3898574.pdf.

[69] http://www.computerworld.com/article/2600344/telematics-insurers-will-now-be-able-to-track-driver-behavior-via-smartphones.html.

[70] http://qz.com/230055/car-insurance-companies-want-to-track-your-every-move-and-youre-going-to-let-them/.

[71] https://en.wikipedia.org/wiki/Usage-based_insurance.

[72] X. Chen, L. Sun, H. Zhu, Y. Zhen, and H. Chen, "Application of internet of things in power-line monitoring," in *Proc*. *Cyber-Enabled Distribute Computing and Knowledge Discovery (CyberC)*, pp. 423–426, Oct 2012.

[73] O. Monnier,"A smarter grid with the Internet of Things," Texas Instruments, 2013.

[74] Ahmad-Reza Sadeghi, Christian Wachsmann, Michael Waidner, "Security and Privacy Challenges in Industrial Internet of Things," in *Proc. DAC'15*, pp. 1-6, Jun. 2015.

[75] Brier, E., Clavier, C., and Olivier, F., "Correlation power analysis with a leakage model," in proc., *Lecture Notes in Computer Science*, vol. 3156, pp. 16–29. Springer, Berlin, 2004.

[76] P. Kocher, J. Jaffe, and B. Jun, "Differential Power Analysis," in *Proc. Crypto'99*, pp.388-397, 1999.

[77] S. Adee, "The hunt for the kill switch," *IEEE Spectrum* , May 1, 2008.

[78] Subhasish Mitra, "Stopping Hardware Trojans in Their Tracks," *IEEE Spectrum*, January 20, 2015.

[79] John Markoff, "F.B.I. Says the Military Had Bogus Computer Gear," May 9, 2008 [Online] Available: http://www.nytimes.com/2008/05/09/technology/09cisco.html?pagewanted=print

[80] H. Pahlevanzadeh, J. Dofe, and Q. Yu, "Assessing CPA Resistance of AES with Different Fault Tolerance Mechanisms," to appear in the *Proceedings of 21st Asia and South Pacific Design Automation Conference (ASP-DAC)*, Jan. 2016.

[81] O'Flynn, C., Chen, Z.D, "Synchronous Sampling and Clock Recovery of Internal Oscillators for Side Channel Analysis," *Cryptology ePrint Archive*, Report 2013/294.

[82] Dave Evans, "The Internet of Things How the Next Evolution of the Internet Is Changing Everything," *Cisco White Paper*, April 2011.

[83] M. Rostami, F. Koushanfar, and R. Karri, "A Primer on Hardware Security: Models, Methods, and Metrics," in *Proc. the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.

[84] McAfee, "Rootkits,Part 1 of 3: The Growing Threat," White Paper, Apr. 2006.

[85] C. O'Flynn and Z. Chen, "Chipwhisperer: An opensource platform for hardware embedded security research," in *Proc. Prouff*, pp. 243–260, 2014.

[86] A. P. Johnson, R. S. Chakraborty, D. Mukhopadyay, "A PUF-enabled Secure Architecture for FPGA-based IoT Applications," *IEEE Trans.on Multi-Scale Computing Systems*, pp. 1–14, 2015.

[87] Gang Qu, Lin Yuan, "Design THINGS for the Internet of Things — An EDA perspective," in *Proc. ICCAD*, pp. 411–416, 2014.

[88] F. Courbon, P. Loubet-Moundi, J. J. A. Fournier, A. Tria, “A high efficiency Hardware Trojan detection technique based on fast SEM imaging,” in *Proc. DATE*, pp. 788–793, 2015.

APPENDIX A – Module Schematics

1. Submodule schematics

    a. Five-port router

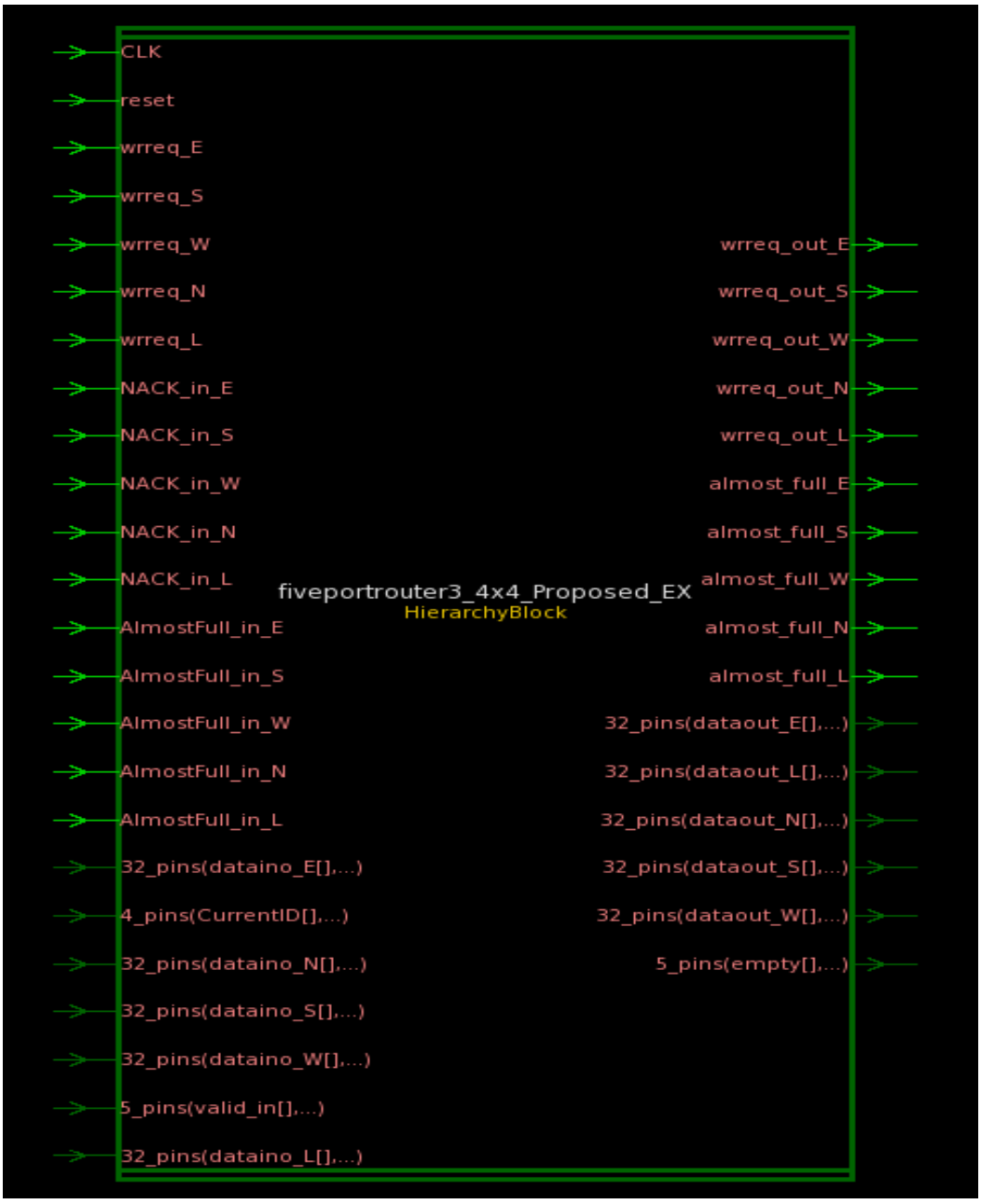


Figure A.1. External schematic of five-port router module, with I/O ports.

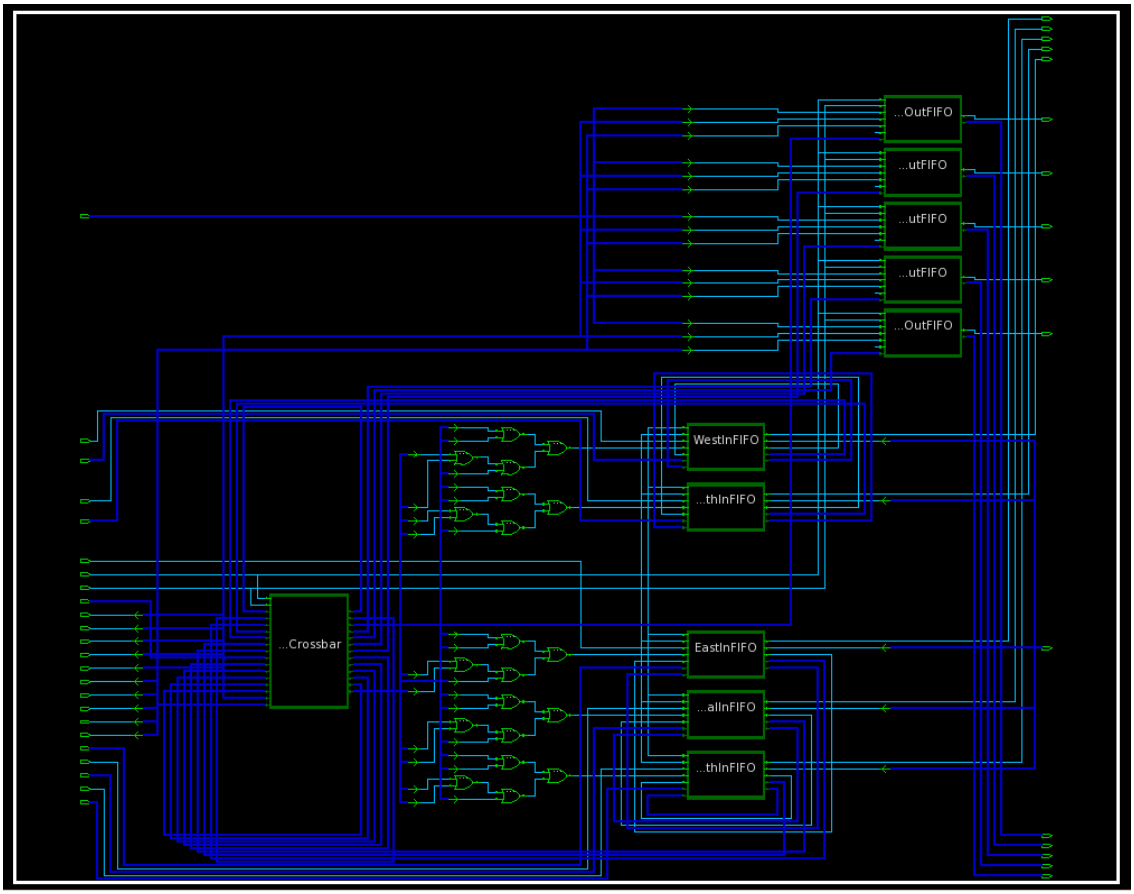


Figure A.2. Internal schematic of five-port router, with internal module connections.

b. Input FIFO

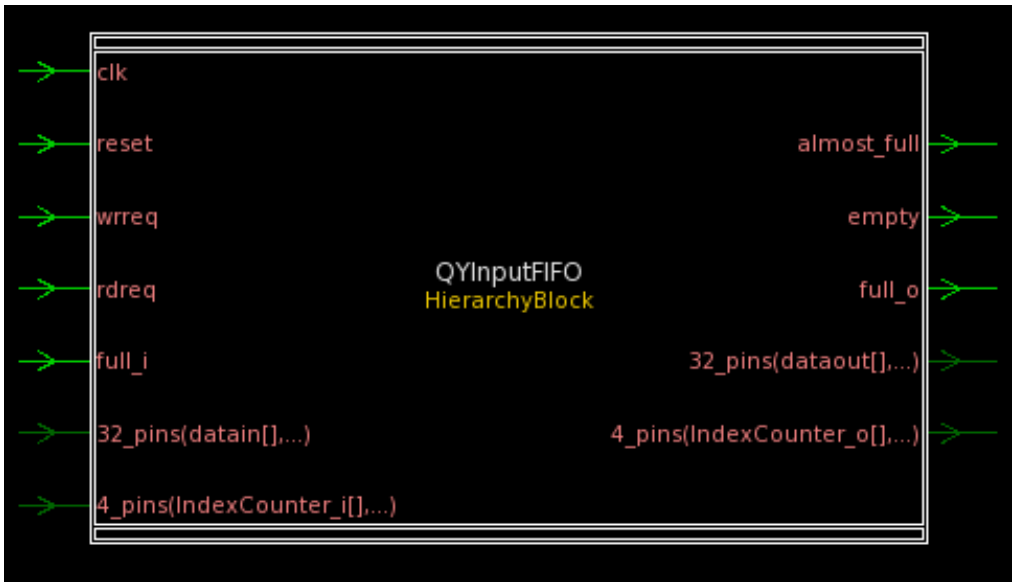


Figure A.3. External schematic for the input FIFO module, showing I/O pins.

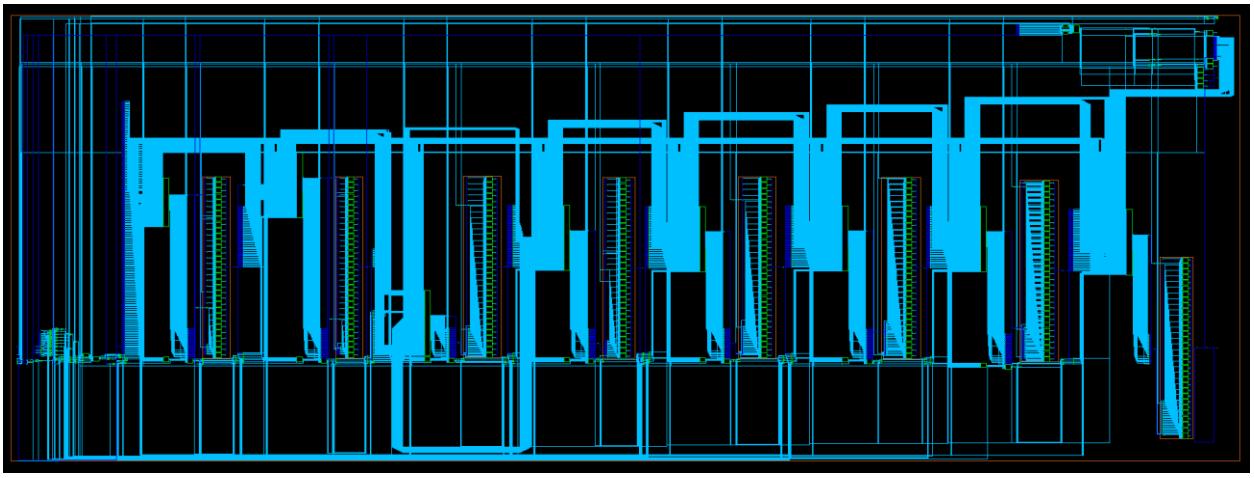


Figure A.4. Internal schematic of the input FIFO, showing internal register connections.

c. Full Crossbar

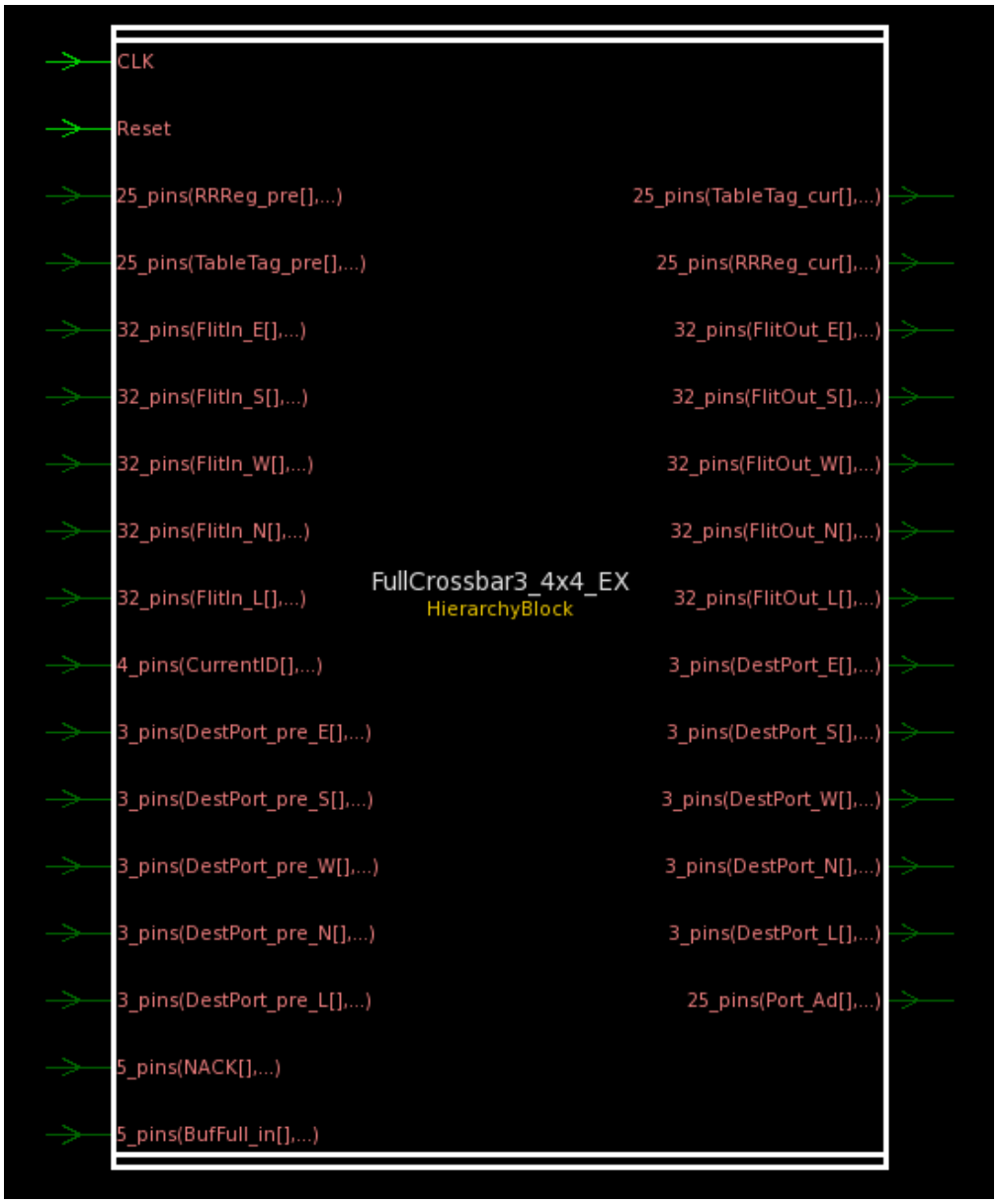


Figure A.5. Full Crossbar external schematic with I/O ports.

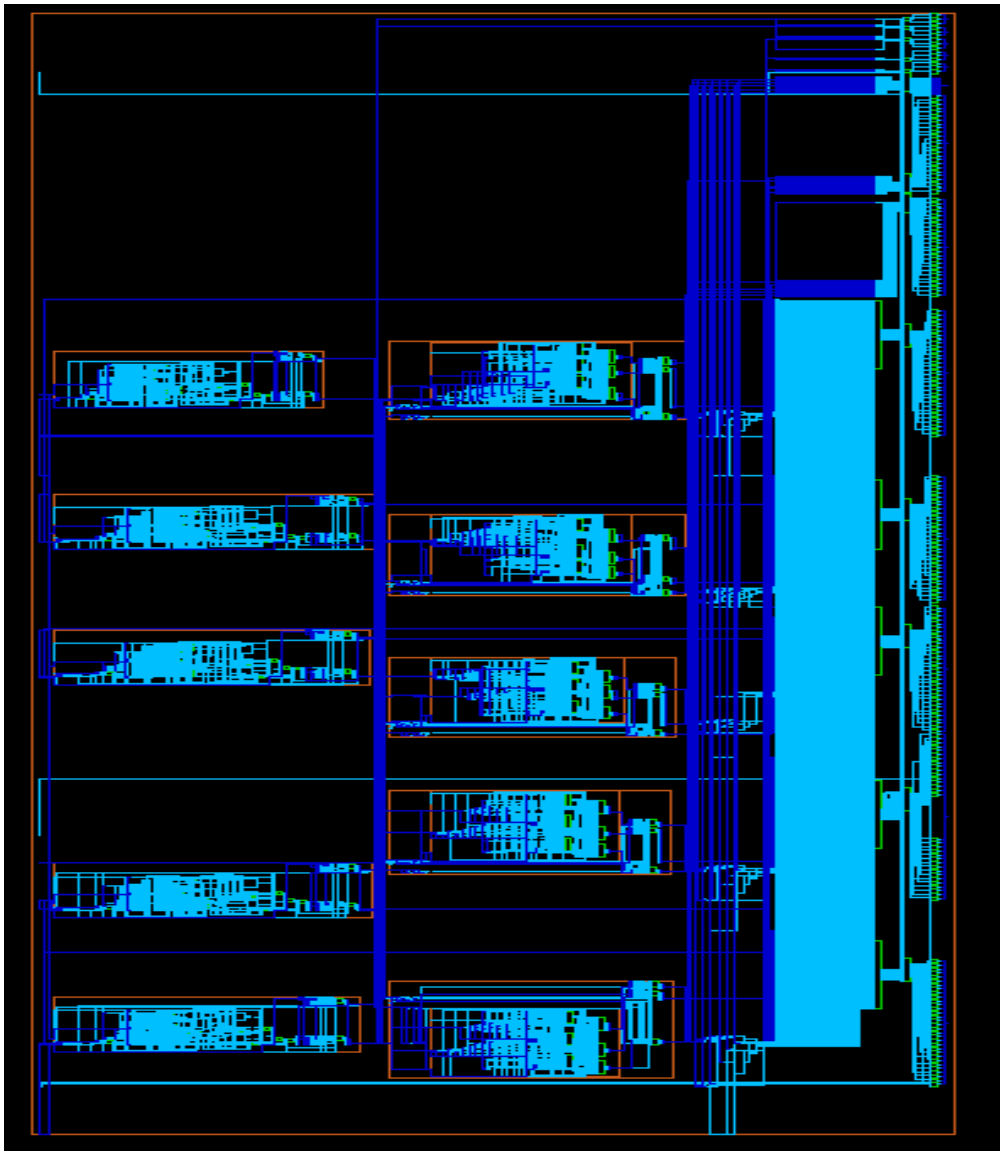


Figure A.6. Full Crossbar internal schematic with submodule connections.

d. Info Extraction

Figure A.7. External schematic of Info Extraction module, with I/O ports.

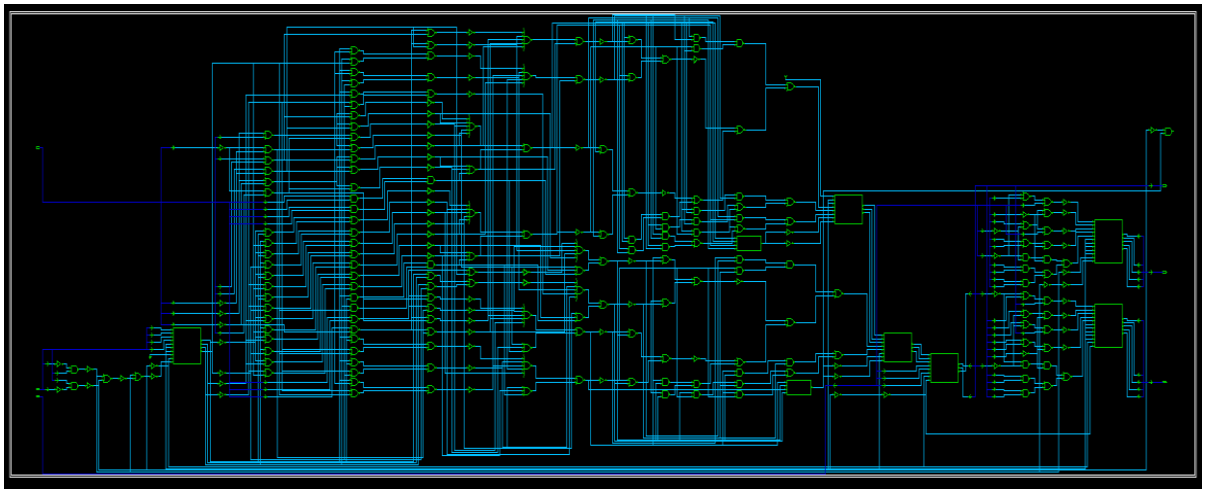


Figure A.8. Internal schematic of Info Extraction module.

e. Port Admission

Figure A.9. External schematic of Port Admission module, with I/O ports.

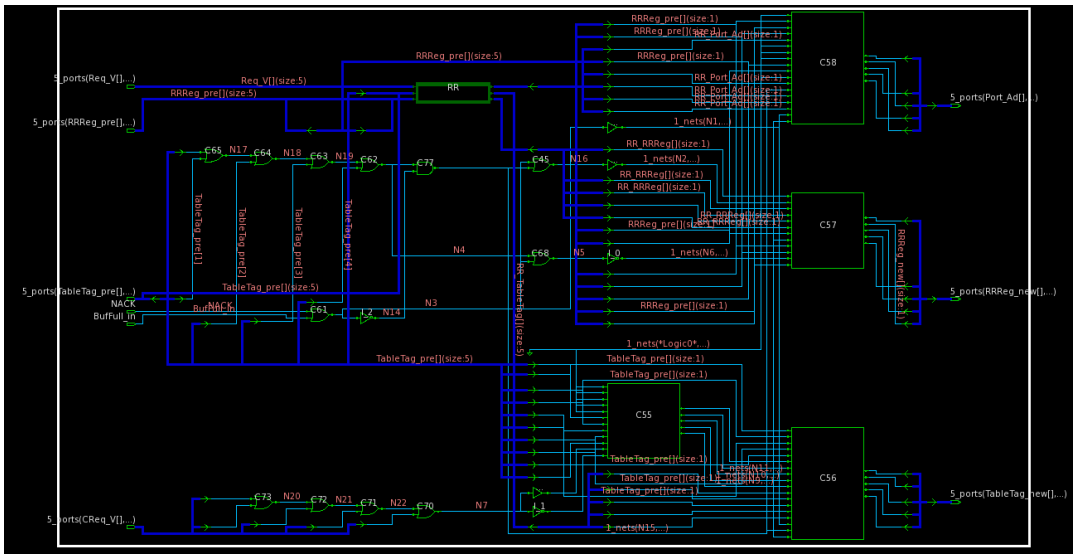


Figure A.10. Internal schematic of Port Admission module.

f. Round Robin

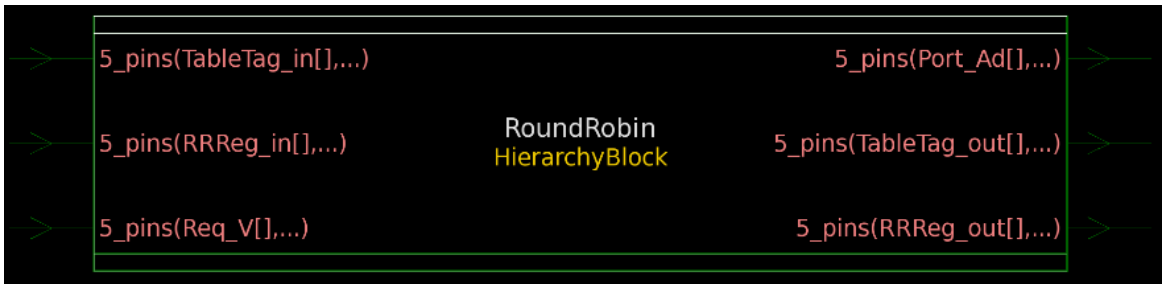


Figure A.11. System level schematic of Round Robin module

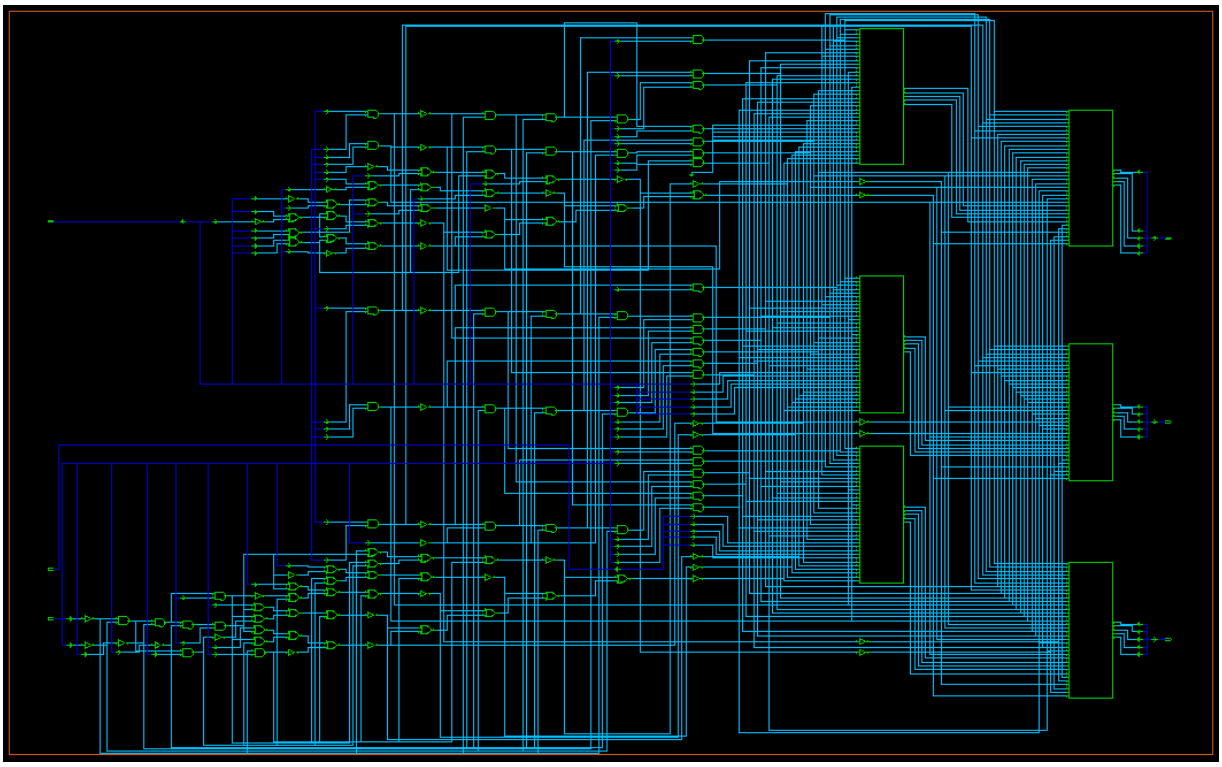


Figure A.12. Internal circuitry of Round Robin module

g. Circular FIFO

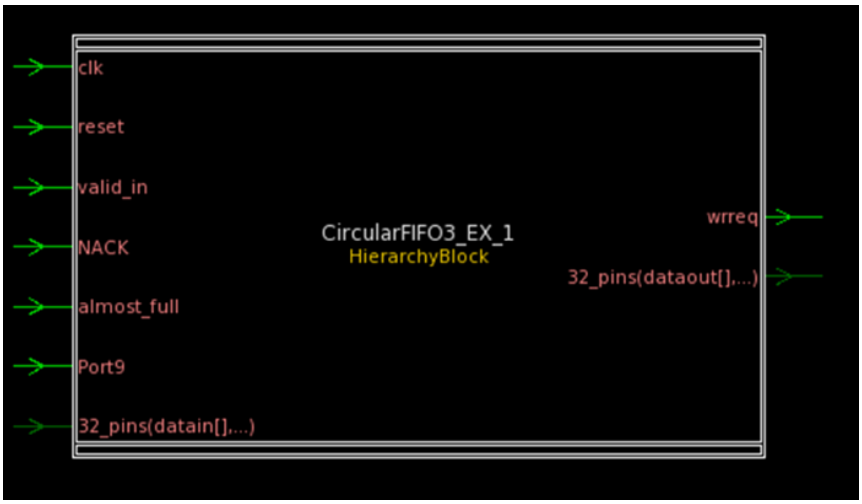


Figure A.13. System level schematic of Circular FIFO module

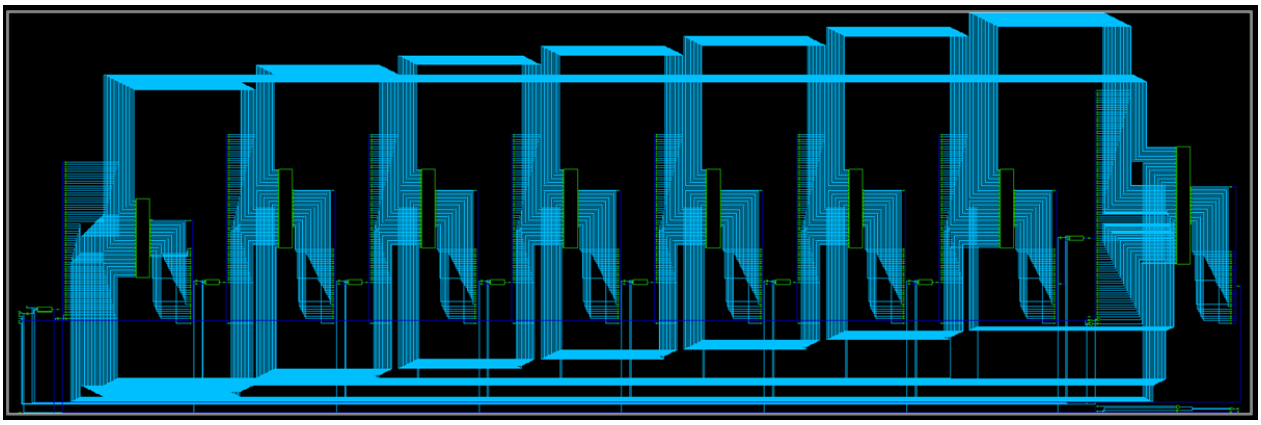


Figure A.14. Internal Circuitry for Circular FIFO module.

APPENDIX B - Verilog Code

a. Top Module (Testbench)

```
`timescale 1ns/1ps
//This testbench is to test all HT cases on the HT mitigation method of filtering illegal address
/*
This testbench is used to test the impact of HT numbers, location, and types on the NOC performance
@Yu & Frey, 12/14/2015
`define HT0 //
`define HT1 //Router 0
`define HT2 //additional Router 5
`define HT3 //additional Router 10, the HT definition is accumulated
`define HT32 //additional Router 15, the HT definition is accumulated
`define CENTER_ROUTER
`define BOUNDARY_ROUTER
`define MULTI_ROUTER //if this is not defined, single router is controlled by HT, otherwise, multiple Router is controlled
`define DEST_HT
`define HEAD_HT
`define TAIL_HT
//define those parameters on the ncverilog command line
*/
//Define which HT type, and which case (0-3HT)
//`define HT1 //additional Router 0
```

```
//`define HT2

//`define HT3

//`define TAIL_HT

//`define DEST_HT

//`define HEAD_HT


module NoC4x4_Proposed_SecureZoneCorrection;


parameter filename0 = "SecureZoneTest0.txt";

parameter filename1 = "SecureZoneTest1.txt";

parameter filename2 = "SecureZoneTest2.txt";

parameter filename3 = "SecureZoneTest3.txt";

parameter filename4 = "SecureZoneTest4.txt";

parameter filename5 = "SecureZoneTest5.txt";

parameter filename6 = "SecureZoneTest6.txt";

parameter filename7 = "SecureZoneTest7.txt";

parameter filename8 = "SecureZoneTest8.txt";

parameter filename9 = "SecureZoneTest9.txt";

parameter filename10 = "SecureZoneTest10.txt";

parameter filename11 = "SecureZoneTest11.txt";

parameter filename12 = "SecureZoneTest12.txt";

parameter filename13 = "SecureZoneTest13.txt";

parameter filename14 = "SecureZoneTest14.txt";
```

```
parameter filename15 = "SecureZoneTest15.txt";
parameter filename16 = "SecureZoneTestBandwidthAndCycles.txt";
parameter filename17 = "SecureZoneTestCycles.txt";
parameter filename18 = "SecureZoneTestBandwidth.txt";
parameter filename19 = "SecureZoneTestPacketInfo_placeholder.txt";
parameter injection = 1600;

//Trigger Signals
reg fivehead,fivetail,fivedest,zerotail,zerohead,zerodest,sevenhead,seventail,sevendest;

//correct/wrong packet check
integer
corr0,corr1,corr2,corr3,corr4,corr5,corr6,corr7,corr8,corr9,corr10,corr11,corr12,corr13,corr14,corr
15, corrtotal,rtotal;
integer ps0,ps1,ps2,ps3,ps4,ps5,ps6,ps7,ps8,ps9,ps10,ps11,ps12,ps13,ps14,ps15,pstotal;
//other
integer links0, links1, links2, links3, links4, links5, links6, links7, links8, links9, links10, links11,
links12, links13, links14, links15, links16, links17, links18, links19, links20, links21, links22,
links23, links24, links25, links26, links27, links28, links29, links30, links31, links32, links33,
links34, links35, links36, links37, links38, links39, links40, links41, links42, links43, links44,
links45, links46, links47 ;
integer code0, code1, code2, code3, code4, code5, code6, code7, code8, code9, code10, code11,
code12, code13, code14, code15;
```

```
integer op_file0, op_file1, op_file10, op_file11, op_file12, op_file13, op_file14, op_file15, op_file2,
op_file3, op_file4, op_file5, op_file6, op_file7, op_file8, op_file9, op_file16, op_file17, op_file18,
op_file19;
integer r_file0, r_file1, r_file2, r_file3, r_file4, r_file5, r_file6, r_file7, r_file8, r_file9, r_file10,
r_file11, r_file12, r_file13, r_file14, r_file15;
integer rpackets0, rpackets1, rpackets10, rpackets11, rpackets12, rpackets13, rpackets14,
rpackets15, rpackets2, rpackets3, rpackets4, rpackets5, rpackets6, rpackets7, rpackets8, rpackets9;
reg trigger; //HT trigger
reg     CLK,CLK2;
reg     reset;
//Interconnections
integer packetsSent;
integer packetsreceived;
integer BW;
wire    wrreq_00;
wire    wrreq_01;
wire    wrreq_02;
wire    wrreq_03;
wire    wrreq_04;
wire    wrreq_10;
wire    wrreq_11;
wire    wrreq_12;
wire    wrreq_13;
```

```
wire    wrreq_14;
wire    wrreq_20;
wire    wrreq_21;
wire    wrreq_22;
wire    wrreq_23;
wire    wrreq_24;
wire    wrreq_30;
wire    wrreq_31;
wire    wrreq_32;
wire    wrreq_33;
wire    wrreq_34;
wire    wrreq_40;
wire    wrreq_41;
wire    wrreq_42;
wire    wrreq_43;
wire    wrreq_44;
wire    wrreq_50;
wire    wrreq_51;
wire    wrreq_52;
wire    wrreq_53;
wire    wrreq_54;
wire    wrreq_60;
wire    wrreq_61;
```

```
wire	wrreq_62;
wire	wrreq_63;
wire	wrreq_64;
wire	wrreq_70;
wire	wrreq_71;
wire	wrreq_72;
wire	wrreq_73;
wire	wrreq_74;
wire	wrreq_80;
wire	wrreq_81;
wire	wrreq_82;
wire	wrreq_83;
wire	wrreq_84;
wire	wrreq_90;
wire	wrreq_91;
wire	wrreq_92;
wire	wrreq_93;
wire	wrreq_94;
wire	wrreq_100;
wire	wrreq_101;
wire	wrreq_102;
wire	wrreq_103;
wire	wrreq_104;
```

```
wire    wrreq_110;
wire    wrreq_111;
wire    wrreq_112;
wire    wrreq_113;
wire    wrreq_114;
wire    wrreq_120;
wire    wrreq_121;
wire    wrreq_122;
wire    wrreq_123;
wire    wrreq_124;
wire    wrreq_130;
wire    wrreq_131;
wire    wrreq_132;
wire    wrreq_133;
wire    wrreq_134;
wire    wrreq_140;
wire    wrreq_141;
wire    wrreq_142;
wire    wrreq_143;
wire    wrreq_144;
wire    wrreq_150;
wire    wrreq_151;
wire    wrreq_152;
```

```
wire    wrreq_153;
wire    wrreq_154;

reg     NACK_in_E;
reg     NACK_in_S;
reg     NACK_in_W;
reg     NACK_in_N;
reg     NACK_in_L;

wire    almost_full_00;
wire    almost_full_01;
wire    almost_full_02;
wire    almost_full_03;
wire    almost_full_04;
wire    almost_full_10;
wire    almost_full_11;
wire    almost_full_12;
wire    almost_full_13;
wire    almost_full_14;
wire    almost_full_20;
wire    almost_full_21;
wire    almost_full_22;
wire    almost_full_23;
```

```
wire	almost_full_24;
wire	almost_full_30;
wire	almost_full_31;
wire	almost_full_32;
wire	almost_full_33;
wire	almost_full_34;
wire	almost_full_40;
wire	almost_full_41;
wire	almost_full_42;
wire	almost_full_43;
wire	almost_full_44;
wire	almost_full_50;
wire	almost_full_51;
wire	almost_full_52;
wire	almost_full_53;
wire	almost_full_54;
wire	almost_full_60;
wire	almost_full_61;
wire	almost_full_62;
wire	almost_full_63;
wire	almost_full_64;
wire	almost_full_70;
wire	almost_full_71;
```

```
wire    almost_full_72;
wire    almost_full_73;
wire    almost_full_74;
wire    almost_full_80;
wire    almost_full_81;
wire    almost_full_82;
wire    almost_full_83;
wire    almost_full_84;
wire    almost_full_90;
wire    almost_full_91;
wire    almost_full_92;
wire    almost_full_93;
wire    almost_full_94;
wire    almost_full_100;
wire    almost_full_101;
wire    almost_full_102;
wire    almost_full_103;
wire    almost_full_104;
wire    almost_full_110;
wire    almost_full_111;
wire    almost_full_112;
wire    almost_full_113;
wire    almost_full_114;
```

```
wire    almost_full_120;
wire    almost_full_121;
wire    almost_full_122;
wire    almost_full_123;
wire    almost_full_124;
wire    almost_full_130;
wire    almost_full_131;
wire    almost_full_132;
wire    almost_full_133;
wire    almost_full_134;
wire    almost_full_140;
wire    almost_full_141;
wire    almost_full_142;
wire    almost_full_143;
wire    almost_full_144;
wire    almost_full_150;
wire    almost_full_151;
wire    almost_full_152;
wire    almost_full_153;
wire    almost_full_154;

wire    [31:0] data_00;
wire    [31:0] data_01;
```

```
wire    [31:0] data_02;
wire    [31:0] data_03;
wire    [31:0] data_04;
wire    [31:0] data_10;
wire    [31:0] data_11;
wire    [31:0] data_12;
wire    [31:0] data_13;
wire    [31:0] data_14;
wire    [31:0] data_20;
wire    [31:0] data_21;
wire    [31:0] data_22;
wire    [31:0] data_23;
wire    [31:0] data_24;
wire    [31:0] data_30;
wire    [31:0] data_31;
wire    [31:0] data_32;
wire    [31:0] data_33;
wire    [31:0] data_34;
wire    [31:0] data_40;
wire    [31:0] data_41;
wire    [31:0] data_42;
wire    [31:0] data_43;
wire    [31:0] data_44;
```

```
wire    [31:0] data_50;
wire    [31:0] data_51;
wire    [31:0] data_52;
wire    [31:0] data_53;
wire    [31:0] data_54;
wire    [31:0] data_60;
wire    [31:0] data_61;
wire    [31:0] data_62;
wire    [31:0] data_63;
wire    [31:0] data_64;
wire    [31:0] data_70;
wire    [31:0] data_71;
wire    [31:0] data_72;
wire    [31:0] data_73;
wire    [31:0] data_74;
wire    [31:0] data_80;
wire    [31:0] data_81;
wire    [31:0] data_82;
wire    [31:0] data_83;
wire    [31:0] data_84;
wire    [31:0] data_90;
wire    [31:0] data_91;
wire    [31:0] data_92;
```

```
wire    [31:0] data_93;
wire    [31:0] data_94;
wire    [31:0] data_100;
wire    [31:0] data_101;
wire    [31:0] data_102;
wire    [31:0] data_103;
wire    [31:0] data_104;
wire    [31:0] data_110;
wire    [31:0] data_111;
wire    [31:0] data_112;
wire    [31:0] data_113;
wire    [31:0] data_114;
wire    [31:0] data_120;
wire    [31:0] data_121;
wire    [31:0] data_122;
wire    [31:0] data_123;
wire    [31:0] data_124;
wire    [31:0] data_130;
wire    [31:0] data_131;
wire    [31:0] data_132;
wire    [31:0] data_133;
wire    [31:0] data_134;
wire    [31:0] data_140;
```

```
wire    [31:0] data_141;
wire    [31:0] data_142;
wire    [31:0] data_143;
wire    [31:0] data_144;
wire    [31:0] data_150;
wire    [31:0] data_151;
wire    [31:0] data_152;
wire    [31:0] data_153;
wire    [31:0] data_154;

wire    [4:0] empty_0;
wire    [4:0] empty_1;
wire    [4:0] empty_2;
wire    [4:0] empty_3;
wire    [4:0] empty_4;
wire    [4:0] empty_5;
wire    [4:0] empty_6;
wire    [4:0] empty_7;
wire    [4:0] empty_8;
wire    [4:0] empty_9;
wire    [4:0] empty_10;
wire    [4:0] empty_11;
wire    [4:0] empty_12;
```

```
wire      [4:0] empty_13;
wire      [4:0] empty_14;
wire      [4:0] empty_15;

wire[31:0] data_in0, data_in1,data_in2, data_in3, data_in4, data_in5, data_in6, data_in7, data_in8,
data_in9, data_in10, data_in11, data_in12, data_in13, data_in14, data_in15;
reg inreq0, inreq1, inreq2, inreq3, inreq4, inreq5, inreq6, inreq7, inreq8, inreq9, inreq10, inreq11,
inreq12, inreq13, inreq14, inreq15;
reg[13:0] Count;
reg[7:0] ID0, ID1, ID2, ID3, ID4, ID5, ID6, ID7, ID8, ID9, ID10, ID11, ID12, ID13, ID14, ID15;
reg[9:0] Info0, Info1, Info2, Info3, Info4, Info5, Info6, Info7, Info8, Info9, Info10, Info11, Info12,
Info13, Info14, Info15;
reg[4:0] Dest0, Dest1, Dest2, Dest3, Dest4, Dest5, Dest6, Dest7, Dest8, Dest9, Dest10, Dest11,
Dest12, Dest13, Dest14, Dest15;
integer Destr0, Destr1, Destr2, Destr3, Destr4, Destr5, Destr6, Destr7, Destr8, Destr9, Destr10,
Destr11, Destr12, Destr13, Destr14, Destr15;
reg[3:0] D0, D1, D2, D3, D4, D5, D6, D7, D8, D9, D10, D11, D12, D13, D14, D15;
reg[3:0] src0, src1, src2, src3, src4, src5, src6, src7, src8, src9, src10, src11, src12, src13, src14,
src15;
integer spackets0, spackets1, spackets2, spackets3, spackets4, spackets5, spackets6, spackets7,
spackets8, spackets9, spackets10, spackets11, spackets12, spackets13, spackets15, spackets14;

fiveportrouter4_Trojaned R0(
```

```
.CLK(CLK),
.reset(reset),
.wrreq_E(wrreq_12),
.wrreq_S(wrreq_33),
.wrreq_W(1'b0),
.wrreq_N(1'b0),
.wrreq_L(inreq0),
.NACK_in_E(NACK_in_E),
.NACK_in_S(NACK_in_S),
.NACK_in_W(NACK_in_W),
.NACK_in_N(NACK_in_N),
.NACK_in_L(NACK_in_L),
.AlmostFull_in_E(almost_full_12),
.AlmostFull_in_S(almost_full_33),
.AlmostFull_in_W(1'b0),
.AlmostFull_in_N(1'b0),
.AlmostFull_in_L(1'b0),
.CurrentID(4'b0000),
.dataino_E(data_12),
.dataino_L(data_in0),
.dataino_N(32'h00000000),
.dataino_S(data_33),
.dataino_W(32'h00000000),
```

```
        .valid_in({~empty_1[2], ~empty_3[3], 1'b0, 1'b0, 1'b1}),
        .wrreq_out_E(wrreq_00),
        .wrreq_out_S(wrreq_01),
        .wrreq_out_W(wrreq_02),
        .wrreq_out_N(wrreq_03),
        .wrreq_out_L(wrreq_04),
        .almost_full_E(almost_full_00),
        .almost_full_S(almost_full_01),
        .almost_full_W(almost_full_02),
        .almost_full_N(almost_full_03),
        .almost_full_L(almost_full_04),
        .dataout_E(data_00),
        .dataout_S(data_01),
        .dataout_W(data_02),
        .dataout_N(data_03),
        .dataout_L(data_04),
        .empty(empty_0),
        .TailHT_Trigger(zerotail), //HT triggers
        .HeadHT_Trigger(zerohead),
        .DestHT_Trigger(zerodest)
);

fiveportrouter4 R1(
```

```
.CLK(CLK),
.reset(reset),
.wrreq_E(wrreq_22),
.wrreq_S(wrreq_43),
.wrreq_W(wrreq_00),
.wrreq_N(1'b0),
.wrreq_L(inreq1),
.NACK_in_E(NACK_in_E),
.NACK_in_S(NACK_in_S),
.NACK_in_W(NACK_in_W),
.NACK_in_N(NACK_in_N),
.NACK_in_L(NACK_in_L),
.AlmostFull_in_E(almost_full_22),
.AlmostFull_in_S(almost_full_43),
.AlmostFull_in_W(almost_full_00),
.AlmostFull_in_N(1'b0),
.AlmostFull_in_L(1'b0),
.CurrentID(4'b0001),
.dataino_E(data_22),
.dataino_L(data_in1),
.dataino_N(32'h00000000),
.dataino_S(data_43),
.dataino_W(data_00),
```

```
        .valid_in({~empty_2[2], ~empty_4[3], ~empty_0[0], 1'b0, 1'b1}),
        .wrreq_out_E(wrreq_10),
        .wrreq_out_S(wrreq_11),
        .wrreq_out_W(wrreq_12),
        .wrreq_out_N(wrreq_13),
        .wrreq_out_L(wrreq_14),
        .almost_full_E(almost_full_10),
        .almost_full_S(almost_full_11),
        .almost_full_W(almost_full_12),
        .almost_full_N(almost_full_13),
        .almost_full_L(almost_full_14),
        .dataout_E(data_10),
        .dataout_S(data_11),
        .dataout_W(data_12),
        .dataout_N(data_13),
        .dataout_L(data_14),
        .empty(empty_1)
);

fiveportrouter4 R2(
        .CLK(CLK),
        .reset(reset),
        .wrreq_E(wrreq_92),
```

.wrreq_S(wrreq_53),

.wrreq_W(wrreq_10),

.wrreq_N(1'b0),

.wrreq_L(inreq2),

.NACK_in_E(NACK_in_E),

.NACK_in_S(NACK_in_S),

.NACK_in_W(NACK_in_W),

.NACK_in_N(NACK_in_N),

.NACK_in_L(NACK_in_L),

.AlmostFull_in_E(almost_full_92),

.AlmostFull_in_S(almost_full_53),

.AlmostFull_in_W(almost_full_10),

.AlmostFull_in_N(1'b0),

.AlmostFull_in_L(1'b0),

.CurrentID(4'b0010),

.dataino_E(data_92),

.dataino_L(data_in2),

.dataino_N(32'h00000000),

.dataino_S(data_53),

.dataino_W(data_10),

.valid_in({~empty_9[2], ~empty_5[3], ~empty_1[0], 1'b0, 1'b1}),

.wrreq_out_E(wrreq_20),

.wrreq_out_S(wrreq_21),

```
        .wrreq_out_W(wrreq_22),
        .wrreq_out_N(wrreq_23),
        .wrreq_out_L(wrreq_24),
        .almost_full_E(almost_full_20),
        .almost_full_S(almost_full_21),
        .almost_full_W(almost_full_22),
        .almost_full_N(almost_full_23),
        .almost_full_L(almost_full_24),
        .dataout_E(data_20),
        .dataout_S(data_21),
        .dataout_W(data_22),
        .dataout_N(data_23),
        .dataout_L(data_24),
        .empty(empty_2));

fiveportrouter4 R3(
        .CLK(CLK),
        .reset(reset),
        .wrreq_E(wrreq_42),
        .wrreq_S(wrreq_63),
        .wrreq_W(1'b0),
        .wrreq_N(wrreq_01),
        .wrreq_L(inreq3),
```

```
.NACK_in_E(NACK_in_E),
.NACK_in_S(NACK_in_S),
.NACK_in_W(NACK_in_W),
.NACK_in_N(NACK_in_N),
.NACK_in_L(NACK_in_L),
.AlmostFull_in_E(almost_full_42),
.AlmostFull_in_S(almost_full_63),
.AlmostFull_in_W(1'b0),
.AlmostFull_in_N(almost_full_01),
.AlmostFull_in_L(1'b0),
.CurrentID(4'b0100),
.dataino_E(data_42),
.dataino_L(data_in3),
.dataino_N(data_01),
.dataino_S(data_63),
.dataino_W(32'h00000000),
.valid_in({~empty_4[2],~empty_6[3], 1'b0, ~empty_0[1], 1'b1}),
.wrreq_out_E(wrreq_30),
.wrreq_out_S(wrreq_31),
.wrreq_out_W(wrreq_32),
.wrreq_out_N(wrreq_33),
.wrreq_out_L(wrreq_34),
.almost_full_E(almost_full_30),
```

```
        .almost_full_S(almost_full_31),
        .almost_full_W(almost_full_32),
        .almost_full_N(almost_full_33),
        .almost_full_L(almost_full_34),
        .dataout_E(data_30),
        .dataout_S(data_31),
        .dataout_W(data_32),
        .dataout_N(data_33),
        .dataout_L(data_34),
        .empty(empty_3));

fiveportrouter4_Trojaned R4(
        .CLK(CLK),
        .reset(reset),
        .wrreq_E(wrreq_52),
        .wrreq_S(wrreq_73),
        .wrreq_W(wrreq_30),
        .wrreq_N(wrreq_11),
        .wrreq_L(inreq4),
        .NACK_in_E(NACK_in_E),
        .NACK_in_S(NACK_in_S),
        .NACK_in_W(NACK_in_W),
        .NACK_in_N(NACK_in_N),
```

```
.NACK_in_L(NACK_in_L),
.AlmostFull_in_E(almost_full_52),
.AlmostFull_in_S(almost_full_73),
.AlmostFull_in_W(almost_full_30),
.AlmostFull_in_N(almost_full_11),
.AlmostFull_in_L(1'b0),
.CurrentID(4'b0101),
.dataino_E(data_52),
.dataino_L(data_in4),
.dataino_N(data_11),
.dataino_S(data_73),
.dataino_W(data_30),
.valid_in({~empty_5[2], ~empty_7[3], ~empty_3[0], ~empty_1[1], 1'b1}),
.wrreq_out_E(wrreq_40),
.wrreq_out_S(wrreq_41),
.wrreq_out_W(wrreq_42),
.wrreq_out_N(wrreq_43),
.wrreq_out_L(wrreq_44),
.almost_full_E(almost_full_40),
.almost_full_S(almost_full_41),
.almost_full_W(almost_full_42),
.almost_full_N(almost_full_43),
.almost_full_L(almost_full_44),
```

```
        .dataout_E(data_40),
        .dataout_S(data_41),
        .dataout_W(data_42),
        .dataout_N(data_43),
        .dataout_L(data_44),
        .empty(empty_4),
        .TailHT_Trigger(fivetail), //HT triggers
        .HeadHT_Trigger(fivehead),
        .DestHT_Trigger(fivedest) );

fiveportrouter4 R5(
        .CLK(CLK),
        .reset(reset),
        .wrreq_E(wrreq_102),
        .wrreq_S(wrreq_83),
        .wrreq_W(wrreq_40),
        .wrreq_N(wrreq_21),
        .wrreq_L(inreq5),
        .NACK_in_E(NACK_in_E),
        .NACK_in_S(NACK_in_S),
        .NACK_in_W(NACK_in_W),
        .NACK_in_N(NACK_in_N),
        .NACK_in_L(NACK_in_L),
```

```
.AlmostFull_in_E(almost_full_102),
.AlmostFull_in_S(almost_full_83),
.AlmostFull_in_W(almost_full_40),
.AlmostFull_in_N(almost_full_21),
.AlmostFull_in_L(1'b0),
.CurrentID(4'b0110),
.dataino_E(data_102),
.dataino_L(data_in5),
.dataino_N(data_21),
.dataino_S(data_83),
.dataino_W(data_40),
.valid_in({~empty_10[2], ~empty_8[3], ~empty_4[0], ~empty_2[1], 1'b1}),
.wrreq_out_E(wrreq_50),
.wrreq_out_S(wrreq_51),
.wrreq_out_W(wrreq_52),
.wrreq_out_N(wrreq_53),
.wrreq_out_L(wrreq_54),
.almost_full_E(almost_full_50),
.almost_full_S(almost_full_51),
.almost_full_W(almost_full_52),
.almost_full_N(almost_full_53),
.almost_full_L(almost_full_54),
.dataout_E(data_50),
```

```
        .dataout_S(data_51),
        .dataout_W(data_52),
        .dataout_N(data_53),
        .dataout_L(data_54),
        .empty(empty_5));

fiveportrouter4 R6(
        .CLK(CLK),
        .reset(reset),
        .wrreq_E(wrreq_72),
        .wrreq_S(wrreq_123),
        .wrreq_W(1'b0),
        .wrreq_N(wrreq_31),
        .wrreq_L(inreq6),
        .NACK_in_E(NACK_in_E),
        .NACK_in_S(NACK_in_S),
        .NACK_in_W(NACK_in_W),
        .NACK_in_N(NACK_in_N),
        .NACK_in_L(NACK_in_L),
        .AlmostFull_in_E(almost_full_72),
        .AlmostFull_in_S(almost_full_123),
        .AlmostFull_in_W(1'b0),
        .AlmostFull_in_N(almost_full_31),
```

.AlmostFull_in_L(1'b0),

.CurrentID(4'b1000),

.dataino_E(data_72),

.dataino_L(data_in6),

.dataino_N(data_31),

.dataino_S(data_123),

.dataino_W(32'h00000000),

.valid_in({~empty_7[2], ~empty_12[3], 1'b0, ~empty_3[1], 1'b1}),

.wrreq_out_E(wrreq_60),

.wrreq_out_S(wrreq_61),

.wrreq_out_W(wrreq_62),

.wrreq_out_N(wrreq_63),

.wrreq_out_L(wrreq_64),

.almost_full_E(almost_full_60),

.almost_full_S(almost_full_61),

.almost_full_W(almost_full_62),

.almost_full_N(almost_full_63),

.almost_full_L(almost_full_64),

.dataout_E(data_60),

.dataout_S(data_61),

.dataout_W(data_62),

.dataout_N(data_63),

.dataout_L(data_64),

```
        .empty(empty_6)
        );

fiveportrouter4_Trojaned R7(
        .CLK(CLK),
        .reset(reset),
        .wrreq_E(wrreq_82),
        .wrreq_S(wrreq_133),
        .wrreq_W(wrreq_60),
        .wrreq_N(wrreq_41),
        .wrreq_L(inreq7),
        .NACK_in_E(NACK_in_E),
        .NACK_in_S(NACK_in_S),
        .NACK_in_W(NACK_in_W),
        .NACK_in_N(NACK_in_N),
        .NACK_in_L(NACK_in_L),
        .AlmostFull_in_E(almost_full_82),
        .AlmostFull_in_S(almost_full_133),
        .AlmostFull_in_W(almost_full_60),
        .AlmostFull_in_N(almost_full_41),
        .AlmostFull_in_L(1'b0),
        .CurrentID(4'b1001),
  .dataino_E(data_82),
```

.dataino_L(data_in7),

.dataino_N(data_41),

.dataino_S(data_133),

.dataino_W(data_60),

.valid_in({~empty_8[2], ~empty_13[3], ~empty_6[0], ~empty_4[1], 1'b1}),

.wrreq_out_E(wrreq_70),

.wrreq_out_S(wrreq_71),

.wrreq_out_W(wrreq_72),

.wrreq_out_N(wrreq_73),

.wrreq_out_L(wrreq_74),

.almost_full_E(almost_full_70),

.almost_full_S(almost_full_71),

.almost_full_W(almost_full_72),

.almost_full_N(almost_full_73),

.almost_full_L(almost_full_74),

.dataout_E(data_70),

.dataout_S(data_71),

.dataout_W(data_72),

.dataout_N(data_73),

.dataout_L(data_74),

.empty(empty_7),

.TailHT_Trigger(seventail), //HT triggers

.HeadHT_Trigger(sevenhead),

```
        .DestHT_Trigger(sevendest)
        );

fiveportrouter4 R8(
        .CLK(CLK),
        .reset(reset),
        .wrreq_E(wrreq_112),
        .wrreq_S(wrreq_143),
        .wrreq_W(wrreq_70),
        .wrreq_N(wrreq_51),
        .wrreq_L(inreq8),
        .NACK_in_E(NACK_in_E),
        .NACK_in_S(NACK_in_S),
        .NACK_in_W(NACK_in_W),
        .NACK_in_N(NACK_in_N),
        .NACK_in_L(NACK_in_L),
        .AlmostFull_in_E(almost_full_112),
        .AlmostFull_in_S(almost_full_143),
        .AlmostFull_in_W(almost_full_70),
        .AlmostFull_in_N(almost_full_51),
        .AlmostFull_in_L(1'b0),
        .CurrentID(4'b1010),
        .dataino_E(data_112),
```

```
.dataino_L(data_in8),
.dataino_N(data_51),
.dataino_S(data_143),
.dataino_W(data_70),
.valid_in({~empty_11[2], ~empty_14[3], ~empty_7[0], ~empty_5[1], 1'b1}),
.wrreq_out_E(wrreq_80),
.wrreq_out_S(wrreq_81),
.wrreq_out_W(wrreq_82),
.wrreq_out_N(wrreq_83),
.wrreq_out_L(wrreq_84),
.almost_full_E(almost_full_80),
.almost_full_S(almost_full_81),
.almost_full_W(almost_full_82),
.almost_full_N(almost_full_83),
.almost_full_L(almost_full_84),
.dataout_E(data_80),
.dataout_S(data_81),
.dataout_W(data_82),
.dataout_N(data_83),
.dataout_L(data_84),
.empty(empty_8)
);
```

```
fiveportrouter4 R9(
        .CLK(CLK),
        .reset(reset),
        .wrreq_E(1'b0),
        .wrreq_S(wrreq_103),
        .wrreq_W(wrreq_20),
        .wrreq_N(1'b0),
        .wrreq_L(inreq9),
        .NACK_in_E(NACK_in_E),
        .NACK_in_S(NACK_in_S),
        .NACK_in_W(NACK_in_W),
        .NACK_in_N(NACK_in_N),
        .NACK_in_L(NACK_in_L),
        .AlmostFull_in_E(1'b0),
        .AlmostFull_in_S(almost_full_103),
        .AlmostFull_in_W(almost_full_20),
        .AlmostFull_in_N(1'b0),
        .AlmostFull_in_L(1'b0),
        .CurrentID(4'b0011),
        .dataino_E(32'd0),
        .dataino_L(data_in9),
        .dataino_N(32'd0),
        .dataino_S(data_103),
```

```
        .dataino_W(data_20),
        .valid_in({1'b0, ~empty_10[3], ~empty_2[0],1'b0, 1'b1}),
        .wrreq_out_E(wrreq_90),
        .wrreq_out_S(wrreq_91),
        .wrreq_out_W(wrreq_92),
        .wrreq_out_N(wrreq_93),
        .wrreq_out_L(wrreq_94),
        .almost_full_E(almost_full_90),
        .almost_full_S(almost_full_91),
        .almost_full_W(almost_full_92),
        .almost_full_N(almost_full_93),
        .almost_full_L(almost_full_94),
        .dataout_E(data_90),
        .dataout_S(data_91),
        .dataout_W(data_92),
        .dataout_N(data_93),
        .dataout_L(data_94),
        .empty(empty_9)
        );

fiveportrouter4 R10(
        .CLK(CLK),
        .reset(reset),
```

.wrreq_E(1'b0),

.wrreq_S(wrreq_113),

.wrreq_W(wrreq_50),

.wrreq_N(wrreq_91),

.wrreq_L(inreq10),

.NACK_in_E(NACK_in_E),

.NACK_in_S(NACK_in_S),

.NACK_in_W(NACK_in_W),

.NACK_in_N(NACK_in_N),

.NACK_in_L(NACK_in_L),

.AlmostFull_in_E(1'b0),

.AlmostFull_in_S(almost_full_113),

.AlmostFull_in_W(almost_full_50),

.AlmostFull_in_N(almost_full_91),

.AlmostFull_in_L(1'b0),

.CurrentID(4'b0111),

.dataino_E(32'd0),

.dataino_L(data_in10),

.dataino_N(data_91),

.dataino_S(data_113),

.dataino_W(data_50),

.valid_in({1'b0, ~empty_11[3], ~empty_5[0], ~empty_9[1], 1'b1}),

.wrreq_out_E(wrreq_100),

```
        .wrreq_out_S(wrreq_101),
        .wrreq_out_W(wrreq_102),
        .wrreq_out_N(wrreq_103),
        .wrreq_out_L(wrreq_104),
        .almost_full_E(almost_full_100),
        .almost_full_S(almost_full_101),
        .almost_full_W(almost_full_102),
        .almost_full_N(almost_full_103),
        .almost_full_L(almost_full_104),
        .dataout_E(data_100),
        .dataout_S(data_101),
        .dataout_W(data_102),
        .dataout_N(data_103),
        .dataout_L(data_104),
        .empty(empty_10)
        );

fiveportrouter4 R11(
        .CLK(CLK),
        .reset(reset),
        .wrreq_E(1'b0),
        .wrreq_S(wrreq_153),
        .wrreq_W(wrreq_80),
```

```
.wrreq_N(wrreq_101),
.wrreq_L(inreq11),
.NACK_in_E(NACK_in_E),
.NACK_in_S(NACK_in_S),
.NACK_in_W(NACK_in_W),
.NACK_in_N(NACK_in_N),
.NACK_in_L(NACK_in_L),
.AlmostFull_in_E(1'b0),
.AlmostFull_in_S(almost_full_153),
.AlmostFull_in_W(almost_full_80),
.AlmostFull_in_N(almost_full_101),
.AlmostFull_in_L(1'b0),
.CurrentID(4'b1011),
.dataino_E(32'd0),
.dataino_L(data_in11),
.dataino_N(data_101),
.dataino_S(data_153),
.dataino_W(data_80),
.valid_in({1'b0, ~empty_15[3], ~empty_8[0], ~empty_10[1], 1'b1}),
.wrreq_out_E(wrreq_110),
.wrreq_out_S(wrreq_111),
.wrreq_out_W(wrreq_112),
.wrreq_out_N(wrreq_113),
```

```
        .wrreq_out_L(wrreq_114),
        .almost_full_E(almost_full_110),
        .almost_full_S(almost_full_111),
        .almost_full_W(almost_full_112),
        .almost_full_N(almost_full_113),
        .almost_full_L(almost_full_114),
        .dataout_E(data_110),
        .dataout_S(data_111),
        .dataout_W(data_112),
        .dataout_N(data_113),
        .dataout_L(data_114),
        .empty(empty_11)
        );

fiveportrouter4 R12(
        .CLK(CLK),
        .reset(reset),
        .wrreq_E(wrreq_132),
        .wrreq_S(1'b0),
        .wrreq_W(1'b0),
        .wrreq_N(wrreq_61),
        .wrreq_L(inreq12),
        .NACK_in_E(NACK_in_E),
```

.NACK_in_S(NACK_in_S),

.NACK_in_W(NACK_in_W),

.NACK_in_N(NACK_in_N),

.NACK_in_L(NACK_in_L),

.AlmostFull_in_E(almost_full_132),

.AlmostFull_in_S(1'b0),

.AlmostFull_in_W(1'b0),

.AlmostFull_in_N(almost_full_61),

.AlmostFull_in_L(1'b0),

.CurrentID(4'b1100),

.dataino_E(data_132),

.dataino_L(data_in12),

.dataino_N(data_61),

.dataino_S(32'd0),

.dataino_W(32'd0),

.valid_in({~empty_13[2], 1'b0, 1'b0, ~empty_6[1], 1'b1}),

.wrreq_out_E(wrreq_120),

.wrreq_out_S(wrreq_121),

.wrreq_out_W(wrreq_122),

.wrreq_out_N(wrreq_123),

.wrreq_out_L(wrreq_124),

.almost_full_E(almost_full_120),

.almost_full_S(almost_full_121),

```
        .almost_full_W(almost_full_122),
        .almost_full_N(almost_full_123),
        .almost_full_L(almost_full_124),
        .dataout_E(data_120),
        .dataout_S(data_121),
        .dataout_W(data_122),
        .dataout_N(data_123),
        .dataout_L(data_124),
        .empty(empty_12)
        );

fiveportrouter4 R13(
        .CLK(CLK),
        .reset(reset),
        .wrreq_E(wrreq_142),
        .wrreq_S(1'b0),
        .wrreq_W(wrreq_120),
        .wrreq_N(wrreq_71),
        .wrreq_L(inreq13),
        .NACK_in_E(NACK_in_E),
        .NACK_in_S(NACK_in_S),
        .NACK_in_W(NACK_in_W),
        .NACK_in_N(NACK_in_N),
```

```
.NACK_in_L(NACK_in_L),
.AlmostFull_in_E(almost_full_142),
.AlmostFull_in_S(1'b0),
.AlmostFull_in_W(almost_full_120),
.AlmostFull_in_N(almost_full_71),
.AlmostFull_in_L(1'b0),
.CurrentID(4'b1101),
.dataino_E(data_142),
.dataino_L(data_in13),
.dataino_N(data_71),
.dataino_S(32'd0),
.dataino_W(data_120),
.valid_in({~empty_14[2], 1'b0, ~empty_12[0], ~empty_7[1], 1'b1}),
.wrreq_out_E(wrreq_130),
.wrreq_out_S(wrreq_131),
.wrreq_out_W(wrreq_132),
.wrreq_out_N(wrreq_133),
.wrreq_out_L(wrreq_134),
.almost_full_E(almost_full_130),
.almost_full_S(almost_full_131),
.almost_full_W(almost_full_132),
.almost_full_N(almost_full_133),
.almost_full_L(almost_full_134),
```

```
        .dataout_E(data_130),
        .dataout_S(data_131),
        .dataout_W(data_132),
        .dataout_N(data_133),
        .dataout_L(data_134),
        .empty(empty_13)
        );

fiveportrouter4 R14(
        .CLK(CLK),
        .reset(reset),
        .wrreq_E(wrreq_152),
        .wrreq_S(1'b0),
        .wrreq_W(wrreq_130),
        .wrreq_N(wrreq_81),
        .wrreq_L(inreq14),
        .NACK_in_E(NACK_in_E),
        .NACK_in_S(NACK_in_S),
        .NACK_in_W(NACK_in_W),
        .NACK_in_N(NACK_in_N),
        .NACK_in_L(NACK_in_L),
        .AlmostFull_in_E(almost_full_152),
        .AlmostFull_in_S(1'b0),
```

.AlmostFull_in_W(almost_full_130),

.AlmostFull_in_N(almost_full_81),

.AlmostFull_in_L(1'b0),

.CurrentID(4'b1110),

.dataino_E(data_152),

.dataino_L(data_in14),

.dataino_N(data_81),

.dataino_S(32'd0),

.dataino_W(data_130),

.valid_in({~empty_15[2], 1'b0, ~empty_13[0], ~empty_8[1], 1'b1}),

.wrreq_out_E(wrreq_140),

.wrreq_out_S(wrreq_141),

.wrreq_out_W(wrreq_142),

.wrreq_out_N(wrreq_143),

.wrreq_out_L(wrreq_144),

.almost_full_E(almost_full_140),

.almost_full_S(almost_full_141),

.almost_full_W(almost_full_142),

.almost_full_N(almost_full_143),

.almost_full_L(almost_full_144),

.dataout_E(data_140),

.dataout_S(data_141),

.dataout_W(data_142),

```
        .dataout_N(data_143),
        .dataout_L(data_144),
        .empty(empty_14)
        );

fiveportrouter4 R15(
        .CLK(CLK),
        .reset(reset),
        .wrreq_E(1'b0),
        .wrreq_S(1'b0),
        .wrreq_W(wrreq_140),
        .wrreq_N(wrreq_111),
        .wrreq_L(inreq15),
        .NACK_in_E(NACK_in_E),
        .NACK_in_S(NACK_in_S),
        .NACK_in_W(NACK_in_W),
        .NACK_in_N(NACK_in_N),
        .NACK_in_L(NACK_in_L),
        .AlmostFull_in_E(1'b0),
        .AlmostFull_in_S(1'b0),
        .AlmostFull_in_W(almost_full_140),
        .AlmostFull_in_N(almost_full_111),
        .AlmostFull_in_L(1'b0),
```

.CurrentID(4'b1111),

.dataino_E(32'd0),

.dataino_L(data_in15),

.dataino_N(data_111),

.dataino_S(32'd0),

.dataino_W(data_140),

.valid_in({1'b0, 1'b0, ~empty_14[0], ~empty_11[1], 1'b1}),

.wrreq_out_E(wrreq_150),

.wrreq_out_S(wrreq_151),

.wrreq_out_W(wrreq_152),

.wrreq_out_N(wrreq_153),

.wrreq_out_L(wrreq_154),

.almost_full_E(almost_full_150),

.almost_full_S(almost_full_151),

.almost_full_W(almost_full_152),

.almost_full_N(almost_full_153),

.almost_full_L(almost_full_154),

.dataout_E(data_150),

.dataout_S(data_151),

.dataout_W(data_152),

.dataout_N(data_153),

.dataout_L(data_154),

.empty(empty_15)

```
        );

assign data_in0 = {Count[13:0],D0[3:0],ID0[3:0],Info0[9:0]};
assign data_in1 = {Count[13:0],D1[3:0],ID1[3:0],Info1[9:0]};
assign data_in2 = {Count[13:0],D2[3:0],ID2[3:0],Info2[9:0]};
assign data_in3 = {Count[13:0],D4[3:0],ID4[3:0],Info4[9:0]};
assign data_in4 = {Count[13:0],D5[3:0],ID5[3:0],Info5[9:0]};
assign data_in5 = {Count[13:0],D6[3:0],ID6[3:0],Info6[9:0]};
assign data_in6 = {Count[13:0],D8[3:0],ID8[3:0],Info8[9:0]};
assign data_in7 = {Count[13:0],D9[3:0],ID9[3:0],Info9[9:0]};
assign data_in8 = {Count[13:0],D10[3:0],ID10[3:0],Info10[9:0]};
assign data_in9 = {Count[13:0],D3[3:0],ID3[3:0],Info3[9:0]};
assign data_in10 = {Count[13:0],D7[3:0],ID7[3:0],Info7[9:0]};
assign data_in11 = {Count[13:0],D11[3:0],ID11[3:0],Info11[9:0]};
assign data_in12 = {Count[13:0],D12[3:0],ID12[3:0],Info12[9:0]};
assign data_in13 = {Count[13:0],D13[3:0],ID13[3:0],Info13[9:0]};
assign data_in14 = {Count[13:0],D14[3:0],ID14[3:0],Info14[9:0]};
assign data_in15 = {Count[13:0],D15[3:0],ID15[3:0],Info15[9:0]};


 initial
 begin
 `ifdef HT1
```

```
    `ifdef DEST_HT
      begin
      fivetail <= 1'b0;
      fivedest <= 1'b1;
      fivehead <= 1'b0;
      zerotail <= 1'b0;
      zerohead <= 1'b0;
      zerodest <= 1'b0;
      sevenhead <= 1'b0;
      seventail <= 1'b0;
      sevendest <= 1'b0;
      end
    `else
      `ifdef HEAD_HT
begin
      fivetail <= 1'b0;
      fivedest <= 1'b0;
      fivehead <= 1'b1;
      zerotail <= 1'b0;
      zerohead <= 1'b0;
      zerodest <= 1'b0;
      sevenhead <= 1'b0;
      seventail <= 1'b0;
```

```
sevendest <= 1'b0;
end

`else
  `ifdef TAIL_HT
    begin
fivetail <= 1'b1;
fivedest <= 1'b0;
fivehead <= 1'b0;
zerotail <= 1'b0;
zerohead <= 1'b0;
zerodest <= 1'b0;
sevenhead <= 1'b0;
seventail <= 1'b0;
sevendest <= 1'b0;
end //tail lost
  `else
     begin
fivetail <= 1'b0;
fivedest <= 1'b0;
fivehead <= 1'b0;
zerotail <= 1'b0;
zerohead <= 1'b0;
```

```
    zerodest <= 1'b0;
    sevenhead <= 1'b0;
    seventail <= 1'b0;
    sevendest <= 1'b0;
    end
      `endif //TAIL_HT
     `endif //HEAD_HT
   `endif //DEST_HT
`else
   `ifdef HT2
         `ifdef DEST_HT
            begin
    fivetail <= 1'b0;
    fivedest <= 1'b1;
    fivehead <= 1'b0;
    zerotail <= 1'b0;
    zerohead <= 1'b0;
    zerodest <= 1'b1;
    sevenhead <= 1'b0;
    seventail <= 1'b0;
    sevendest <= 1'b0;
    end
         `else
```

```
            `ifdef HEAD_HT
              begin
fivetail <= 1'b0;
fivedest <= 1'b0;
fivehead <= 1'b1;
zerotail <= 1'b0;
zerohead <= 1'b1;
zerodest <= 1'b0;
sevenhead <= 1'b0;
seventail <= 1'b0;
sevendest <= 1'b0;
end//header lost
            `else
              `ifdef TAIL_HT
                        begin
fivetail <= 1'b1;
fivedest <= 1'b0;
fivehead <= 1'b0;
zerotail <= 1'b1;
zerohead <= 1'b0;
zerodest <= 1'b0;
sevenhead <= 1'b0;
seventail <= 1'b0;
```

```
sevendest <= 1'b0;
end //tail lost
            `else
                begin
fivetail <= 1'b0;
fivedest <= 1'b0;
fivehead <= 1'b0;
zerotail <= 1'b0;
zerohead <= 1'b0;
zerodest <= 1'b0;
sevenhead <= 1'b0;
seventail <= 1'b0;
sevendest <= 1'b0;
end
            `endif //TAIL_HT
          `endif //HEAD_HT
        `endif //DEST_HT
`else
      `ifdef HT3
              `ifdef DEST_HT
                begin
fivetail <= 1'b0;
fivedest <= 1'b1;
```

```
fivehead <= 1'b0;
zerotail <= 1'b0;
zerohead <= 1'b0;
zerodest <= 1'b1;
sevenhead <= 1'b0;
seventail <= 1'b0;
sevendest <= 1'b1;
end
                `else
                 `ifdef HEAD_HT
                  begin
fivetail <= 1'b0;
fivedest <= 1'b0;
fivehead <= 1'b1;
zerotail <= 1'b0;
zerohead <= 1'b1;
zerodest <= 1'b0;
sevenhead <= 1'b1;
seventail <= 1'b0;
sevendest <= 1'b0;
end//header lost
                `else
                 `ifdef TAIL_HT
```

```
                begin
fivetail <= 1'b1;
fivedest <= 1'b0;
fivehead <= 1'b0;
zerotail <= 1'b1;
zerohead <= 1'b0;
zerodest <= 1'b0;
sevenhead <= 1'b0;
seventail <= 1'b1;
sevendest <= 1'b0;
end//tail lost
                `else
                    begin
fivetail <= 1'b0;
fivedest <= 1'b0;
fivehead <= 1'b0;
zerotail <= 1'b0;
zerohead <= 1'b0;
zerodest <= 1'b0;
sevenhead <= 1'b0;
seventail <= 1'b0;
sevendest <= 1'b0;
end
```

```
                `endif //TAIL_HT
               `endif //HEAD_HT
              `endif //DEST_HT
        `else
                begin
        fivetail <= 1'b0;
        fivedest <= 1'b0;
        fivehead <= 1'b0;
        zerotail <= 1'b0;
        zerohead <= 1'b0;
        zerodest <= 1'b0;
        sevenhead <= 1'b0;
        seventail <= 1'b0;
        sevendest <= 1'b0;
        end
            `endif//HT3
       `endif//HT2
    `endif//HT1*/


links0 = 0;
links1 = 0;
links2 = 0;
```

```
links3 = 0;

links4 = 0;

links5 = 0;

links6 = 0;

links7 = 0;

links8 = 0;

links9 = 0;

links10 = 0;

links11 = 0;

links12 = 0;

links13 = 0;

links14 = 0;

links15 = 0;

links16 = 0;

links17 = 0;

links18 = 0;

links19 = 0;

links20 = 0;

links21 = 0;

links22 = 0;

links23 = 0;

links24 = 0;

links25 = 0;
```

```
links26 = 0;
links27 = 0;
links28 = 0;
links29 = 0;
links30 = 0;
links31 = 0;
links32 = 0;
links33 = 0;
links34 = 0;
links35 = 0;
links36 = 0;
links37 = 0;
links38 = 0;
links39 = 0;
links40 = 0;
links41 = 0;
links42 = 0;
links43 = 0;
links44 = 0;
links45 = 0;
links46 = 0;
links47 = 0;
end
```

```
        always @(data_00  or almost_full_00)
begin
  if((|data_00 >0) | (almost_full_00))
     links0 = links0  + 1;
  else
     links0 = links0 ;
end

 always @(data_01  or almost_full_01)
begin
  if((|data_01 >0) | (almost_full_01))
     links1 = links1  + 1;
  else
     links1 = links1 ;
end

 always @(data_10  or almost_full_10)
begin
  if((|data_10 >0) | (almost_full_10))
     links2 = links2  + 1;
  else
```

```
    links2 = links2 ;
end

 always @(data_11  or almost_full_11)
begin
  if((|data_11 >0) | (almost_full_11))
    links3 = links3  + 1;
  else
    links3 = links3 ;
end

 always @(data_12  or almost_full_12)
begin
  if((|data_12 >0) | (almost_full_12))
    links4 = links4  + 1;
  else
    links4 = links4 ;
end

 always @(data_20  or almost_full_20)
begin
  if((|data_20 >0) | (almost_full_20))
    links5 = links5  + 1;
```

```
  else
     links5 = links5 ;
end

 always @(data_21  or almost_full_21)
begin
  if((|data_21 >0) | (almost_full_21))
     links6 = links6  + 1;
  else
     links6 = links6 ;
end

 always @(data_22  or almost_full_22)
begin
  if((|data_22 >0) | (almost_full_22))
     links7 = links7  + 1;
  else
     links7 = links7 ;
end

 always @(data_30  or almost_full_30)
begin
  if((|data_30 >0) | (almost_full_30))
```

```
    links8 = links8  + 1;
  else
    links8 = links8 ;
end

 always @(data_31  or almost_full_31)
begin
  if((|data_31 >0) | (almost_full_31))
    links9 = links9  + 1;
  else
    links9 = links9 ;
end

 always @(data_33  or almost_full_33)
begin
  if((|data_33 >0) | (almost_full_33))
    links10 = links10  + 1;
  else
    links10 = links10 ;
end

 always @(data_40  or almost_full_40)
begin
```

```
  if((|data_40 >0) | (almost_full_40))
     links11 = links11  + 1;
  else
     links11 = links11 ;
end

 always @(data_41  or almost_full_41)
begin
  if((|data_41 >0) | (almost_full_41))
     links12 = links12  + 1;
  else
     links12 = links12 ;
end

 always @(data_42  or almost_full_42)
begin
  if((|data_42 >0) | (almost_full_42))
     links13 = links13  + 1;
  else
     links13 = links13 ;
end

 always @(data_43  or almost_full_43)
```

```
begin
  if((|data_43 >0) | (almost_full_43))
     links14 = links14  + 1;
  else
     links14 = links14 ;
end

 always @(data_50  or almost_full_50)
begin
  if((|data_50 >0) | (almost_full_50))
     links15 = links15  + 1;
  else
     links15 = links15 ;
end

 always @(data_51  or almost_full_51)
begin
  if((|data_51 >0) | (almost_full_51))
     links16 = links16  + 1;
  else
     links16 = links16 ;
end
```

```
 always @(data_52  or almost_full_52)
begin
  if((|data_52 >0) | (almost_full_52))
     links17 = links17  + 1;
  else
     links17 = links17 ;
end

 always @(data_53  or almost_full_53)
begin
  if((|data_53 >0) | (almost_full_53))
     links18 = links18  + 1;
  else
     links18 = links18 ;
end

 always @(data_60  or almost_full_60)
begin
  if((|data_60 >0) | (almost_full_60))
     links19 = links19  + 1;
  else
     links19 = links19 ;
end
```

```
 always @(data_61  or almost_full_61)
begin
  if((|data_61 >0) | (almost_full_61))
     links20 = links20  + 1;
  else
     links20 = links20 ;
end

 always @(data_63  or almost_full_63)
begin
  if((|data_63 >0) | (almost_full_63))
     links21 = links21  + 1;
  else
     links21 = links21 ;
end

 always @(data_70  or almost_full_70)
begin
  if((|data_70 >0) | (almost_full_70))
     links22 = links22  + 1;
  else
     links22 = links22 ;
```

```
end

 always @(data_71  or almost_full_71)
begin
  if((|data_71 >0) | (almost_full_71))
     links23 = links23  + 1;
  else
     links23 = links23 ;
end

 always @(data_72  or almost_full_72)
begin
  if((|data_72 >0) | (almost_full_72))
     links24 = links24  + 1;
  else
     links24 = links24 ;
end

 always @(data_73  or almost_full_73)
begin
  if((|data_73 >0) | (almost_full_73))
     links25 = links25  + 1;
  else
```

```
    links25 = links25 ;
end

 always @(data_80  or almost_full_80)
begin
  if((|data_80 >0) | (almost_full_80))
    links26 = links26  + 1;
  else
    links26 = links26 ;
end

 always @(data_81  or almost_full_81)
begin
  if((|data_81 >0) | (almost_full_81))
    links27 = links27  + 1;
  else
    links27 = links27 ;
end

 always @(data_82  or almost_full_82)
begin
  if((|data_82 >0) | (almost_full_82))
    links28 = links28  + 1;
```

```
  else
     links28 = links28 ;
end

 always @(data_83  or almost_full_83)
begin
  if((|data_83 >0) | (almost_full_83))
     links29 = links29  + 1;
  else
     links29 = links29 ;
end

 always @(data_91  or almost_full_91)
begin
  if((|data_91 >0) | (almost_full_91))
     links30 = links30  + 1;
  else
     links30 = links30 ;
end

 always @(data_92  or almost_full_92)
begin
  if((|data_92 >0) | (almost_full_92))
```

```
    links31 = links31  + 1;
  else
    links31 = links31 ;
end

 always @(data_101  or almost_full_101)
begin
  if((|data_101 >0) | (almost_full_101))
    links32 = links32  + 1;
  else
    links32 = links32 ;
end

 always @(data_102  or almost_full_102)
begin
  if((|data_102 >0) | (almost_full_102))
    links33 = links33  + 1;
  else
    links33 = links33 ;
end

 always @(data_103  or almost_full_103)
begin
```

```
  if((|data_103 >0) | (almost_full_103))
     links34 = links34  + 1;
  else
     links34 = links34 ;
end

 always @(data_111  or almost_full_111)
begin
  if((|data_111 >0) | (almost_full_111))
     links35 = links35  + 1;
  else
     links35 = links35 ;
end

 always @(data_112  or almost_full_112)
begin
  if((|data_112 >0) | (almost_full_112))
     links36 = links36  + 1;
  else
     links36 = links36 ;
end

 always @(data_113  or almost_full_113)
```

```
begin
  if((|data_113 >0) | (almost_full_113))
     links37 = links37  + 1;
  else
     links37 = links37 ;
end

 always @(data_120  or almost_full_120)
begin
  if((|data_120 >0) | (almost_full_120))
     links38 = links38  + 1;
  else
     links38 = links38 ;
end

 always @(data_123  or almost_full_123)
begin
  if((|data_123 >0) | (almost_full_123))
     links39 = links39  + 1;
  else
     links39 = links39 ;
end
```

```
 always @(data_130  or almost_full_130)
begin
  if((|data_130 >0) | (almost_full_130))
     links40 = links40  + 1;
  else
     links40 = links40 ;
end

 always @(data_132  or almost_full_132)
begin
  if((|data_132 >0) | (almost_full_132))
     links41 = links41  + 1;
  else
     links41 = links41 ;
end

 always @(data_133  or almost_full_133)
begin
  if((|data_133 >0) | (almost_full_133))
     links42 = links42  + 1;
  else
     links42 = links42 ;
end
```

```
 always @(data_140  or almost_full_140)
begin
  if((|data_140 >0) | (almost_full_140))
     links43 = links43  + 1;
  else
     links43 = links43 ;
end

 always @(data_142  or almost_full_142)
begin
  if((|data_142 >0) | (almost_full_142))
     links44 = links44  + 1;
  else
     links44 = links44 ;
end

 always @(data_143  or almost_full_143)
begin
  if((|data_143 >0) | (almost_full_143))
     links45 = links45  + 1;
  else
     links45 = links45 ;
```

```
end

 always @(data_152  or almost_full_152)
begin
  if((|data_152 >0) | (almost_full_152))
     links46 = links46  + 1;
  else
     links46 = links46 ;
end

 always @(data_153  or almost_full_153)
begin
  if((|data_153 >0) | (almost_full_153))
     links47 = links47  + 1;
  else
     links47 = links47 ;
end



initial
begin
op_file0 = $fopen(filename0, "w");
```

```
op_file1 = $fopen(filename1, "w");
op_file2 = $fopen(filename2, "w");
op_file3 = $fopen(filename3, "w");
op_file4 = $fopen(filename4, "w");
op_file5 = $fopen(filename5, "w");
op_file6 = $fopen(filename6, "w");
op_file7 = $fopen(filename7, "w");
op_file8 = $fopen(filename8, "w");
op_file9 = $fopen(filename9, "w");
op_file10 = $fopen(filename10, "w");
op_file11 = $fopen(filename11, "w");
op_file12 = $fopen(filename12, "w");
op_file13 = $fopen(filename13, "w");
op_file14 = $fopen(filename14, "w");
op_file15 = $fopen(filename15, "w");
op_file16 = $fopen(filename16, "w");
op_file17 = $fopen(filename17, "w");
op_file18 = $fopen(filename18, "w");
op_file19 = $fopen(filename19, "w");

r_file0 = $fopen("SecureZoneTraceR0.txt","r");
r_file1 = $fopen("SecureZoneTraceR1.txt","r");
r_file2 = $fopen("SecureZoneTraceR2.txt","r");
```

```
r_file3 = $fopen("SecureZoneTraceR3.txt","r");
r_file4 = $fopen("SecureZoneTraceR4.txt","r");
r_file5 = $fopen("SecureZoneTraceR5.txt","r");
r_file6 = $fopen("SecureZoneTraceR6.txt","r");
r_file7 = $fopen("SecureZoneTraceR7.txt","r");
r_file8 = $fopen("SecureZoneTraceR8.txt","r");
r_file9 = $fopen("SecureZoneTraceR9.txt","r");
r_file10 = $fopen("SecureZoneTraceR10.txt","r");
r_file11 = $fopen("SecureZoneTraceR11.txt","r");
r_file12 = $fopen("SecureZoneTraceR12.txt","r");
r_file13 = $fopen("SecureZoneTraceR13.txt","r");
r_file14 = $fopen("SecureZoneTraceR14.txt","r");
r_file15 = $fopen("SecureZoneTraceR15.txt","r");

end

initial
begin
  CLK = 1'b0;
  CLK2 = 1'b0;
            reset = 1'b1;
            Count = 14'd0;
            trigger = 1'b0;
```

```
NACK_in_E = 1'b0;

NACK_in_S = 1'b0;

NACK_in_W = 1'b0;

NACK_in_N = 1'b0;

NACK_in_L = 1'b0;

BW = 0; //measures bandwidth x/48

ps0 = 0;

ps1 = 0;

ps2 = 0;

ps3 = 0;

ps4 = 0;

ps5 = 0;

ps6 = 0;

ps7 = 0;

ps8 = 0;

ps9 = 0;

ps10 = 0;

ps11 = 0;

ps12 = 0;

ps13 = 0;

ps14 = 0;

ps15 = 0;

pstotal = 0;
```

```
src0 = 4'b0000;

src1 = 4'b0001;

src2 = 4'b0010;

src3 = 4'b0011;

src4 = 4'b0100;

src5 = 4'b0101;

src6 = 4'b0110;

src7 = 4'b0111;

src8 = 4'b1000;

src9 = 4'b1001;

src10 = 4'b1010;

src11 = 4'b1011;

src12 = 4'b1100;

src13 = 4'b1101;

src14 = 4'b1110;

src15 = 4'b1111;

Dest0 = 5'd0;

Dest1 = 5'd0;

Dest2 = 5'd0;

Dest3 = 5'd0;

Dest4 = 5'd0;

Dest5 = 5'd0;

Dest6 = 5'd0;
```

```
Dest7 = 5'd0;

Dest8 = 5'd0;

Dest9 = 5'd0;

Dest10 = 5'd0;

Dest11 = 5'd0;

Dest12 = 5'd0;

Dest13 = 5'd0;

Dest14 = 5'd0;

Dest15 = 5'd0;

D0 = 4'd0;

D1 = 4'd0;

D2 = 4'd0;

D3 = 4'd0;

D4 = 4'd0;

D5 = 4'd0;

D6 = 4'd0;

D7 = 4'd0;

D8 = 4'd0;

D9 = 4'd0;

D10 = 4'd0;

D11 = 4'd0;

D12 = 4'd0;

D13 = 4'd0;
```

```
D14 = 4'd0;

D15 = 4'd0;

spackets0 = 0;

spackets1 = 0;

spackets2 = 0;

spackets3 = 0;

spackets4 = 0;

spackets5 = 0;

spackets6 = 0;

spackets7 = 0;

spackets8 = 0;

spackets9 = 0;

spackets10 = 0;

spackets11 = 0;

spackets12 = 0;

spackets13 = 0;

spackets14 = 0;

spackets15 = 0;

rpackets0 = 0;

rpackets1 = 0;

rpackets2 = 0;

rpackets3 = 0;

rpackets4 = 0;
```

```
rpackets5 = 0;
rpackets6 = 0;
rpackets7 = 0;
rpackets8 = 0;
rpackets9 = 0;
rpackets10 = 0;
rpackets11 = 0;
rpackets12 = 0;
rpackets13 = 0;
rpackets14 = 0;
rpackets15 = 0;
inreq0 = 1'b0;
inreq1 = 1'b0;
inreq2 = 1'b0;
inreq3 = 1'b0;
inreq4 = 1'b0;
inreq5 = 1'b0;
inreq6 = 1'b0;
inreq7 = 1'b0;
inreq8 = 1'b0;
inreq9 = 1'b0;
inreq10 = 1'b0;
inreq11 = 1'b0;
```

```
inreq12 = 1'b0;
inreq13 = 1'b0;
inreq14 = 1'b0;
inreq15 = 1'b0;
ID0 = 8'h00;
ID1 = 8'h00;
ID2 = 8'h00;
ID3 = 8'h00;
ID4 = 8'h00;
ID5 = 8'h00;
ID6 = 8'h00;
ID7 = 8'h00;
ID8 = 8'h00;
ID9 = 8'h00;
ID10 = 8'h00;
ID11 = 8'h00;
ID12 = 8'h00;
ID13 = 8'h00;
ID14 = 8'h00;
ID15 = 8'h00;
Info0 = 10'd0;
Info1 = 10'd0;
Info2 = 10'd0;
```

```
Info3 = 10'd0;
Info4 = 10'd0;
Info5 = 10'd0;
Info6 = 10'd0;
Info7 = 10'd0;
Info8 = 10'd0;
Info9 = 10'd0;
Info10 = 10'd0;
Info11 = 10'd0;
Info12 = 10'd0;
Info13 = 10'd0;
Info14 = 10'd0;
Info15 = 10'd0;
corr0 = 0;
corr1 = 0;
corr2 = 0;
corr3 = 0;
corr4 = 0;
corr5 = 0;
corr6 = 0;
corr7 = 0;
corr8 = 0;
corr9 = 0;
```

```
        corr10 = 0;
        corr11 = 0;
        corr12 = 0;
        corr13 = 0;
        corr14 = 0;
        corr15 = 0;
        corrtotal = 0;
        rtotal = 0;
        #900
        reset = 1'b0; //initialization done, start tests
        packetsreceived = 0;
        packetsSent = 0;
  trigger = 1'b1;
        #8501000;
        rtotal =
rpackets0+rpackets1+rpackets2+rpackets3+rpackets4+rpackets5+rpackets6+rpackets7+rpackets8+
rpackets9+rpackets10+rpackets11+rpackets12+rpackets13+rpackets14+rpackets15;
        corrtotal =
corr0+corr1+corr2+corr3+corr4+corr5+corr6+corr7+corr8+corr9+corr10+corr11+corr12+corr13+
corr14+corr15;
        $fwrite(op_file16, "%d \n %d \n %d \n %d \n %d \n %d \n %d \n %d \n %d \n %d \n %d \n
%d \n %d \n %d \n %d \n %d \n %d \n %d \n  %d \n %d \n %d \n %d \n %d \n %d \n %d \n %d \n
%d \n %d \n %d \n %d \n %d \n %d \n %d \n %d \n %d \n %d \n %d \n %d \n %d \n %d \n %d \n
```

```
%d \n %d \n %d \n %d \n %d \n %d \n %d \n %d \n %d \n %d \n" ,ps0, ps1, ps2, ps3, ps4, ps5, ps6, ps7, ps8, ps9, ps10, ps11, ps12, ps13, ps14, ps15, pstotal, rpackets0, rpackets1, rpackets2, rpackets3, rpackets4, rpackets5, rpackets6, rpackets7, rpackets8, rpackets9, rpackets10, rpackets11, rpackets12, rpackets13, rpackets14, rpackets15, rtotal,corr0,corr1,corr2,corr3,corr4,corr5,corr6,corr7,corr8,corr9,corr10,corr11,corr12,corr13,corr14,corr15, corrtotal);
	$fwrite(op_file19, "%d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n %d\n ",links0, links1, links2, links3, links4, links5, links6, links7, links8, links9, links10, links11, links12, links13, links14, links15, links16, links17, links18, links19, links20, links21, links22, links23, links24, links25, links26, links27, links28, links29, links30, links31, links32, links33, links34, links35, links36, links37, links38, links39, links40, links41, links42, links43, links44, links45, links46, links47);
	#100
	$fclose(op_file0);
	$fclose(op_file1);
	$fclose(op_file2);
	$fclose(op_file3);
	$fclose(op_file4);
	$fclose(op_file5);
	$fclose(op_file6);
	$fclose(op_file7);
```

```
$fclose(op_file8);
$fclose(op_file9);
$fclose(op_file10);
$fclose(op_file11);
$fclose(op_file12);
$fclose(op_file13);
$fclose(op_file14);
$fclose(op_file15);
$fclose(op_file16);
$fclose(op_file17);
$fclose(op_file18);
$fclose(op_file19);
$fclose(r_file0);
$fclose(r_file1);
$fclose(r_file2);
$fclose(r_file3);
$fclose(r_file4);
$fclose(r_file5);
$fclose(r_file6);
$fclose(r_file7);
$fclose(r_file8);
$fclose(r_file9);
$fclose(r_file10);
```

```
        $fclose(r_file11);
        $fclose(r_file12);
        $fclose(r_file13);
        $fclose(r_file14);
        $fclose(r_file15);
        $finish;
end

always
#injection CLK2 <= ~CLK2;

always@(posedge CLK2)
begin
        code0 = $fscanf(r_file0, "%dddd", Destr0);
        code0 = $fseek(r_file0, 1, 1);
        code1 = $fscanf(r_file1, "%dddd", Destr1);
        code1 = $fseek(r_file1, 1, 1);
        code2 = $fscanf(r_file2, "%dddd", Destr2);
        code2 = $fseek(r_file2, 1, 1);
        code3 = $fscanf(r_file3, "%dddd", Destr3);
        code3 = $fseek(r_file3, 1, 1);
        code4 = $fscanf(r_file4, "%dddd", Destr4);
        code4 = $fseek(r_file4, 1, 1);
```

```
        code5 = $fscanf(r_file5, "%dddd", Destr5);
        code5 = $fseek(r_file5, 1, 1);
        code6 = $fscanf(r_file6, "%dddd", Destr6);
        code6 = $fseek(r_file6, 1, 1);
        code7 = $fscanf(r_file7, "%dddd", Destr7);
        code7 = $fseek(r_file7, 1, 1);
        code8 = $fscanf(r_file8, "%dddd", Destr8);
        code8 = $fseek(r_file8, 1, 1);
        code9 = $fscanf(r_file9, "%dddd", Destr9);
        code9 = $fseek(r_file9, 1, 1);
        code10 = $fscanf(r_file10, "%dddd", Destr10);
        code10 = $fseek(r_file10, 1, 1);
        code11 = $fscanf(r_file11, "%dddd", Destr11);
        code11 = $fseek(r_file11, 1, 1);
        code12 = $fscanf(r_file12, "%dddd", Destr12);
        code12 = $fseek(r_file12, 1, 1);
        code13 = $fscanf(r_file13, "%dddd", Destr13);
        code13 = $fseek(r_file13, 1, 1);
        code14 = $fscanf(r_file14, "%dddd", Destr14);
        code14 = $fseek(r_file14, 1, 1);
        code15 = $fscanf(r_file15, "%dddd", Destr15);
        code15 = $fseek(r_file15, 1, 1);
end
```

```
always@(Destr0)
  begin
    if((Destr0 > 0)&&(Destr0 !== 100))
      begin
        Dest0 = Destr0/100;
        case(Dest0)
        5'd1: begin D0 = 4'b0000; ps0 = ps0; end
        5'd2: begin D0 = 4'b0001; ps1 = ps1 +1; end
        5'd3: begin D0 = 4'b0010; ps2 = ps2 +1; end
        5'd4: begin D0 = 4'b0011; ps3 = ps3 +1; end
        5'd5: begin D0 = 4'b0100; ps4 = ps4 +1; end
        5'd6: begin D0 = 4'b0101; ps5 = ps5 +1; end
        5'd7: begin D0 = 4'b0110; ps6 = ps6 +1; end
        5'd8: begin D0 = 4'b0111; ps7 = ps7 +1; end
        5'd9: begin D0 = 4'b1000; ps8 = ps8 +1; end
        5'd10: begin D0 = 4'b1001; ps9 = ps9 +1; end
        5'd11: begin D0 = 4'b1010; ps10 = ps10 +1; end
        5'd12: begin D0 = 4'b1011; ps11 = ps11 +1; end
        5'd13: begin D0 = 4'b1100; ps12 = ps12 +1; end
        5'd14: begin D0 = 4'b1101; ps13 = ps13 +1; end
        5'd15: begin D0 = 4'b1110; ps14 = ps14 +1; end
        5'd16: begin D0 = 4'b1111; ps15 = ps15 +1; end
```

```
        default: D0 = 4'b0000;
        endcase
    end
    else
      begin
        Dest0 = 5'd0;
        D0 = 4'b0000;
      end
  end
always@(Dest0)
begin
#5;
if((D0 !== 4'b0000)&&(Dest0 !== 5'd0))
begin
@(negedge CLK)
Info0[9:0] = {D0[3:0], src0[3:0], 2'b01};
@(posedge CLK)
Info0[9:0] = {D0[3:0], src0[3:0], 2'b11};
@(posedge CLK)
Info0[9:0] = {D0[3:0], src0[3:0], 2'b11};
@(posedge CLK)
Info0[9:0] = {D0[3:0], src0[3:0], 2'b10};
@(posedge CLK)
```

```
Info0[9:0] = {4'b0000, src0[3:0], 2'b00};
spackets0 = spackets0 + 1;
end
else
Info0[9:0] = 10'd0;//Info0[9:0] = {D0[3:0],src0[3:0], 2'b00};
end
//1
always@(Destr1)
  begin
    if((Destr1 > 0)&&(Destr1 !== 200))
      begin
        Dest1 = Destr1/100;
        case(Dest1)
        5'd1: begin D1 = 4'b0000; ps0 = ps0 +1; end
        5'd2: begin D1 = 4'b0001; ps1 = ps1; end
        5'd3: begin D1 = 4'b0010; ps2 = ps2 +1; end
        5'd4: begin D1 = 4'b0011; ps3 = ps3 +1; end
        5'd5: begin D1 = 4'b0100; ps4 = ps4 +1; end
        5'd6: begin D1 = 4'b0101; ps5 = ps5 +1; end
        5'd7: begin D1 = 4'b0110; ps6 = ps6 +1; end
        5'd8: begin D1 = 4'b0111; ps7 = ps7 +1; end
        5'd9: begin D1 = 4'b1000; ps8 = ps8 +1; end
        5'd10: begin D1 = 4'b1001; ps9 = ps9 +1; end
```

```
        5'd11: begin D1 = 4'b1010; ps10 = ps10 +1; end
        5'd12: begin D1 = 4'b1011; ps11 = ps11 +1; end
        5'd13: begin D1 = 4'b1100; ps12 = ps12 +1; end
        5'd14: begin D1 = 4'b1101; ps13 = ps13 +1; end
        5'd15: begin D1 = 4'b1110; ps14 = ps14 +1; end
        5'd16: begin D1 = 4'b1111; ps15 = ps15 +1; end
        default: D1 = 4'b0000;
        endcase
    end
    else
      begin
        Dest1 = 5'd0;
        D1 = 4'b0000;
      end
  end
always@(Dest1)
begin
#5;
if((D1 !== 4'b0001)&&(Dest1 !== 5'd0))
begin
@(negedge CLK)
Info1[9:0] = {D1[3:0], src1[3:0], 2'b01};
@(posedge CLK)
```

```
Info1[9:0] = {D1[3:0], src1[3:0], 2'b11};
@(posedge CLK)
Info1[9:0] = {D1[3:0], src1[3:0], 2'b11};
@(posedge CLK)
Info1[9:0] = {D1[3:0], src1[3:0], 2'b10};
@(posedge CLK)
Info1[9:0] = {4'b0000, src1[3:0], 2'b00};
spackets1 = spackets1 + 1;
end
else
Info1[9:0] = 10'd0;//Info1[9:0] = {D1[3:0],src1[3:0], 2'b00};
end
//2
always@(Destr2)
  begin
    if((Destr2 > 0)&&(Destr2 !== 300))
      begin
        Dest2 = Destr2/100;
        case(Dest2)
        5'd1: begin D2 = 4'b0000; ps0 = ps0 +1; end
        5'd2: begin D2 = 4'b0001; ps1 = ps1 +1; end
        5'd3: begin D2 = 4'b0010; ps2 = ps2; end
        5'd4: begin D2 = 4'b0011; ps3 = ps3 +1; end
```

```
        5'd5: begin D2 = 4'b0100; ps4 = ps4 +1; end
        5'd6: begin D2 = 4'b0101; ps5 = ps5 +1; end
        5'd7: begin D2 = 4'b0110; ps6 = ps6 +1; end
        5'd8: begin D2 = 4'b0111; ps7 = ps7 +1; end
        5'd9: begin D2 = 4'b1000; ps8 = ps8 +1; end
        5'd10: begin D2 = 4'b1001; ps9 = ps9 +1; end
        5'd11: begin D2 = 4'b1010; ps10 = ps10 +1; end
        5'd12: begin D2 = 4'b1011; ps11 = ps11 +1; end
        5'd13: begin D2 = 4'b1100; ps12 = ps12 +1; end
        5'd14: begin D2 = 4'b1101; ps13 = ps13 +1; end
        5'd15: begin D2 = 4'b1110; ps14 = ps14 +1; end
        5'd16: begin D2 = 4'b1111; ps15 = ps15 +1; end
        default: D2 = 4'b0000;
        endcase
    end
    else
      begin
        Dest2 = 5'd0;
        D2 = 4'b0000;
      end
  end
always@(Dest2)
begin
```

```
#5;
if((D2 !== 4'b0010)&&(Dest2 !== 5'd0))
begin
@(negedge CLK)
Info2[9:0] = {D2[3:0], src2[3:0], 2'b01};
@(posedge CLK)
Info2[9:0] = {D2[3:0], src2[3:0], 2'b11};
@(posedge CLK)
Info2[9:0] = {D2[3:0], src2[3:0], 2'b11};
@(posedge CLK)
Info2[9:0] = {D2[3:0], src2[3:0], 2'b10};
@(posedge CLK)
Info2[9:0] = {4'b0000, src2[3:0], 2'b00};
spackets2 = spackets2 + 1;
end
else
Info2[9:0] = 10'd0;
end
//3
always@(Destr3)
  begin
    if((Destr3 > 0)&&(Destr3 !== 400))
      begin
```

```
    Dest3 = Destr3/100;
    case(Dest3)
    5'd1: begin D3 = 4'b0000; ps0 = ps0 +1; end
    5'd2: begin D3 = 4'b0001; ps1 = ps1 +1; end
    5'd3: begin D3 = 4'b0010; ps2 = ps2 +1; end
    5'd4: begin D3 = 4'b0011; ps3 = ps3; end
    5'd5: begin D3 = 4'b0100; ps4 = ps4 +1; end
    5'd6: begin D3 = 4'b0101; ps5 = ps5 +1; end
    5'd7: begin D3 = 4'b0110; ps6 = ps6 +1; end
    5'd8: begin D3 = 4'b0111; ps7 = ps7 +1; end
    5'd9: begin D3 = 4'b1000; ps8 = ps8 +1; end
    5'd10: begin D3 = 4'b1001; ps9 = ps9 +1; end
    5'd11: begin D3 = 4'b1010; ps10 = ps10 +1; end
    5'd12: begin D3 = 4'b1011; ps11 = ps11 +1; end
    5'd13: begin D3 = 4'b1100; ps12 = ps12 +1; end
    5'd14: begin D3 = 4'b1101; ps13 = ps13 +1; end
    5'd15: begin D3 = 4'b1110; ps14 = ps14 +1; end
    5'd16: begin D3 = 4'b1111; ps15 = ps15 +1; end
    default: D3 = 4'b0000;
    endcase
end
else
  begin
```

```
        Dest3 = 5'd0;
        D3 = 4'b0000;
      end
  end
always@(Dest3)
begin
#5;
if((D3 !== 4'b0011)&&(Dest3 !== 5'd0))
begin
@(negedge CLK)
Info3[9:0] = {D3[3:0], src3[3:0], 2'b01};
@(posedge CLK)
Info3[9:0] = {D3[3:0], src3[3:0], 2'b11};
@(posedge CLK)
Info3[9:0] = {D3[3:0], src3[3:0], 2'b11};
@(posedge CLK)
Info3[9:0] = {D3[3:0], src3[3:0], 2'b10};
@(posedge CLK)
Info3[9:0] = {4'b0000, src3[3:0], 2'b00};
spackets3 = spackets3 + 1;
end
else
Info3[9:0] = 10'd0;
```

```
end
//4
always@(Destr4)
  begin
    if((Destr4 > 0)&&(Destr4 !== 500))
      begin
        Dest4 = Destr4/100;
        case(Dest4)
        5'd1: begin D4 = 4'b0000; ps0 = ps0 +1; end
        5'd2: begin D4 = 4'b0001; ps1 = ps1 +1; end
        5'd3: begin D4 = 4'b0010; ps2 = ps2 +1; end
        5'd4: begin D4 = 4'b0011; ps3 = ps3 +1; end
        5'd5: begin D4 = 4'b0100; ps4 = ps4; end
        5'd6: begin D4 = 4'b0101; ps5 = ps5 +1; end
        5'd7: begin D4 = 4'b0110; ps6 = ps6 +1; end
        5'd8: begin D4 = 4'b0111; ps7 = ps7 +1; end
        5'd9: begin D4 = 4'b1000; ps8 = ps8 +1; end
        5'd10: begin D4 = 4'b1001; ps9 = ps9 +1; end
        5'd11: begin D4 = 4'b1010; ps10 = ps10 +1; end
        5'd12: begin D4 = 4'b1011; ps11 = ps11 +1; end
        5'd13: begin D4 = 4'b1100; ps12 = ps12 +1; end
        5'd14: begin D4 = 4'b1101; ps13 = ps13 +1; end
        5'd15: begin D4 = 4'b1110; ps14 = ps14 +1; end
```

```
        5'd16: begin D4 = 4'b1111; ps15 = ps15 +1; end
        default: D4 = 4'b0000;
        endcase
    end
    else
      begin
        Dest4 = 5'd0;
        D4 = 4'b0000;
      end
  end
always@(Dest4)
begin
#5;
if((D4 !== 4'b0100)&&(Dest4 !== 5'd0))
begin
@(negedge CLK)
Info4[9:0] = {D4[3:0], src4[3:0], 2'b01};
@(posedge CLK)
Info4[9:0] = {D4[3:0], src4[3:0], 2'b11};
@(posedge CLK)
Info4[9:0] = {D4[3:0], src4[3:0], 2'b11};
@(posedge CLK)
Info4[9:0] = {D4[3:0], src4[3:0], 2'b10};
```

```
@(posedge CLK)
Info4[9:0] = {4'b0000, src4[3:0], 2'b00};
spackets4 = spackets4 + 1;
end
else
Info4[9:0] = 10'd0;
end
//5
always@(Destr5)
  begin
    if((Destr5 > 0)&&(Destr5 !== 600))
      begin
        Dest5 = Destr5/100;
        case(Dest5)
        5'd1: begin D5 = 4'b0000; ps0 = ps0 +1; end
        5'd2: begin D5 = 4'b0001; ps1 = ps1 +1; end
        5'd3: begin D5 = 4'b0010; ps2 = ps2 +1; end
        5'd4: begin D5 = 4'b0011; ps3 = ps3 +1; end
        5'd5: begin D5 = 4'b0100; ps4 = ps4 +1; end
        5'd6: begin D5 = 4'b0101; ps5 = ps5; end
        5'd7: begin D5 = 4'b0110; ps6 = ps6 +1; end
        5'd8: begin D5 = 4'b0111; ps7 = ps7 +1; end
        5'd9: begin D5 = 4'b1000; ps8 = ps8 +1; end
```

```
        5'd10: begin D5 = 4'b1001; ps9 = ps9 +1; end
        5'd11: begin D5 = 4'b1010; ps10 = ps10 +1; end
        5'd12: begin D5 = 4'b1011; ps11 = ps11 +1; end
        5'd13: begin D5 = 4'b1100; ps12 = ps12 +1; end
        5'd14: begin D5 = 4'b1101; ps13 = ps13 +1; end
        5'd15: begin D5 = 4'b1110; ps14 = ps14 +1; end
        5'd16: begin D5 = 4'b1111; ps15 = ps15 +1; end
        default: D5 = 4'b0000;
        endcase
    end
    else
      begin
        Dest5 = 5'd0;
        D5 = 4'b0000;
      end
  end
always@(Dest5)
begin
#5;
if((D5 !== 4'b0101)&&(Dest5 !== 5'd0))
begin
@(negedge CLK)
Info5[9:0] = {D5[3:0], src5[3:0], 2'b01};
```

```
@(posedge CLK)
Info5[9:0] = {D5[3:0], src5[3:0], 2'b11};
@(posedge CLK)
Info5[9:0] = {D5[3:0], src5[3:0], 2'b11};
@(posedge CLK)
Info5[9:0] = {D5[3:0], src5[3:0], 2'b10};
@(posedge CLK)
Info5[9:0] = {4'b0000, src5[3:0], 2'b00};
spackets5 = spackets5 + 1;


end
else
Info5[9:0] = 10'd0;
end
//6
always@((Destr6 > 0)&&(Destr6 !== 700))
  begin
    if(Destr6 > 0)
      begin
        Dest6 = Destr6/100;
        case(Dest6)
        5'd1: begin D6 = 4'b0000; ps0 = ps0 +1; end
        5'd2: begin D6 = 4'b0001; ps1 = ps1 +1; end
```

```
      5'd3: begin D6 = 4'b0010; ps2 = ps2 +1; end
      5'd4: begin D6 = 4'b0011; ps3 = ps3 +1; end
      5'd5: begin D6 = 4'b0100; ps4 = ps4 +1; end
      5'd6: begin D6 = 4'b0101; ps5 = ps5 +1; end
      5'd7: begin D6 = 4'b0110; ps6 = ps6; end
      5'd8: begin D6 = 4'b0111; ps7 = ps7 +1; end
      5'd9: begin D6 = 4'b1000; ps8 = ps8 +1; end
      5'd10: begin D6 = 4'b1001; ps9 = ps9 +1; end
      5'd11: begin D6 = 4'b1010; ps10 = ps10 +1; end
      5'd12: begin D6 = 4'b1011; ps11 = ps11 +1; end
      5'd13: begin D6 = 4'b1100; ps12 = ps12 +1; end
      5'd14: begin D6 = 4'b1101; ps13 = ps13 +1; end
      5'd15: begin D6 = 4'b1110; ps14 = ps14 +1; end
      5'd16: begin D6 = 4'b1111; ps15 = ps15 +1; end
      default: D6 = 4'b0000;
      endcase
   end
   else
     begin
      Dest6 = 5'd0;
      D6 = 4'b0000;
     end
end
```

```
always@(Dest6)
begin
#5;
if((D6 !== 4'b0110)&&(Dest6 !== 5'd0))
begin
@(negedge CLK)
Info6[9:0] = {D6[3:0], src6[3:0], 2'b01};
@(posedge CLK)
Info6[9:0] = {D6[3:0], src6[3:0], 2'b11};
@(posedge CLK)
Info6[9:0] = {D6[3:0], src6[3:0], 2'b11};
@(posedge CLK)
Info6[9:0] = {D6[3:0], src6[3:0], 2'b10};
@(posedge CLK)
Info6[9:0] = {4'b0000, src6[3:0], 2'b00};
spackets6 = spackets6 + 1;

end
else
Info6[9:0] = 10'd0;
end
//7
always@(Destr7)
```

```
begin
  if((Destr7 > 0)&&(Destr7 !== 800))
    begin
      Dest7 = Destr7/100;
      case(Dest7)
      5'd1: begin D7 = 4'b0000; ps0 = ps0 +1; end
      5'd2: begin D7 = 4'b0001; ps1 = ps1 +1; end
      5'd3: begin D7 = 4'b0010; ps2 = ps2 +1; end
      5'd4: begin D7 = 4'b0011; ps3 = ps3 +1; end
      5'd5: begin D7 = 4'b0100; ps4 = ps4 +1; end
      5'd6: begin D7 = 4'b0101; ps5 = ps5 +1; end
      5'd7: begin D7 = 4'b0110; ps6 = ps6 +1; end
      5'd8: begin D7 = 4'b0111; ps7 = ps7; end
      5'd9: begin D7 = 4'b1000; ps8 = ps8 +1; end
      5'd10: begin D7 = 4'b1001; ps9 = ps9 +1; end
      5'd11: begin D7 = 4'b1010; ps10 = ps10 +1; end
      5'd12: begin D7 = 4'b1011; ps11 = ps11 +1; end
      5'd13: begin D7 = 4'b1100; ps12 = ps12 +1; end
      5'd14: begin D7 = 4'b1101; ps13 = ps13 +1; end
      5'd15: begin D7 = 4'b1110; ps14 = ps14 +1; end
      5'd16: begin D7 = 4'b1111; ps15 = ps15 +1; end
      default: D7 = 4'b0000;
      endcase
```

```
    end
    else
      begin
        Dest7 = 5'd0;
        D7 = 4'b0000;
      end
  end
always@(Dest7)
begin
#5;
if((D7 !== 4'b0111)&&(Dest7 !== 5'd0))
begin
@(negedge CLK)
Info7[9:0] = {D7[3:0], src7[3:0], 2'b01};
@(posedge CLK)
Info7[9:0] = {D7[3:0], src7[3:0], 2'b11};
@(posedge CLK)
Info7[9:0] = {D7[3:0], src7[3:0], 2'b11};
@(posedge CLK)
Info7[9:0] = {D7[3:0], src7[3:0], 2'b10};
@(posedge CLK)
Info7[9:0] = {4'b0000, src7[3:0], 2'b00};
spackets7 = spackets7 + 1;
```

```
end
else
Info7[9:0] = 10'd0;
end
//8
always@(Destr8)
  begin
    if((Destr8 > 0)&&(Destr8 !== 900))
      begin
        Dest8 = Destr8/100;
        case(Dest8)
        5'd1: begin D8 = 4'b0000; ps0 = ps0 +1; end
        5'd2: begin D8 = 4'b0001; ps1 = ps1 +1; end
        5'd3: begin D8 = 4'b0010; ps2 = ps2 +1; end
        5'd4: begin D8 = 4'b0011; ps3 = ps3 +1; end
        5'd5: begin D8 = 4'b0100; ps4 = ps4 +1; end
        5'd6: begin D8 = 4'b0101; ps5 = ps5 +1; end
        5'd7: begin D8 = 4'b0110; ps6 = ps6 +1; end
        5'd8: begin D8 = 4'b0111; ps7 = ps7 +1; end
        5'd9: begin D8 = 4'b1000; ps8 = ps8; end
        5'd10: begin D8 = 4'b1001; ps9 = ps9 +1; end
        5'd11: begin D8 = 4'b1010; ps10 = ps10 +1; end
```

```
        5'd12: begin D8 = 4'b1011; ps11 = ps11 +1; end
        5'd13: begin D8 = 4'b1100; ps12 = ps12 +1; end
        5'd14: begin D8 = 4'b1101; ps13 = ps13 +1; end
        5'd15: begin D8 = 4'b1110; ps14 = ps14 +1; end
        5'd16: begin D8 = 4'b1111; ps15 = ps15 +1; end
        default: D8 = 4'b0000;
        endcase
    end
    else
      begin
        Dest8 = 5'd0;
        D8 = 4'b0000;
      end
  end
always@(Dest8)
begin
#5;
if((D8 !== 4'b1000)&&(Dest8 !== 5'd0))
begin
@(negedge CLK)
Info8[9:0] = {D8[3:0], src8[3:0], 2'b01};
@(posedge CLK)
Info8[9:0] = {D8[3:0], src8[3:0], 2'b11};
```

```
@(posedge CLK)
Info8[9:0] = {D8[3:0], src8[3:0], 2'b11};
@(posedge CLK)
Info8[9:0] = {D8[3:0], src8[3:0], 2'b10};
@(posedge CLK)
Info8[9:0] = {4'b0000, src8[3:0], 2'b00};
spackets8 = spackets8 + 1;


end
else
Info8[9:0] = 10'd0;
end
//9
always@(Destr9)
  begin
    if((Destr9 > 0)&&(Destr9 !== 1000))
      begin
        Dest9 = Destr9/100;
        case(Dest9)
        5'd1: begin D9 = 4'b0000; ps0 = ps0 +1; end
        5'd2: begin D9 = 4'b0001; ps1 = ps1 +1; end
        5'd3: begin D9 = 4'b0010; ps2 = ps2 +1; end
        5'd4: begin D9 = 4'b0011; ps3 = ps3 +1; end
```

```
        5'd5: begin D9 = 4'b0100; ps4 = ps4 +1; end
        5'd6: begin D9 = 4'b0101; ps5 = ps5 +1; end
        5'd7: begin D9 = 4'b0110; ps6 = ps6 +1; end
        5'd8: begin D9 = 4'b0111; ps7 = ps7 +1; end
        5'd9: begin D9 = 4'b1000; ps8 = ps8 +1; end
        5'd10: begin D9 = 4'b1001; ps9 = ps9; end
        5'd11: begin D9 = 4'b1010; ps10 = ps10 +1; end
        5'd12: begin D9 = 4'b1011; ps11 = ps11 +1; end
        5'd13: begin D9 = 4'b1100; ps12 = ps12 +1; end
        5'd14: begin D9 = 4'b1101; ps13 = ps13 +1; end
        5'd15: begin D9 = 4'b1110; ps14 = ps14 +1; end
        5'd16: begin D9 = 4'b1111; ps15 = ps15 +1; end
        default: D9 = 4'b0000;
        endcase
    end
    else
      begin
        Dest9 = 5'd0;
        D9 = 4'b0000;
      end
  end
always@(Dest9)
begin
```

```
#5;
if((D9 !== 4'b1001)&&(Dest9 !== 5'd0))
begin
@(negedge CLK)
Info9[9:0] = {D9[3:0], src9[3:0], 2'b01};
@(posedge CLK)
Info9[9:0] = {D9[3:0], src9[3:0], 2'b11};
@(posedge CLK)
Info9[9:0] = {D9[3:0], src9[3:0], 2'b11};
@(posedge CLK)
Info9[9:0] = {D9[3:0], src9[3:0], 2'b10};
@(posedge CLK)
Info9[9:0] = {4'b0000, src9[3:0], 2'b00};
spackets9 = spackets9 + 1;

end
else
Info9[9:0] = 10'd0;
end
//10
always@(Destr10)
  begin
    if((Destr10 > 0)&&(Destr10 !== 1100))
```

```
 begin
   Dest10 = Destr10/100;
   case(Dest10)
   5'd1: begin D10 = 4'b0000; ps0 = ps0 +1; end
   5'd2: begin D10 = 4'b0001; ps1 = ps1 +1; end
   5'd3: begin D10 = 4'b0010; ps2 = ps2 +1; end
   5'd4: begin D10 = 4'b0011; ps3 = ps3 +1; end
   5'd5: begin D10 = 4'b0100; ps4 = ps4 +1; end
   5'd6: begin D10 = 4'b0101; ps5 = ps5 +1; end
   5'd7: begin D10 = 4'b0110; ps6 = ps6 +1; end
   5'd8: begin D10 = 4'b0111; ps7 = ps7 +1; end
   5'd9: begin D10 = 4'b1000; ps8 = ps8 +1; end
   5'd10: begin D10 = 4'b1001; ps9 = ps9 +1; end
   5'd11: begin D10 = 4'b1010; ps10 = ps10; end
   5'd12: begin D10 = 4'b1011; ps11 = ps11 +1; end
   5'd13: begin D10 = 4'b1100; ps12 = ps12 +1; end
   5'd14: begin D10 = 4'b1101; ps13 = ps13 +1; end
   5'd15: begin D10 = 4'b1110; ps14 = ps14 +1; end
   5'd16: begin D10 = 4'b1111; ps15 = ps15 +1; end
   default: D10 = 4'b0000;
   endcase
end
else
```

```
      begin
        Dest10 = 5'd0;
        D10 = 4'b0000;
      end
  end
always@(Dest10)
begin
#5;
if((D10 !== 4'b1010)&&(Dest10 !== 5'd0))
begin
@(negedge CLK)
Info10[9:0] = {D10[3:0], src10[3:0], 2'b01};
@(posedge CLK)
Info10[9:0] = {D10[3:0], src10[3:0], 2'b11};
@(posedge CLK)
Info10[9:0] = {D10[3:0], src10[3:0], 2'b11};
@(posedge CLK)
Info10[9:0] = {D10[3:0], src10[3:0], 2'b10};
@(posedge CLK)
Info10[9:0] = {4'b0000, src10[3:0], 2'b00};
spackets10 = spackets10 + 1;

end
```

```
else
Info10[9:0] = 10'd0;
end
//11
always@(Destr11)
  begin
    if((Destr11 > 0)&&(Destr11 !== 1200))
      begin
        Dest11 = Destr11/100;
        case(Dest11)
        5'd1: begin D11 = 4'b0000; ps0 = ps0 +1; end
        5'd2: begin D11 = 4'b0001; ps1 = ps1 +1; end
        5'd3: begin D11 = 4'b0010; ps2 = ps2 +1; end
        5'd4: begin D11 = 4'b0011; ps3 = ps3 +1; end
        5'd5: begin D11 = 4'b0100; ps4 = ps4 +1; end
        5'd6: begin D11 = 4'b0101; ps5 = ps5 +1; end
        5'd7: begin D11 = 4'b0110; ps6 = ps6 +1; end
        5'd8: begin D11 = 4'b0111; ps7 = ps7 +1; end
        5'd9: begin D11 = 4'b1000; ps8 = ps8 +1; end
        5'd10: begin D11 = 4'b1001; ps9 = ps9 +1; end
        5'd11: begin D11 = 4'b1010; ps10 = ps10 +1; end
        5'd12: begin D11 = 4'b1011; ps11 = ps11; end
        5'd13: begin D11 = 4'b1100; ps12 = ps12 +1; end
```

```
        5'd14: begin D11 = 4'b1101; ps13 = ps13 +1; end
        5'd15: begin D11 = 4'b1110; ps14 = ps14 +1; end
        5'd16: begin D11 = 4'b1111; ps15 = ps15 +1; end
        default: D11 = 4'b0000;
        endcase
    end
    else
      begin
        Dest11 = 5'd0;
        D11 = 4'b0000;
      end
  end
always@(Dest11)
begin
#5;
if((D11 !== 4'b1011)&&(Dest11 !== 5'd0))
begin
@(negedge CLK)
Info11[9:0] = {D11[3:0], src11[3:0], 2'b01};
@(posedge CLK)
Info11[9:0] = {D11[3:0], src11[3:0], 2'b11};
@(posedge CLK)
Info11[9:0] = {D11[3:0], src11[3:0], 2'b11};
```

```
@(posedge CLK)
Info11[9:0] = {D11[3:0], src11[3:0], 2'b10};
@(posedge CLK)
Info11[9:0] = {4'b0000, src11[3:0], 2'b00};
spackets11 = spackets11 + 1;


end
else
Info11[9:0] = 10'd0;
end
//12
always@(Destr12)
  begin
    if((Destr12 > 0)&&(Destr12 !== 1300))
      begin
        Dest12 = Destr12/100;
        case(Dest12)
        5'd1: begin D12 = 4'b0000; ps0 = ps0 +1; end
        5'd2: begin D12 = 4'b0001; ps1 = ps1 +1; end
        5'd3: begin D12 = 4'b0010; ps2 = ps2 +1; end
        5'd4: begin D12 = 4'b0011; ps3 = ps3 +1; end
        5'd5: begin D12 = 4'b0100; ps4 = ps4 +1; end
        5'd6: begin D12 = 4'b0101; ps5 = ps5 +1; end
```

```
        5'd7: begin D12 = 4'b0110; ps6 = ps6 +1; end
        5'd8: begin D12 = 4'b0111; ps7 = ps7 +1; end
        5'd9: begin D12 = 4'b1000; ps8 = ps8 +1; end
        5'd10: begin D12 = 4'b1001; ps9 = ps9 +1; end
        5'd11: begin D12 = 4'b1010; ps10 = ps10 +1; end
        5'd12: begin D12 = 4'b1011; ps11 = ps11 +1; end
        5'd13: begin D12 = 4'b1100; ps12 = ps12; end
        5'd14: begin D12 = 4'b1101; ps13 = ps13 +1; end
        5'd15: begin D12 = 4'b1110; ps14 = ps14 +1; end
        5'd16: begin D12 = 4'b1111; ps15 = ps15 +1; end
        default: D12 = 4'b0000;
        endcase
    end
    else
      begin
        Dest12 = 5'd0;
        D12 = 4'b0000;
      end
  end
always@(Dest12)
begin
#5;
if((D12 !== 4'b1100)&&(Dest12 !== 5'd0))
```

```
begin
@(negedge CLK)
Info12[9:0] = {D12[3:0], src12[3:0], 2'b01};
@(posedge CLK)
Info12[9:0] = {D12[3:0], src12[3:0], 2'b11};
@(posedge CLK)
Info12[9:0] = {D12[3:0], src12[3:0], 2'b11};
@(posedge CLK)
Info12[9:0] = {D12[3:0], src12[3:0], 2'b10};
@(posedge CLK)
Info12[9:0] = {4'b0000, src12[3:0], 2'b00};
spackets12 = spackets12 + 1;

end
else
Info12[9:0] = 10'd0;
end
//13
always@(Destr13)
  begin
    if((Destr13 > 0)&&(Destr13 !== 1400))
      begin
        Dest13 = Destr13/100;
```

```
    case(Dest13)
    5'd1: begin D13 = 4'b0000; ps0 = ps0 +1; end
    5'd2: begin D13 = 4'b0001; ps1 = ps1 +1; end
    5'd3: begin D13 = 4'b0010; ps2 = ps2 +1; end
    5'd4: begin D13 = 4'b0011; ps3 = ps3 +1; end
    5'd5: begin D13 = 4'b0100; ps4 = ps4 +1; end
    5'd6: begin D13 = 4'b0101; ps5 = ps5 +1; end
    5'd7: begin D13 = 4'b0110; ps6 = ps6 +1; end
    5'd8: begin D13 = 4'b0111; ps7 = ps7 +1; end
    5'd9: begin D13 = 4'b1000; ps8 = ps8 +1; end
    5'd10: begin D13 = 4'b1001; ps9 = ps9 +1; end
    5'd11: begin D13 = 4'b1010; ps10 = ps10 +1; end
    5'd12: begin D13 = 4'b1011; ps11 = ps11 +1; end
    5'd13: begin D13 = 4'b1100; ps12 = ps12 +1; end
    5'd14: begin D13 = 4'b1101; ps13 = ps13; end
    5'd15: begin D13 = 4'b1110; ps14 = ps14 +1; end
    5'd16: begin D13 = 4'b1111; ps15 = ps15 +1; end
    default: D13 = 4'b0000;
    endcase
end
else
  begin
    Dest13 = 5'd0;
```

```
        D13 = 4'b0000;
      end
  end
always@(Dest13)
begin
#5;
if((D13 !== 4'b1101)&&(Dest13 !== 5'd0))
begin
@(negedge CLK)
Info13[9:0] = {D13[3:0], src13[3:0], 2'b01};
@(posedge CLK)
Info13[9:0] = {D13[3:0], src13[3:0], 2'b11};
@(posedge CLK)
Info13[9:0] = {D13[3:0], src13[3:0], 2'b11};
@(posedge CLK)
Info13[9:0] = {D13[3:0], src13[3:0], 2'b10};
@(posedge CLK)
Info13[9:0] = {4'b0000, src13[3:0], 2'b00};
spackets13 = spackets13 + 1;

end
else
Info13[9:0] = 10'd0;
```

```
end
//14
always@(Destr14)
  begin
    if((Destr14 > 0)&&(Destr14 !== 1500))
      begin
        Dest14 = Destr14/100;
        case(Dest14)
        5'd1: begin D14 = 4'b0000; ps0 = ps0 +1; end
        5'd2: begin D14 = 4'b0001; ps1 = ps1 +1; end
        5'd3: begin D14 = 4'b0010; ps2 = ps2 +1; end
        5'd4: begin D14 = 4'b0011; ps3 = ps3 +1; end
        5'd5: begin D14 = 4'b0100; ps4 = ps4 +1; end
        5'd6: begin D14 = 4'b0101; ps5 = ps5 +1; end
        5'd7: begin D14 = 4'b0110; ps6 = ps6 +1; end
        5'd8: begin D14 = 4'b0111; ps7 = ps7 +1; end
        5'd9: begin D14 = 4'b1000; ps8 = ps8 +1; end
        5'd10: begin D14 = 4'b1001; ps9 = ps9 +1; end
        5'd11: begin D14 = 4'b1010; ps10 = ps10 +1; end
        5'd12: begin D14 = 4'b1011; ps11 = ps11 +1; end
        5'd13: begin D14 = 4'b1100; ps12 = ps12 +1; end
        5'd14: begin D14 = 4'b1101; ps13 = ps13 +1; end
        5'd15: begin D14 = 4'b1110; ps14 = ps14; end
```

```
        5'd16: begin D14 = 4'b1111; ps15 = ps15 +1; end
        default: D14 = 4'b0000;
        endcase
    end
    else
      begin
        Dest14 = 5'd0;
        D14 = 4'b0000;
      end
  end
always@(Dest14)
begin
#5;
if((D14 !== 4'b1110)&&(Dest14 !== 5'd0))
begin
@(negedge CLK)
Info14[9:0] = {D14[3:0], src14[3:0], 2'b01};
@(posedge CLK)
Info14[9:0] = {D14[3:0], src14[3:0], 2'b11};
@(posedge CLK)
Info14[9:0] = {D14[3:0], src14[3:0], 2'b11};
@(posedge CLK)
Info14[9:0] = {D14[3:0], src14[3:0], 2'b10};
```

```
@(posedge CLK)
Info14[9:0] = {4'b0000, src14[3:0], 2'b00};
spackets14 = spackets14 + 1;


end
else
Info14[9:0] = 10'd0;
end
//15
always@(Destr15)
  begin
    if((Destr15 > 0)&&(Destr15 !== 1600))
      begin
        Dest15 = Destr15/100;
        case(Dest15)
        5'd1: begin D15 = 4'b0000; ps0 = ps0 +1; end
        5'd2: begin D15 = 4'b0001; ps1 = ps1 +1; end
        5'd3: begin D15 = 4'b0010; ps2 = ps2 +1; end
        5'd4: begin D15 = 4'b0011; ps3 = ps3 +1; end
        5'd5: begin D15 = 4'b0100; ps4 = ps4 +1; end
        5'd6: begin D15 = 4'b0101; ps5 = ps5 +1; end
        5'd7: begin D15 = 4'b0110; ps6 = ps6 +1; end
        5'd8: begin D15 = 4'b0111; ps7 = ps7 +1; end
```

```
        5'd9: begin D15 = 4'b1000; ps8 = ps8 +1; end
        5'd10: begin D15 = 4'b1001; ps9 = ps9 +1; end
        5'd11: begin D15 = 4'b1010; ps10 = ps10 +1; end
        5'd12: begin D15 = 4'b1011; ps11 = ps11 +1; end
        5'd13: begin D15 = 4'b1100; ps12 = ps12 +1; end
        5'd14: begin D15 = 4'b1101; ps13 = ps13 +1; end
        5'd15: begin D15 = 4'b1110; ps14 = ps14 +1; end
        5'd16: begin D15 = 4'b1111; ps15 = ps15; end
        default: D15 = 4'b0000;
        endcase
    end
    else
      begin
        Dest15 = 5'd0;
        D15 = 4'b0000;
      end
  end
always@(Dest15)
begin
#5;
if((D15 !== 4'b1111)&&(Dest15 !== 5'd0))
begin
@(negedge CLK)
```

```
Info15[9:0] = {D15[3:0], src15[3:0], 2'b01};
@(posedge CLK)
Info15[9:0] = {D15[3:0], src15[3:0], 2'b11};
@(posedge CLK)
Info15[9:0] = {D15[3:0], src15[3:0], 2'b11};
@(posedge CLK)
Info15[9:0] = {D15[3:0], src15[3:0], 2'b10};
@(posedge CLK)
Info15[9:0] = {4'b0000, src15[3:0], 2'b00};
spackets15 = spackets15 + 1;

end
else
Info15[9:0] = 10'd0;
end



always
begin
#200 CLK <= ~CLK;
end

//Measure total packets sent
```

```
always@(spackets0 or spackets1 or spackets2 or spackets3 or spackets4 or spackets5 or spackets6
or spackets7 or spackets8 or spackets9 or spackets10 or spackets11 or spackets12 or spackets13 or
spackets14 or spackets15)
begin
packetsSent = spackets0 + spackets1 + spackets2 + spackets3 + spackets4 + spackets5 + spackets6
+ spackets7 + spackets8 + spackets9 + spackets10 + spackets11 + spackets12 + spackets13 +
spackets14 + spackets15;
end



always@(posedge CLK)
begin
if(reset)
begin
Count = 0;
end
else
begin
Count = Count+ 14'b00000000000001;
end
end
```

```
always@(negedge CLK) //capture Bandwidth at Negative edge
begin
if(reset)
BW = 0;
else
begin
BW = ((|data_00) | almost_full_00) + ((|data_01) | almost_full_01) + ((|data_10) | almost_full_10) +
((|data_11) | almost_full_11) + ((|data_12) | almost_full_12) + ((|data_20) | almost_full_20) +
((|data_21) | almost_full_21) + ((|data_22) | almost_full_22) + ((|data_30) | almost_full_30) +
((|data_31) | almost_full_31) + ((|data_33) | almost_full_33) + ((|data_40) | almost_full_40) +
((|data_41) | almost_full_41) + ((|data_42) | almost_full_42) + ((|data_43) | almost_full_43) +
((|data_50) | almost_full_50) + ((|data_51) | almost_full_51) + ((|data_52) | almost_full_52) +
((|data_53) | almost_full_53) + ((|data_60) | almost_full_60) + ((|data_61) | almost_full_61) +
((|data_63) | almost_full_63) + ((|data_70) | almost_full_70) + ((|data_71) | almost_full_71) +
((|data_72) | almost_full_72) + ((|data_73) | almost_full_73) + ((|data_80) | almost_full_80) +
((|data_81) | almost_full_81) + ((|data_82) | almost_full_82) + ((|data_83) | almost_full_83) +
((|data_91) | almost_full_91) + ((|data_92) | almost_full_92) + ((|data_101) | almost_full_101) +
((|data_102) | almost_full_102) + ((|data_103) | almost_full_103) + ((|data_111) | almost_full_111)
+ ((|data_112) | almost_full_112) + ((|data_113) | almost_full_113) + ((|data_120) | almost_full_120)
+ ((|data_123) | almost_full_123) + ((|data_130) | almost_full_130) + ((|data_132) | almost_full_132)
+ ((|data_133) | almost_full_133) + ((|data_140) | almost_full_140) + ((|data_142) | almost_full_142)
+ ((|data_143) | almost_full_143) + ((|data_152) | almost_full_152) + ((|data_152) | almost_full_152
);
```

```
$fwrite(op_file17,"%d \n",Count);
$fwrite(op_file18,"%d \n",(48-BW));
end
end

always@(ps0 or ps1 or ps2 or ps3 or ps4 or ps5 or ps6 or ps7 or ps8 or ps9 or ps10 or ps11 or ps12
or ps13 or ps14 or ps15)
begin
pstotal = ps0 + ps1 + ps2 + ps3 + ps4 + ps5 + ps6 + ps7 + ps8 + ps9 + ps10 + ps11 + ps12 + ps13 +
ps14 + ps15;
end

always@(data_04)
begin
if(data_04[1:0] > 2'b00)
begin

        if(data_04[1:0] == 2'b10)
        begin
        rpackets0 = rpackets0 + 1;
          if(data_04[17:14] == 4'b0000) begin
          $fwrite(op_file0,"%d\n", (Count[13:0] - data_04[31:18]));
            corr0 = corr0 + 1; end
```

```
            else
            corr0 = corr0;
        end
        else
        rpackets0 = rpackets0;
end
else
packetsreceived = packetsreceived + 1;
end

always@(data_14)
begin
if(data_14[1:0] > 2'b00)
begin

        if(data_14[1:0] == 2'b10)
        begin
        rpackets1 = rpackets1 + 1;
        if(data_14[17:14] == 4'b0001)
        begin
        $fwrite(op_file1, "%d\n", (Count[13:0] - data_14[31:18]));
            corr1 = corr1 + 1; end
            else
```

```
            corr1 = corr1;
            end
        else
        rpackets1 = rpackets1;
end
else
packetsreceived = packetsreceived + 1;
end

always@(data_24)
begin
if(data_24[1:0] > 2'b00)
begin

        if(data_24[1:0] == 2'b10)
        begin
        rpackets2 = rpackets2 + 1;
        if(data_24[17:14] == 4'b0010)begin
        $fwrite(op_file2, "%d\n",(Count[13:0] - data_24[31:18]));
            corr2 = corr2 + 1;end
            else
            corr2 = corr2;
            end
```

```
        else
        rpackets2 = rpackets2;
end
else
packetsreceived = packetsreceived + 1;
end

always@(data_34)
begin
if(data_34[1:0] > 2'b00)
begin

        if(data_34[1:0] == 2'b10)
        begin
        rpackets4 = rpackets4 + 1;
        if(data_34[17:14] == 4'b0100)begin
        $fwrite(op_file4,"%d\n",(Count[13:0] - data_34[31:18]));
           corr4 = corr4 + 1;end
           else
           corr4 = corr4;
        end
        else
        rpackets4 = rpackets4;
```

```
end
else
packetsreceived = packetsreceived + 1;
end

always@(data_44)
begin
if(data_44[1:0] > 2'b00)
begin

        if(data_44[1:0] == 2'b10)
        begin rpackets5 = rpackets5 + 1;
        if(data_44[17:14] == 4'b0101) begin
        $fwrite(op_file5,"%d\n", (Count[13:0] - data_44[31:18]));
           corr5 = corr5 + 1; end
           else
           corr5 = corr5;
        end
        else
        rpackets5 = rpackets5;
end
else
packetsreceived = packetsreceived + 1;
```

```
end


always@(data_54)
begin
if(data_54[1:0] > 2'b00)
begin


        if(data_54[1:0] == 2'b10) begin
        rpackets6 = rpackets6 + 1;
        if(data_54[17:14] == 4'b0110)
        begin
        $fwrite(op_file6,"%d\n", (Count[13:0] - data_54[31:18]));
           corr6 = corr6 + 1;
           end
           else
           corr6 = corr6;
        end
        else
        rpackets6 = rpackets6;
end
else
packetsreceived = packetsreceived + 1;
end
```

```
always@(data_64)
begin
if(data_64[1:0] > 2'b00)
begin
        if(data_64[1:0] == 2'b10) begin
        rpackets8 = rpackets8 + 1;
        if(data_64[17:14] == 4'b1000)
        begin
        $fwrite(op_file8, "%d\n",(Count[13:0] - data_64[31:18]));
           corr8 = corr8 + 1;
           end
           else
           corr8 = corr8;
        end
        else
        rpackets8 = rpackets8;
end
else
packetsreceived = packetsreceived + 1;
end
always@(data_74)
begin
if(data_74[1:0] > 2'b00)
```

```
begin

        if(data_74[1:0] == 2'b10) begin
        rpackets9 = rpackets9 + 1;
        if(data_74[17:14] == 4'b1001)
        begin
        $fwrite(op_file9,"%d\n",(Count[13:0] - data_74[31:18]));
           corr9 = corr9 + 1;
           end
           else
           corr9 = corr9;
        end
        else
        rpackets9 = rpackets9;
end
else
packetsreceived = packetsreceived + 1;
end

always@(data_84)
begin
if(data_84[1:0] > 2'b00)
begin
```

```
        if(data_84[1:0] == 2'b10) begin
        rpackets10 = rpackets10 + 1;
        if(data_84[17:14] == 4'b1010)
        begin
        $fwrite(op_file10, "%d\n",(Count[13:0] - data_84[31:18]));
           corr10 = corr10 + 1;
           end
           else
           corr10 = corr10;
        end
        else
        rpackets10 = rpackets10;
end
else
packetsreceived = packetsreceived + 1;
end

always@(data_94)
begin
if(data_94[1:0] > 2'b00)
begin
```

```
        if(data_94[1:0] == 2'b10) begin
        rpackets3 = rpackets3 + 1;
        if(data_94[17:14] == 4'b0011)
        begin
        $fwrite(op_file3,"%d\n", (Count[13:0] - data_94[31:18]));

           corr3 = corr3 + 1;end
           else
           corr3 = corr3;
        end
        else
        rpackets3 = rpackets3;
end
else
packetsreceived = packetsreceived + 1;
end

always@(data_104)
begin if(data_104[1:0] > 2'b00)
begin

        if(data_104[1:0] == 2'b10) begin
        rpackets7 = rpackets7 + 1;
```

```
        if(data_104[17:14] == 4'b0111)
        begin
           corr7 = corr7 + 1;
           $fwrite(op_file7,"%d\n", (Count[13:0] - data_104[31:18]));
           end
           else
           corr7 = corr7;
        end
        else
        rpackets7 = rpackets7;
end
else
packetsreceived = packetsreceived + 1;
end

always@(data_114)
begin if(data_114[1:0] > 2'b00)
begin

        if(data_114[1:0] == 2'b10) begin
        rpackets11 = rpackets11 + 1;
        if(data_114[17:14] == 4'b1011)
        begin
```

```
            corr11 = corr11 + 1;
            $fwrite(op_file11, "%d\n",(Count[13:0] - data_114[31:18]));
            end
            else
            corr11 = corr11;
        end
        else
        rpackets11 = rpackets11;
end
else
packetsreceived = packetsreceived + 1;
end

always@(data_124)
begin if(data_124[1:0] > 2'b00)
begin

        if(data_124[1:0] == 2'b10) begin
        rpackets12 = rpackets12 + 1;
        if(data_124[17:14] == 4'b1100)
        begin
        $fwrite(op_file12, "%d\n", (Count[13:0] - data_124[31:18]));
            corr12 = corr12 + 1;
```

```
            end
            else
            corr12 = corr12;
        end
        else
        rpackets12 = rpackets12;
end
else
packetsreceived = packetsreceived + 1;
end

always@(data_134)
begin if(data_134[1:0] > 2'b00)
begin

        if(data_134[1:0] == 2'b10) begin
        rpackets13 = rpackets13 + 1;
        if(data_134[17:14] == 4'b1101)
        begin
        $fwrite(op_file13,"%d\n", (Count[13:0] - data_134[31:18]));
            corr13 = corr13 + 1;
            end
            else
```

```
            corr13 = corr13;
        end
        else
        rpackets13 = rpackets13;
end
else
packetsreceived = packetsreceived + 1;
end

always@(data_144)
begin if(data_144[1:0] > 2'b00)
begin

        if(data_144[1:0] == 2'b10) begin
        rpackets14 = rpackets14 + 1;
        if(data_144[17:14] == 4'b1110)
        begin
        $fwrite(op_file14, "%d\n", (Count[13:0] - data_144[31:18]));
            corr14 = corr14 + 1;
            end
            else
            corr14 = corr14;
        end
```

```
        else
        rpackets14 = rpackets14;
end
else
packetsreceived = packetsreceived + 1;
end

always@(data_154)
begin
if(data_154[1:0] > 2'b00)
begin

        if(data_154[1:0] == 2'b10) begin
        rpackets15 = rpackets15 + 1;
        if(data_154[17:14] == 4'b1111)begin
        $fwrite(op_file15,"%d\n", (Count[13:0] - data_154[31:18]));
           corr15 = corr15 + 1;
           end
           else
           corr15 = corr15;
        end
        else
        rpackets15 = rpackets15;
```

```
end
else
packetsreceived = packetsreceived + 1;
end


always@(data_in0)
begin
if(data_in0[1:0] > 2'b00)
inreq0 = 1'b1;
else
inreq0 = 1'b0;
end

always@(data_in1)
begin
if(data_in1[1:0] > 2'b00)
inreq1 = 1'b1;
else
inreq1 = 1'b0;
end

always@(data_in2)
```

```
begin
if(data_in2[1:0] > 2'b00)
inreq2 = 1'b1;
else
inreq2 = 1'b0;
end

always@(data_in3)
begin
if(data_in3[1:0] > 2'b00)
inreq3 = 1'b1;
else
inreq3 = 1'b0;
end

always@(data_in4)
begin
if(data_in4[1:0] > 2'b00)
inreq4 = 1'b1;
else
inreq4 = 1'b0;
end
```

```
always@(data_in5)
begin
if(data_in5[1:0] > 2'b00)
inreq5 = 1'b1;
else
inreq5 = 1'b0;
end

always@(data_in6)
begin
if(data_in6[1:0] > 2'b00)
inreq6 = 1'b1;
else
inreq6 = 1'b0;
end

always@(data_in7)
begin
if(data_in7[1:0] > 2'b00)
inreq7 = 1'b1;
else
inreq7 = 1'b0;
end
```

```
always@(data_in8)
begin
if(data_in8[1:0] > 2'b00)
inreq8 = 1'b1;
else
inreq8 = 1'b0;
end

always@(data_in9)
begin
if(data_in9[1:0] > 2'b00)
inreq9 = 1'b1;
else
inreq9 = 1'b0;
end

always@(data_in10)
begin
if(data_in10[1:0] > 2'b00)
inreq10 = 1'b1;
else
inreq10 = 1'b0;
```

```
end

always@(data_in11)
begin
if(data_in11[1:0] > 2'b00)
inreq11 = 1'b1;
else
inreq11 = 1'b0;
end

always@(data_in12)
begin
if(data_in12[1:0] > 2'b00)
inreq12 = 1'b1;
else
inreq12 = 1'b0;
end

always@(data_in13)
begin
if(data_in13[1:0] > 2'b00)
inreq13 = 1'b1;
else
```

```
inreq13 = 1'b0;
end

always@(data_in14)
begin
if(data_in14[1:0] > 2'b00)
inreq14 = 1'b1;
else
inreq14 = 1'b0;
end

always@(data_in15)
begin
if(data_in15[1:0] > 2'b00)
inreq15 = 1'b1;
else
inreq15 = 1'b0;
end

endmodule
```

b. Five-port router

```
`timescale 1ns/1ns
// Copyright (C) 1991-2009 Altera Corporation
// Your use of Altera Corporation's design tools, logic functions
// and other software and tools, and its AMPP partner logic
// functions, and any output files from any of the foregoing
// (including device programming or simulation files), and any
// associated documentation or information are expressly subject
// to the terms and conditions of the Altera Program License
// Subscription Agreement, Altera MegaCore Function License
// Agreement, or other applicable license agreement, including,
// without limitation, that your use is for the sole purpose of
// programming logic devices manufactured by Altera and sold by
// Altera or its authorized distributors.  Please refer to the
// applicable agreement for further details.
// PROGRAM		"Quartus II"
// VERSION		"Version 9.1 Build 222 10/21/2009 SJ Web Edition"
// CREATED		"Fri Jan 22 14:22:52 2010"
//Edited by Jonathan Frey 5/26/2015
module fiveportrouter4(CLK,
                       reset,
                       wrreq_E,
                       wrreq_S,
```

wrreq_W,

wrreq_N,

wrreq_L,

NACK_in_E,

NACK_in_S,

NACK_in_W,

NACK_in_N,

NACK_in_L,

AlmostFull_in_E,

AlmostFull_in_S,

AlmostFull_in_W,

AlmostFull_in_N,

AlmostFull_in_L,

CurrentID,

dataino_E,

dataino_L,

dataino_N,

dataino_S,

dataino_W,

valid_in,

wrreq_out_E,

wrreq_out_S,

wrreq_out_W,

```
                wrreq_out_N,
                wrreq_out_L,
                almost_full_E,
                almost_full_S,
                almost_full_W,
                almost_full_N,
                almost_full_L,
                dataout_E,
                dataout_L,
                dataout_N,
                dataout_S,
                dataout_W,
                empty);
parameter INPUTWIDTH = 32;
parameter FINALWIDTH = 36;
input CLK;
input reset;
input wrreq_E;
input wrreq_S;
input wrreq_W;
input wrreq_N;
input wrreq_L;
input NACK_in_E;
```

```
input NACK_in_S;
input NACK_in_W;
input NACK_in_N;
input NACK_in_L;
input AlmostFull_in_E;
input AlmostFull_in_S;
input AlmostFull_in_W;
input AlmostFull_in_N;
input AlmostFull_in_L;
input[3 : 0] CurrentID;
input[INPUTWIDTH - 1 : 0] dataino_E;
input[INPUTWIDTH - 1 : 0] dataino_L;
input[INPUTWIDTH - 1 : 0] dataino_N;
input[INPUTWIDTH - 1 : 0] dataino_S;
input[INPUTWIDTH - 1 : 0] dataino_W;
input[4 : 0] valid_in;
output wrreq_out_E;
output wrreq_out_S;
output wrreq_out_W;
output wrreq_out_N;
output wrreq_out_L;
output almost_full_E;
output almost_full_S;
```

```
output almost_full_W;
output almost_full_N;
output almost_full_L;
output[INPUTWIDTH - 1 : 0] dataout_E;
output[INPUTWIDTH - 1 : 0] dataout_L;
output[INPUTWIDTH - 1 : 0] dataout_N;
output[INPUTWIDTH - 1 : 0] dataout_S;
output[INPUTWIDTH - 1 : 0] dataout_W;
output[4 : 0] empty;
wire[FINALWIDTH - 1 : 0] datain_E;
wire[FINALWIDTH - 1 : 0] datain_W;
wire[FINALWIDTH - 1 : 0] datain_N;
wire[FINALWIDTH - 1 : 0] datain_S;
wire[FINALWIDTH - 1 : 0] datain_L;
wire[FINALWIDTH - 1 : 0] datain_E_Sc_enc;
wire[FINALWIDTH - 1 : 0] datain_W_Sc_enc;
wire[FINALWIDTH - 1 : 0] datain_N_Sc_enc;
wire[FINALWIDTH - 1 : 0] datain_S_Sc_enc;
wire[FINALWIDTH - 1 : 0] datain_L_Sc_enc;
wire[4 : 0] empty_ALTERA_SYNTHESIZED;
wire[4 : 0] IntendedBufFull;
wire[4 : 0] NACK_in;
wire[24 : 0] Port_Ad;
```

```
wire SYNTHESIZED_WIRE_0;
wire SYNTHESIZED_WIRE_1;
wire[3 : 0] SYNTHESIZED_WIRE_2;
wire SYNTHESIZED_WIRE_3;
wire SYNTHESIZED_WIRE_4;
wire[3 : 0] SYNTHESIZED_WIRE_5;
wire[FINALWIDTH - 1 : 0] SYNTHESIZED_WIRE_6;
wire[FINALWIDTH - 1 : 0] SYNTHESIZED_WIRE_7;
wire[FINALWIDTH - 1 : 0] SYNTHESIZED_WIRE_8;
wire[FINALWIDTH - 1 : 0] SYNTHESIZED_WIRE_9;
wire[FINALWIDTH - 1 : 0] SYNTHESIZED_WIRE_10;
wire SYNTHESIZED_WIRE_11;
wire SYNTHESIZED_WIRE_12;
wire[3 : 0] SYNTHESIZED_WIRE_13;
wire SYNTHESIZED_WIRE_14;
wire SYNTHESIZED_WIRE_15;
wire SYNTHESIZED_WIRE_16;
wire SYNTHESIZED_WIRE_17;
wire SYNTHESIZED_WIRE_18;
wire SYNTHESIZED_WIRE_19;
wire SYNTHESIZED_WIRE_20;
wire SYNTHESIZED_WIRE_21;
wire[3 : 0] SYNTHESIZED_WIRE_22;
```

```
wire SYNTHESIZED_WIRE_23;
wire SYNTHESIZED_WIRE_24;
wire SYNTHESIZED_WIRE_25;
wire SYNTHESIZED_WIRE_26;
wire SYNTHESIZED_WIRE_27;
wire SYNTHESIZED_WIRE_28;
wire[3 : 0] SYNTHESIZED_WIRE_29;
wire[2 : 0] SYNTHESIZED_WIRE_30;
wire[2 : 0] SYNTHESIZED_WIRE_31;
wire[2 : 0] SYNTHESIZED_WIRE_32;
wire[2 : 0] SYNTHESIZED_WIRE_33;
wire[2 : 0] SYNTHESIZED_WIRE_34;
wire[FINALWIDTH - 1 : 0] SYNTHESIZED_WIRE_35;
wire[FINALWIDTH - 1 : 0] SYNTHESIZED_WIRE_36;
wire[FINALWIDTH - 1 : 0] SYNTHESIZED_WIRE_37;
wire[FINALWIDTH - 1 : 0] SYNTHESIZED_WIRE_38;
wire[FINALWIDTH - 1 : 0] SYNTHESIZED_WIRE_39;
wire[24 : 0] SYNTHESIZED_WIRE_40;
wire[24 : 0] SYNTHESIZED_WIRE_41;
wire[2 : 0] select; /////For permutation, from keygen
wire ErrEpre,
    ErrLpre,
    ErrNpre,
```

```
    ErrSpre,
    ErrWpre;
wire ErrE,
    ErrL,
    ErrN,
    ErrS,
    ErrW;

wire[FINALWIDTH - 1 : 0] dataout_Epre;
wire[FINALWIDTH - 1 : 0] dataout_Lpre;
wire[FINALWIDTH - 1 : 0] dataout_Spre;
wire[FINALWIDTH - 1 : 0] dataout_Npre;
wire[FINALWIDTH - 1 : 0] dataout_Wpre;
wire[FINALWIDTH - 1 : 0] dataout_E_enc;
wire[FINALWIDTH - 1 : 0] dataout_L_enc;
wire[FINALWIDTH - 1 : 0] dataout_S_enc;
wire[FINALWIDTH - 1 : 0] dataout_N_enc;
wire[FINALWIDTH - 1 : 0] dataout_W_enc;
assign select = 3'b111;
////////////// START ENCODING PROCESS
HammingEncoder15_11 E0(.DataIn(dataino_E[31 : 0]),
                    .DataOut(datain_E_Sc_enc[35 : 0]));
```

```
HammingEncoder15_11 E1(.DataIn(dataino_W[31 : 0]),
                .DataOut(datain_W_Sc_enc[35 : 0]));

HammingEncoder15_11 E2(.DataIn(dataino_N[31 : 0]),
                .DataOut(datain_N_Sc_enc[35 : 0]));

HammingEncoder15_11 E3(.DataIn(dataino_S[31 : 0]),
                .DataOut(datain_S_Sc_enc[35 : 0]));

HammingEncoder15_11 E4(.DataIn(dataino_L[31 : 0]),
                .DataOut(datain_L_Sc_enc[35 : 0]));
//END ENCODING PROCESS
//START FLIT SCRAMBLING PROCESS
Perm32plus4 p0(.inputa(datain_E_Sc_enc),
        .select(select),
        .outputb(datain_E));
Perm32plus4 p1(.inputa(datain_W_Sc_enc),
        .select(select),
        .outputb(datain_W));
Perm32plus4 p2(.inputa(datain_N_Sc_enc),
        .select(select),
        .outputb(datain_N));
Perm32plus4 p3(.inputa(datain_S_Sc_enc),
```

```
            .select(select),
            .outputb(datain_S));
  Perm32plus4 p4(.inputa(datain_L_Sc_enc),
            .select(select),
            .outputb(datain_L));
  //END FLIT SCRAMBLING PROCESS
  QYInputFIFO3 EastInFIFO(.clk(CLK),
                  .reset(reset),
                  .wrreq(wrreq_E),
                  .rdreq(SYNTHESIZED_WIRE_27),
                  .full_i(SYNTHESIZED_WIRE_28),
                  .datain(datain_E),
                  .IndexCounter_i(SYNTHESIZED_WIRE_29),
                  .almost_full(almost_full_E),
                  .empty(empty_ALTERA_SYNTHESIZED[0]),
                  .full_o(SYNTHESIZED_WIRE_28),
                  .dataout(SYNTHESIZED_WIRE_35),
                  .IndexCounter_o(SYNTHESIZED_WIRE_29),
                  .ErrDin(ErrEpre));
  defparam EastInFIFO.FLITWIDTH = 36;
  assign SYNTHESIZED_WIRE_27 = SYNTHESIZED_WIRE_25 |
SYNTHESIZED_WIRE_26;
  assign SYNTHESIZED_WIRE_26 = Port_Ad[4] | Port_Ad[9] | Port_Ad[14];
```

```
  assign SYNTHESIZED_WIRE_25 = Port_Ad[19] | Port_Ad[24];
  QYInputFIFO3 SouthInFIFO(.clk(CLK),
                           .reset(reset),
                           .wrreq(wrreq_S),
                           .rdreq(SYNTHESIZED_WIRE_20),
                           .full_i(SYNTHESIZED_WIRE_21),
                           .datain(datain_S),
                           .IndexCounter_i(SYNTHESIZED_WIRE_22),
                           .almost_full(almost_full_S),
                           .empty(empty_ALTERA_SYNTHESIZED[1]),
                           .full_o(SYNTHESIZED_WIRE_21),
                           .dataout(SYNTHESIZED_WIRE_38),
                           .IndexCounter_o(SYNTHESIZED_WIRE_22),
                           .ErrDin(ErrSpre));
  defparam SouthInFIFO.FLITWIDTH = 36;
  assign SYNTHESIZED_WIRE_20 = SYNTHESIZED_WIRE_14 |
SYNTHESIZED_WIRE_15;
  assign SYNTHESIZED_WIRE_15 = Port_Ad[3] | Port_Ad[8] | Port_Ad[13];
  assign SYNTHESIZED_WIRE_14 = Port_Ad[18] | Port_Ad[23];
  QYInputFIFO3 WestInFIFO(.clk(CLK),
                          .reset(reset),
                          .wrreq(wrreq_W),
                          .rdreq(SYNTHESIZED_WIRE_11),
```

```
                .full_i(SYNTHESIZED_WIRE_12),
                .datain(datain_W),
                .IndexCounter_i(SYNTHESIZED_WIRE_13),
                .almost_full(almost_full_W),
                .empty(empty_ALTERA_SYNTHESIZED[2]),
                .full_o(SYNTHESIZED_WIRE_12),
                .dataout(SYNTHESIZED_WIRE_39),
                .IndexCounter_o(SYNTHESIZED_WIRE_13),
                .ErrDin(ErrWpre));
  defparam WestInFIFO.FLITWIDTH = 36;
  assign SYNTHESIZED_WIRE_11 = SYNTHESIZED_WIRE_16 |
SYNTHESIZED_WIRE_17;
  assign SYNTHESIZED_WIRE_17 = Port_Ad[2] | Port_Ad[7] | Port_Ad[12];
  assign SYNTHESIZED_WIRE_16 = Port_Ad[17] | Port_Ad[22];
  QYInputFIFO3 NorthInFIFO(.clk(CLK),
                .reset(reset),
                .wrreq(wrreq_N),
                .rdreq(SYNTHESIZED_WIRE_0),
                .full_i(SYNTHESIZED_WIRE_1),
                .datain(datain_N),
                .IndexCounter_i(SYNTHESIZED_WIRE_2),
                .almost_full(almost_full_N),
                .empty(empty_ALTERA_SYNTHESIZED[3]),
```

```
                .full_o(SYNTHESIZED_WIRE_1),
                .dataout(SYNTHESIZED_WIRE_37),
                .IndexCounter_o(SYNTHESIZED_WIRE_2),
                .ErrDin(ErrNpre));
  defparam NorthInFIFO.FLITWIDTH = 36;
  assign SYNTHESIZED_WIRE_0 = SYNTHESIZED_WIRE_18 |
SYNTHESIZED_WIRE_19;
  assign SYNTHESIZED_WIRE_19 = Port_Ad[1] | Port_Ad[6] | Port_Ad[11];
  assign SYNTHESIZED_WIRE_18 = Port_Ad[16] | Port_Ad[21];
  QYInputFIFO3 LocalInFIFO(.clk(CLK),
                .reset(reset),
                .wrreq(wrreq_L),
                .rdreq(SYNTHESIZED_WIRE_3),
                .full_i(SYNTHESIZED_WIRE_4),
                .datain(datain_L),
                .IndexCounter_i(SYNTHESIZED_WIRE_5),
                .almost_full(almost_full_L),
                .empty(empty_ALTERA_SYNTHESIZED[4]),
                .full_o(SYNTHESIZED_WIRE_4),
                .dataout(SYNTHESIZED_WIRE_36),
                .IndexCounter_o(SYNTHESIZED_WIRE_5),
                .ErrDin(ErrLpre));
  defparam LocalInFIFO.FLITWIDTH = 36;
```

```
 assign SYNTHESIZED_WIRE_3 = SYNTHESIZED_WIRE_23 |
SYNTHESIZED_WIRE_24;
 assign SYNTHESIZED_WIRE_24 = Port_Ad[0] | Port_Ad[5] | Port_Ad[10];
 assign SYNTHESIZED_WIRE_23 = Port_Ad[15] | Port_Ad[20];
 CircularFIFO4 EastOutFIFO(.clk(CLK),
                .reset(reset),
                .valid_in(valid_in[0]),
                .NACK(NACK_in[0]),
                .almost_full(IntendedBufFull[0]),
                .datain(SYNTHESIZED_WIRE_6),
                .wrreq(wrreq_out_E),
                .select(select),
                .dataout(dataout_Epre));
 defparam EastOutFIFO.CodedFLITWIDTH = 36;
 defparam EastOutFIFO.FLITWIDTH = 32;
 //DP and DE EAST
 DePerm32plus4 DP0(.inputa(dataout_Epre),
            .select(select),
            .outputb(dataout_E_enc));
 HammingDecoder15_11 DE0(.DataIno(dataout_E_enc[35 : 0]),
                .DataOut(dataout_E[31 : 0]),
                .ErrDetection(ErrE));
 CircularFIFO4 SouthOutFIFO(.clk(CLK),
```

```
                    .reset(reset),
                    .valid_in(valid_in[1]),
                    .NACK(NACK_in[1]),
                    .almost_full(IntendedBufFull[1]),
                    .datain(SYNTHESIZED_WIRE_7),
                    .wrreq(wrreq_out_S),
                    .select(select),
                    .dataout(dataout_Spre));
defparam SouthOutFIFO.CodedFLITWIDTH = 36;
defparam SouthOutFIFO.FLITWIDTH = 32;
//DP and DE SOUTH
DePerm32plus4 DP1(.inputa(dataout_Spre),
            .select(select),
            .outputb(dataout_S_enc));
HammingDecoder15_11 DE1(.DataIno(dataout_S_enc[35 : 0]),
                .DataOut(dataout_S[31 : 0]),
                .ErrDetection(ErrS));
CircularFIFO4 WestOutFIFO(.clk(CLK),
                    .reset(reset),
                    .valid_in(valid_in[2]),
                    .NACK(NACK_in[2]),
                    .almost_full(IntendedBufFull[2]),
                    .datain(SYNTHESIZED_WIRE_8),
```

```
                .wrreq(wrreq_out_W),
                .select(select),
                .dataout(dataout_Wpre));
defparam WestOutFIFO.CodedFLITWIDTH = 36;
defparam WestOutFIFO.FLITWIDTH = 32;
//DP and DE WEST
DePerm32plus4 DP2(.inputa(dataout_Wpre),
          .select(select),
          .outputb(dataout_W_enc));
HammingDecoder15_11 DE2(.DataIno(dataout_W_enc[35 : 0]),
              .DataOut(dataout_W[31 : 0]),
              .ErrDetection(ErrW));
CircularFIFO4 NorthOutFIFO(.clk(CLK),
                .reset(reset),
                .valid_in(valid_in[3]),
                .NACK(NACK_in[3]),
                .almost_full(IntendedBufFull[3]),
                .datain(SYNTHESIZED_WIRE_9),
                .wrreq(wrreq_out_N),
                .select(select),
                .dataout(dataout_Npre));
defparam NorthOutFIFO.CodedFLITWIDTH = 36;
defparam NorthOutFIFO.FLITWIDTH = 32;
```

```
  //DP and DE NORTH
  DePerm32plus4 DP3(.inputa(dataout_Npre),
                    .select(select),
                    .outputb(dataout_N_enc));
  HammingDecoder15_11 DE3(.DataIno(dataout_N_enc[35 : 0]),
                          .DataOut(dataout_N[31 : 0]),
                          .ErrDetection(ErrN));
  CircularFIFO4 LocalOutFIFO(.clk(CLK),
                             .reset(reset),
                             .valid_in(valid_in[4]),
                             .NACK(NACK_in[4]),
                             .almost_full(IntendedBufFull[4]),
                             .datain(SYNTHESIZED_WIRE_10),
//errorinject//SYNTHESIZED_WIRE_10[35:25],25'd0}),
                             .wrreq(wrreq_out_L),
                             .select(select),
                             .dataout(dataout_Lpre));
  defparam LocalOutFIFO.CodedFLITWIDTH = 36;
  defparam LocalOutFIFO.FLITWIDTH = 32;
  //DP and DE LOCAL
  DePerm32plus4 DP4(.inputa(dataout_Lpre),
                    .select(select),
                    .outputb(dataout_L_enc));
```

```
HammingDecoder15_11 DE4(.DataIno(dataout_L_enc[35 : 0]),
                        .DataOut(dataout_L[31 : 0]),
                        .ErrDetection(ErrL));
FullCrossbar4 FullCrossbar(.CLK(CLK),
                           .Reset(reset),
                           .select(select),
                           .BufFull_in(IntendedBufFull),
                           .CurrentID(CurrentID),
                           .DestPort_pre_E(SYNTHESIZED_WIRE_30),
                           .DestPort_pre_L(SYNTHESIZED_WIRE_31),
                           .DestPort_pre_N(SYNTHESIZED_WIRE_32),
                           .DestPort_pre_S(SYNTHESIZED_WIRE_33),
                           .DestPort_pre_W(SYNTHESIZED_WIRE_34),
                           .FlitIn_E(SYNTHESIZED_WIRE_35),
                           .FlitIn_L(SYNTHESIZED_WIRE_36),
                           .FlitIn_N(SYNTHESIZED_WIRE_37),
                           .FlitIn_S(SYNTHESIZED_WIRE_38),
                           .FlitIn_W(SYNTHESIZED_WIRE_39),
                           .NACK(NACK_in),
                           .RRReg_pre(SYNTHESIZED_WIRE_40),
                           .TableTag_pre(SYNTHESIZED_WIRE_41),
                           .DestPort_E(SYNTHESIZED_WIRE_30),
                           .DestPort_L(SYNTHESIZED_WIRE_31),
```

```
                .DestPort_N(SYNTHESIZED_WIRE_32),
                .DestPort_S(SYNTHESIZED_WIRE_33),
                .DestPort_W(SYNTHESIZED_WIRE_34),
                .FlitOut_E(SYNTHESIZED_WIRE_6),
                .FlitOut_L(SYNTHESIZED_WIRE_10),
                .FlitOut_N(SYNTHESIZED_WIRE_9),
                .FlitOut_S(SYNTHESIZED_WIRE_7),
                .FlitOut_W(SYNTHESIZED_WIRE_8),
                .Port_Ad(Port_Ad),
                .RRReg_cur(SYNTHESIZED_WIRE_40),
                .TableTag_cur(SYNTHESIZED_WIRE_41),
                .ErrEpre(ErrEpre),
                .ErrLpre(ErrLpre),
                .ErrNpre(ErrNpre),
                .ErrSpre(ErrSpre),
                .ErrWpre(ErrWpre));
defparam FullCrossbar.FLITWIDTH = 36;
defparam FullCrossbar.NOCSIZE = 4;
assign empty = empty_ALTERA_SYNTHESIZED;
assign IntendedBufFull[0] = AlmostFull_in_E;
assign IntendedBufFull[1] = AlmostFull_in_S;
assign IntendedBufFull[2] = AlmostFull_in_W;
assign IntendedBufFull[3] = AlmostFull_in_N;
```

```
        assign IntendedBufFull[4] = AlmostFull_in_L;

        assign NACK_in[0] = NACK_in_E;

        assign NACK_in[1] = NACK_in_S;

        assign NACK_in[2] = NACK_in_W;

        assign NACK_in[3] = NACK_in_N;

        assign NACK_in[4] = NACK_in_L;

endmodule
```

c. Input FIFO

```
//Use FPGA InputChannel to create a circular FIFO
`timescale 1ns/1ns
module QYInputFIFO3(clk,
                    reset,
                    wrreq,
                    rdreq, //connect to Port_Ad
                    datain,
                    IndexCounter_i,
                    full_i,
                    almost_full,
                    empty,
                    dataout,
                    IndexCounter_o,
                    full_o,
                    ErrDin);
  parameter FLITWIDTH = 36;
  input clk;
  input reset;
  input wrreq;
  input rdreq;
  input[FLITWIDTH - 1 : 0] datain;
  input[3 : 0] IndexCounter_i;
```

```
input full_i;
input ErrDin;
output[FLITWIDTH - 1 : 0] dataout;
output almost_full;
output empty;
output[3 : 0] IndexCounter_o;
output full_o;
wire clk;
wire reset;
wire wrreq;
wire rdreq;
wire[FLITWIDTH - 1 : 0] datain;
wire[3 : 0] IndexCounter_i;
wire full_i;
reg[FLITWIDTH - 1 : 0] dataout;
reg almost_full;
reg empty;
reg[3 : 0] IndexCounter_o;
reg full_o;
//intermediate signals
reg[FLITWIDTH - 1 : 0] fifo_input_s1;
reg[FLITWIDTH - 1 : 0] fifo_input_s2;
reg[FLITWIDTH - 1 : 0] fifo_input_s3;
```

```
reg[FLITWIDTH - 1 : 0] fifo_input_s4;
reg[FLITWIDTH - 1 : 0] fifo_input_s5;
reg[FLITWIDTH - 1 : 0] fifo_input_s6;
reg[FLITWIDTH - 1 : 0] fifo_input_s7;
reg[FLITWIDTH - 1 : 0] fifo_input_s8;
wire[FLITWIDTH - 1 : 0] fifo_output_s1;
wire[FLITWIDTH - 1 : 0] fifo_output_s2;
wire[FLITWIDTH - 1 : 0] fifo_output_s3;
wire[FLITWIDTH - 1 : 0] fifo_output_s4;
wire[FLITWIDTH - 1 : 0] fifo_output_s5;
wire[FLITWIDTH - 1 : 0] fifo_output_s6;
wire[FLITWIDTH - 1 : 0] fifo_output_s7;
wire[FLITWIDTH - 1 : 0] fifo_output_s8;
reg[1 : 0] Ctrl_s1;
reg[1 : 0] Ctrl_s2;
reg[1 : 0] Ctrl_s3;
reg[1 : 0] Ctrl_s4;
reg[1 : 0] Ctrl_s5;
reg[1 : 0] Ctrl_s6;
reg[1 : 0] Ctrl_s7;
reg[1 : 0] Ctrl_s8;
reg[3 : 0] IndexCounter_o_tmp;
```

```
  //Ctrl =00: write back Q, 01: write zero vector, 10: propagate previous DFF, 11: write new
data
  always
    @(rdreq or
      wrreq or
      full_i or
      datain or
      IndexCounter_i)
    begin
      if (rdreq)
        if (wrreq) //read and write
          begin
            case (IndexCounter_i)
              4'b0000:
                begin //the data is read out, but the value should stay in the DFF
                  Ctrl_s1[1 : 0] <= 2'b11;
                  Ctrl_s2[1 : 0] <= 2'b01;
                  Ctrl_s3[1 : 0] <= 2'b01;
                  Ctrl_s4[1 : 0] <= 2'b01;
                  Ctrl_s5[1 : 0] <= 2'b01;
                  Ctrl_s6[1 : 0] <= 2'b01;
                  Ctrl_s7[1 : 0] <= 2'b01;
                  Ctrl_s8[1 : 0] <= 2'b01;
```

```
  end
4'b0001:
  begin
    Ctrl_s1[1 : 0] <= 2'b11;
    Ctrl_s2[1 : 0] <= 2'b01;
    Ctrl_s3[1 : 0] <= 2'b01;
    Ctrl_s4[1 : 0] <= 2'b01;
    Ctrl_s5[1 : 0] <= 2'b01;
    Ctrl_s6[1 : 0] <= 2'b01;
    Ctrl_s7[1 : 0] <= 2'b01;
    Ctrl_s8[1 : 0] <= 2'b01;
  end
4'b0010:
  begin
    Ctrl_s1[1 : 0] <= 2'b10;
    Ctrl_s2[1 : 0] <= 2'b11;
    Ctrl_s3[1 : 0] <= 2'b01;
    Ctrl_s4[1 : 0] <= 2'b01;
    Ctrl_s5[1 : 0] <= 2'b01;
    Ctrl_s6[1 : 0] <= 2'b01;
    Ctrl_s7[1 : 0] <= 2'b01;
    Ctrl_s8[1 : 0] <= 2'b01;
  end
```

```
4'b0011:
  begin
    Ctrl_s1[1 : 0] <= 2'b10;
    Ctrl_s2[1 : 0] <= 2'b10;
    Ctrl_s3[1 : 0] <= 2'b11;
    Ctrl_s4[1 : 0] <= 2'b01;
    Ctrl_s5[1 : 0] <= 2'b01;
    Ctrl_s6[1 : 0] <= 2'b01;
    Ctrl_s7[1 : 0] <= 2'b01;
    Ctrl_s8[1 : 0] <= 2'b01;
  end
4'b0100:
  begin
    Ctrl_s1[1 : 0] <= 2'b10;
    Ctrl_s2[1 : 0] <= 2'b10;
    Ctrl_s3[1 : 0] <= 2'b10;
    Ctrl_s4[1 : 0] <= 2'b11;
    Ctrl_s5[1 : 0] <= 2'b01;
    Ctrl_s6[1 : 0] <= 2'b01;
    Ctrl_s7[1 : 0] <= 2'b01;
    Ctrl_s8[1 : 0] <= 2'b01;
  end
4'b0101:
```

```
  begin
    Ctrl_s1[1 : 0] <= 2'b10;
    Ctrl_s2[1 : 0] <= 2'b10;
    Ctrl_s3[1 : 0] <= 2'b10;
    Ctrl_s4[1 : 0] <= 2'b10;
    Ctrl_s5[1 : 0] <= 2'b11;
    Ctrl_s6[1 : 0] <= 2'b01;
    Ctrl_s7[1 : 0] <= 2'b01;
    Ctrl_s8[1 : 0] <= 2'b01;
  end
4'b0110:
  begin
    Ctrl_s1[1 : 0] <= 2'b10;
    Ctrl_s2[1 : 0] <= 2'b10;
    Ctrl_s3[1 : 0] <= 2'b10;
    Ctrl_s4[1 : 0] <= 2'b10;
    Ctrl_s5[1 : 0] <= 2'b10;
    Ctrl_s6[1 : 0] <= 2'b11;
    Ctrl_s7[1 : 0] <= 2'b01;
    Ctrl_s8[1 : 0] <= 2'b01;
  end
4'b0111:
  begin
```

```
    Ctrl_s1[1 : 0] <= 2'b10;
    Ctrl_s2[1 : 0] <= 2'b10;
    Ctrl_s3[1 : 0] <= 2'b10;
    Ctrl_s4[1 : 0] <= 2'b10;
    Ctrl_s5[1 : 0] <= 2'b10;
    Ctrl_s6[1 : 0] <= 2'b10;
    Ctrl_s7[1 : 0] <= 2'b11;
    Ctrl_s8[1 : 0] <= 2'b01;
  end
4'b1000:
  begin
    Ctrl_s1[1 : 0] <= 2'b10;
    Ctrl_s2[1 : 0] <= 2'b10;
    Ctrl_s3[1 : 0] <= 2'b10;
    Ctrl_s4[1 : 0] <= 2'b10;
    Ctrl_s5[1 : 0] <= 2'b10;
    Ctrl_s6[1 : 0] <= 2'b10;
    Ctrl_s7[1 : 0] <= 2'b10;
    Ctrl_s8[1 : 0] <= 2'b11;
  end
default:
  begin
    Ctrl_s1[1 : 0] <= 2'b00;
```

```
        Ctrl_s2[1 : 0] <= 2'b00;
        Ctrl_s3[1 : 0] <= 2'b00;
        Ctrl_s4[1 : 0] <= 2'b00;
        Ctrl_s5[1 : 0] <= 2'b00;
        Ctrl_s6[1 : 0] <= 2'b00;
        Ctrl_s7[1 : 0] <= 2'b00;
        Ctrl_s8[1 : 0] <= 2'b00;
      end
    endcase
    IndexCounter_o_tmp <= IndexCounter_i;
  end
else
  begin //push out and append zeros
    if (IndexCounter_i == 4'b0000)
      begin
        Ctrl_s1[1 : 0] <= 2'b01;
        Ctrl_s2[1 : 0] <= 2'b01;
        Ctrl_s3[1 : 0] <= 2'b01;
        Ctrl_s4[1 : 0] <= 2'b01;
        Ctrl_s5[1 : 0] <= 2'b01;
        Ctrl_s6[1 : 0] <= 2'b01;
        Ctrl_s7[1 : 0] <= 2'b01;
        Ctrl_s8[1 : 0] <= 2'b01;
```

```
        IndexCounter_o_tmp <= IndexCounter_i;
      end
    else
      begin
        Ctrl_s1[1 : 0] <= 2'b10;
        Ctrl_s2[1 : 0] <= 2'b10;
        Ctrl_s3[1 : 0] <= 2'b10;
        Ctrl_s4[1 : 0] <= 2'b10;
        Ctrl_s5[1 : 0] <= 2'b10;
        Ctrl_s6[1 : 0] <= 2'b10;
        Ctrl_s7[1 : 0] <= 2'b10;
        Ctrl_s8[1 : 0] <= 2'b01; //the last one is a little different
        IndexCounter_o_tmp <= IndexCounter_i - 1'b1;
      end
  end
else
  if (!full_i & wrreq) //append new data, and other DFFs remain
    begin
      if (IndexCounter_i == 4'b0000)
        Ctrl_s1[1 : 0] <= 2'b11; //write new data
      else
        Ctrl_s1[1 : 0] <= 2'b00; //remains
      if (IndexCounter_i == 4'b0001)
```

```
  Ctrl_s2[1 : 0] <= 2'b11;
else
  Ctrl_s2[1 : 0] <= 2'b00;
if (IndexCounter_i == 4'b0010)
  Ctrl_s3[1 : 0] <= 2'b11;
else
  Ctrl_s3[1 : 0] <= 2'b00;
if (IndexCounter_i == 4'b0011)
  Ctrl_s4[1 : 0] <= 2'b11;
else
  Ctrl_s4[1 : 0] <= 2'b00;
if (IndexCounter_i == 4'b0100)
  Ctrl_s5[1 : 0] <= 2'b11;
else
  Ctrl_s5[1 : 0] <= 2'b00;
if (IndexCounter_i == 4'b0101)
  Ctrl_s6[1 : 0] <= 2'b11;
else
  Ctrl_s6[1 : 0] <= 2'b00;
if (IndexCounter_i == 4'b0110)
  Ctrl_s7[1 : 0] <= 2'b11;
else
  Ctrl_s7[1 : 0] <= 2'b00;
```

```
        if (IndexCounter_i == 4'b0111)
          Ctrl_s8[1 : 0] <= 2'b11;
        else
          Ctrl_s8[1 : 0] <= 2'b00;
        IndexCounter_o_tmp <= IndexCounter_i + 1'b1; //need optimization
      end
    else // remains the same data
      begin
        Ctrl_s1[1 : 0] <= 2'b0;
        Ctrl_s2[1 : 0] <= 2'b0;
        Ctrl_s3[1 : 0] <= 2'b0;
        Ctrl_s4[1 : 0] <= 2'b0;
        Ctrl_s5[1 : 0] <= 2'b0;
        Ctrl_s6[1 : 0] <= 2'b0;
        Ctrl_s7[1 : 0] <= 2'b0;
        Ctrl_s8[1 : 0] <= 2'b0;
        IndexCounter_o_tmp <= IndexCounter_i;
      end
  end
always
  @(Ctrl_s1 or
    Ctrl_s2 or
    Ctrl_s3 or
```

```
  Ctrl_s4 or
  Ctrl_s5 or
  Ctrl_s6 or
  Ctrl_s7 or
  Ctrl_s8 or
  datain or
  fifo_output_s1 or
  fifo_output_s2 or
  fifo_output_s3 or
  fifo_output_s4 or
  fifo_output_s5 or
  fifo_output_s6 or
  fifo_output_s7 or
  fifo_output_s8)
begin
  case (Ctrl_s1)
    2'b00:
      fifo_input_s1 <= fifo_output_s1;
    2'b01:
      fifo_input_s1 <= 36'b0;
    2'b10:
      fifo_input_s1 <= fifo_output_s2;
    2'b11:
```

```
    fifo_input_s1 <= datain;
endcase
case (Ctrl_s2)
  2'b00:
    fifo_input_s2 <= fifo_output_s2;
  2'b01:
    fifo_input_s2 <= 36'b0;
  2'b10:
    fifo_input_s2 <= fifo_output_s3;
  2'b11:
    fifo_input_s2 <= datain;
endcase
case (Ctrl_s3)
  2'b00:
    fifo_input_s3 <= fifo_output_s3;
  2'b01:
    fifo_input_s3 <= 36'b0;
  2'b10:
    fifo_input_s3 <= fifo_output_s4;
  2'b11:
    fifo_input_s3 <= datain;
endcase
case (Ctrl_s4)
```

```
  2'b00:
    fifo_input_s4 <= fifo_output_s4;
  2'b01:
    fifo_input_s4 <= 36'b0;
  2'b10:
    fifo_input_s4 <= fifo_output_s5;
  2'b11:
    fifo_input_s4 <= datain;
endcase
case (Ctrl_s5)
  2'b00:
    fifo_input_s5 <= fifo_output_s5;
  2'b01:
    fifo_input_s5 <= 36'b0;
  2'b10:
    fifo_input_s5 <= fifo_output_s6;
  2'b11:
    fifo_input_s5 <= datain;
endcase
case (Ctrl_s6)
  2'b00:
    fifo_input_s6 <= fifo_output_s6;
  2'b01:
```

```
    fifo_input_s6 <= 36'b0;
  2'b10:
    fifo_input_s6 <= fifo_output_s7;
  2'b11:
    fifo_input_s6 <= datain;
endcase
case (Ctrl_s7)
  2'b00:
    fifo_input_s7 <= fifo_output_s7;
  2'b01:
    fifo_input_s7 <= 36'b0;
  2'b10:
    fifo_input_s7 <= fifo_output_s8;
  2'b11:
    fifo_input_s7 <= datain;
endcase
case (Ctrl_s8)
  2'b00:
    fifo_input_s8 <= fifo_output_s8;
  2'b01:
    fifo_input_s8 <= 36'b0;
  2'b10:
    fifo_input_s8 <= 36'b0;
```

```
      2'b11:
        fifo_input_s8 <= datain;
    endcase
  end
stagereg31_Proposed_EX QYFIFO_s1(.Preset_act_low(1'b1),
                                 .Clear_act_low(!reset),
                                 .CLK(clk),
                                 .DataIn(fifo_input_s1),
                                 .DataOut(fifo_output_s1));
stagereg31_Proposed_EX QYFIFO_s2(.Preset_act_low(1'b1),
                                 .Clear_act_low(!reset),
                                 .CLK(clk),
                                 .DataIn(fifo_input_s2),
                                 .DataOut(fifo_output_s2));
stagereg31_Proposed_EX QYFIFO_s3(.Preset_act_low(1'b1),
                                 .Clear_act_low(!reset),
                                 .CLK(clk),
                                 .DataIn(fifo_input_s3),
                                 .DataOut(fifo_output_s3));
stagereg31_Proposed_EX QYFIFO_s4(.Preset_act_low(1'b1),
                                 .Clear_act_low(!reset),
                                 .CLK(clk),
                                 .DataIn(fifo_input_s4),
```

```
                .DataOut(fifo_output_s4));
stagereg31_Proposed_EX QYFIFO_s5(.Preset_act_low(1'b1),
                .Clear_act_low(!reset),
                .CLK(clk),
                .DataIn(fifo_input_s5),
                .DataOut(fifo_output_s5));
stagereg31_Proposed_EX QYFIFO_s6(.Preset_act_low(1'b1),
                .Clear_act_low(!reset),
                .CLK(clk),
                .DataIn(fifo_input_s6),
                .DataOut(fifo_output_s6));
stagereg31_Proposed_EX QYFIFO_s7(.Preset_act_low(1'b1),
                .Clear_act_low(!reset),
                .CLK(clk),
                .DataIn(fifo_input_s7),
                .DataOut(fifo_output_s7));
stagereg31_Proposed_EX QYFIFO_s8(.Preset_act_low(1'b1),
                .Clear_act_low(!reset),
                .CLK(clk),
                .DataIn(fifo_input_s8),
                .DataOut(fifo_output_s8));

always
```

```
  @(fifo_output_s1)
  begin

    dataout = fifo_output_s1;
  end
 always
  @(posedge clk or
    posedge reset)
  begin
    if (reset)
      begin
        almost_full <= 1'b0;
        empty <= 1'b1;
        full_o <= 1'b0;
        IndexCounter_o <= 4'b0000;
      end
    else
      begin
        //edited by jonathan (almost_full to extend from 6FIFO to 8)
        almost_full <= (IndexCounter_o_tmp[3] & !IndexCounter_o_tmp[2] &
!IndexCounter_o_tmp[1] & !IndexCounter_o_tmp[0]);
        empty <= (IndexCounter_o_tmp == 4'b0000) ? 1'b1 : 1'b0;
        full_o <= (IndexCounter_o_tmp == 4'b1000) ? 1'b1 : 1'b0;
```

```
                IndexCounter_o <= IndexCounter_o_tmp;
            end
        end
endmodule
```

d. Full Crossbar

```
`timescale 1ns/1ns
/*
The functionality of the full crossbar are
(1)
Extract the information contained in each entered flit
(2)
Detect whether the intended the output port is available or not
(3)
If so, make connection
otherwise, wait for the available time slot
*/
module FullCrossbar4(CLK,
                     Reset,
                     select,
                     TableTag_pre,
                     RRReg_pre,
                     FlitIn_E,
                     FlitIn_S,
                     FlitIn_W,
                     FlitIn_N,
                     FlitIn_L,
                     CurrentID,
```

DestPort_pre_E,

DestPort_pre_S,

DestPort_pre_W,

DestPort_pre_N,

DestPort_pre_L,

NACK,

BufFull_in,

//output

TableTag_cur,

RRReg_cur,

FlitOut_E,

FlitOut_S,

FlitOut_W,

FlitOut_N,

FlitOut_L,

DestPort_E,

DestPort_S,

DestPort_W,

DestPort_N,

DestPort_L,

Port_Ad,

ErrEpre,

ErrLpre,

```
                ErrNpre,
                ErrSpre,
                ErrWpre //Before crossbar, important flit content [10:0] error detection
               );
parameter FLITWIDTH = 36;
parameter NOCSIZE = 4;
input[2 : 0] select;
input CLK;
input Reset;
input[24 : 0] TableTag_pre;
input[24 : 0] RRReg_pre;
input[FLITWIDTH - 1 : 0] FlitIn_E;
input[FLITWIDTH - 1 : 0] FlitIn_S;
input[FLITWIDTH - 1 : 0] FlitIn_W;
input[FLITWIDTH - 1 : 0] FlitIn_N;
input[FLITWIDTH - 1 : 0] FlitIn_L;
input[3 : 0] CurrentID;
input[2 : 0] DestPort_pre_E;
input[2 : 0] DestPort_pre_S;
input[2 : 0] DestPort_pre_W;
input[2 : 0] DestPort_pre_N;
input[2 : 0] DestPort_pre_L;
input[4 : 0] NACK;
```

input[4 : 0] BufFull_in;

output[24 : 0] TableTag_cur;

output[24 : 0] RRReg_cur;

output[FLITWIDTH - 1 : 0] FlitOut_E;

output[FLITWIDTH - 1 : 0] FlitOut_S;

output[FLITWIDTH - 1 : 0] FlitOut_W;

output[FLITWIDTH - 1 : 0] FlitOut_N;

output[FLITWIDTH - 1 : 0] FlitOut_L;

output[2 : 0] DestPort_E;

output[2 : 0] DestPort_S;

output[2 : 0] DestPort_W;

output[2 : 0] DestPort_N;

output[2 : 0] DestPort_L;

output[24 : 0] Port_Ad;

wire CLK;

wire Reset;

wire[24 : 0] TableTag_pre;

wire[24 : 0] RRReg_pre;

wire[FLITWIDTH - 1 : 0] FlitIn_E;

wire[FLITWIDTH - 1 : 0] FlitIn_S;

wire[FLITWIDTH - 1 : 0] FlitIn_W;

wire[FLITWIDTH - 1 : 0] FlitIn_N;

wire[FLITWIDTH - 1 : 0] FlitIn_L;

```
wire[2 : 0] DestPort_pre_E;
wire[2 : 0] DestPort_pre_S;
wire[2 : 0] DestPort_pre_W;
wire[2 : 0] DestPort_pre_N;
wire[2 : 0] DestPort_pre_L;
wire[3 : 0] CurrentID;
wire[4 : 0] NACK;
wire[4 : 0] BufFull_in;
reg[24 : 0] TableTag_cur;
reg[24 : 0] RRReg_cur;
reg[FLITWIDTH - 1 : 0] FlitOut_E;
reg[FLITWIDTH - 1 : 0] FlitOut_S;
reg[FLITWIDTH - 1 : 0] FlitOut_W;
reg[FLITWIDTH - 1 : 0] FlitOut_N;
reg[FLITWIDTH - 1 : 0] FlitOut_L;
reg[2 : 0] DestPort_E;
reg[2 : 0] DestPort_S;
reg[2 : 0] DestPort_W;
reg[2 : 0] DestPort_N;
reg[2 : 0] DestPort_L;
reg[24 : 0] Port_Ad;
//internal signals
wire[4 : 0] Req_V_E;
```

```
wire[4 : 0] CReq_V_E;

wire[4 : 0] Req_V_S;

wire[4 : 0] CReq_V_S;

wire[4 : 0] Req_V_W;

wire[4 : 0] CReq_V_W;

wire[4 : 0] Req_V_N;

wire[4 : 0] CReq_V_N;

wire[4 : 0] Req_V_L;

wire[4 : 0] CReq_V_L;

wire[2 : 0] DestPort_E_tmp;

wire[2 : 0] DestPort_S_tmp;

wire[2 : 0] DestPort_W_tmp;

wire[2 : 0] DestPort_N_tmp;

wire[2 : 0] DestPort_L_tmp;

wire[4 : 0] TableTag_pre_oE;

wire[4 : 0] TableTag_pre_oS;

wire[4 : 0] TableTag_pre_oW;

wire[4 : 0] TableTag_pre_oN;

wire[4 : 0] TableTag_pre_oL;

wire[4 : 0] RRReg_pre_oE;

wire[4 : 0] RRReg_pre_oS;

wire[4 : 0] RRReg_pre_oW;

wire[4 : 0] RRReg_pre_oN;
```

```
wire[4 : 0] RRReg_pre_oL;
wire NACK_oE;
wire NACK_oS;
wire NACK_oW;
wire NACK_oN;
wire NACK_oL;
wire BufFull_in_oE;
wire BufFull_in_oS;
wire BufFull_in_oW;
wire BufFull_in_oN;
wire BufFull_in_oL;
wire[4 : 0] TableTag_new_oE;
wire[4 : 0] TableTag_new_oS;
wire[4 : 0] TableTag_new_oW;
wire[4 : 0] TableTag_new_oN;
wire[4 : 0] TableTag_new_oL;
wire[4 : 0] RRReg_new_oE;
wire[4 : 0] RRReg_new_oS;
wire[4 : 0] RRReg_new_oW;
wire[4 : 0] RRReg_new_oN;
wire[4 : 0] RRReg_new_oL;
wire[4 : 0] Port_Ad_oE;
wire[4 : 0] Port_Ad_oS;
```

```
wire[4 : 0] Port_Ad_oW;
wire[4 : 0] Port_Ad_oN;
wire[4 : 0] Port_Ad_oL;
reg[FLITWIDTH - 1 : 0] FlitOut_E_tmp;
reg[FLITWIDTH - 1 : 0] FlitOut_S_tmp;
reg[FLITWIDTH - 1 : 0] FlitOut_W_tmp;
reg[FLITWIDTH - 1 : 0] FlitOut_N_tmp;
reg[FLITWIDTH - 1 : 0] FlitOut_L_tmp;
output ErrEpre, ErrLpre, ErrNpre, ErrSpre, ErrWpre;
assign TableTag_pre_oE = TableTag_pre[4 : 0];
assign TableTag_pre_oS = TableTag_pre[9 : 5];
assign TableTag_pre_oW = TableTag_pre[14 : 10];
assign TableTag_pre_oN = TableTag_pre[19 : 15];
assign TableTag_pre_oL = TableTag_pre[24 : 20];
assign RRReg_pre_oE = RRReg_pre[4 : 0];
assign RRReg_pre_oS = RRReg_pre[9 : 5];
assign RRReg_pre_oW = RRReg_pre[14 : 10];
assign RRReg_pre_oN = RRReg_pre[19 : 15];
assign RRReg_pre_oL = RRReg_pre[24 : 20];
assign NACK_oE = NACK[0];
assign NACK_oS = NACK[1];
assign NACK_oW = NACK[2];
assign NACK_oN = NACK[3];
```

```
assign NACK_oL = NACK[4];
assign BufFull_in_oE = BufFull_in[0];
assign BufFull_in_oS = BufFull_in[1];
assign BufFull_in_oW = BufFull_in[2];
assign BufFull_in_oN = BufFull_in[3];
assign BufFull_in_oL = BufFull_in[4];

InfoExtract4 FlitBreakE(.Flit(FlitIn_E),
                        .CurrentID(CurrentID),
                        .DestPort_pre(DestPort_pre_E),
                        .select(select),
                        //output
                        .DestPort(DestPort_E_tmp),
                        .Req_V(Req_V_E),
                        .CReq_V(CReq_V_E),
                        .ErrDetectionOut(ErrEpre));
InfoExtract4 FlitBreakS(.Flit(FlitIn_S),
                        .CurrentID(CurrentID),
                        .DestPort_pre(DestPort_pre_S),
                        .select(select),
                        //output
                        .DestPort(DestPort_S_tmp),
                        .Req_V(Req_V_S),
```

```
                .CReq_V(CReq_V_S),
                .ErrDetectionOut(ErrSpre));
InfoExtract4 FlitBreakW(.Flit(FlitIn_W),
                .CurrentID(CurrentID),
                .DestPort_pre(DestPort_pre_W),
                .select(select),
                //output
                .DestPort(DestPort_W_tmp),
                .Req_V(Req_V_W),
                .CReq_V(CReq_V_W),
                .ErrDetectionOut(ErrWpre));
InfoExtract4 FlitBreakN(.Flit(FlitIn_N),
                .CurrentID(CurrentID),
                .DestPort_pre(DestPort_pre_N),
                .select(select),
                //output
                .DestPort(DestPort_N_tmp),
                .Req_V(Req_V_N),
                .CReq_V(CReq_V_N),
                .ErrDetectionOut(ErrNpre));
InfoExtract4 FlitBreakL(.Flit(FlitIn_L),
                .CurrentID(CurrentID),
                .DestPort_pre(DestPort_pre_L),
```

```
                .select(select),
                //output
                .DestPort(DestPort_L_tmp),
                .Req_V(Req_V_L),
                .CReq_V(CReq_V_L),
                .ErrDetectionOut(ErrLpre));
  //Round Robin In/Out port connection
  //Rule-1: no 180 degree transmission (i.e. the data from east input port cannot be transfered via east output port immediately)
  //assign Req_V_oE = {Req_V_E[0], Req_V_S[0],Req_V_W[0],Req_V_N[0],Req_V_L[0]};
  PortAdmission PortAd_E(
                //.Req_V({Req_V_E[0], Req_V_S[0],Req_V_W[0],Req_V_N[0],Req_V_L[0]}),
                //.CReq_V({CReq_V_E[0], CReq_V_S[0],CReq_V_W[0],CReq_V_N[0],CReq_V_L[0]}),
                .Req_V({ Req_V_L[0], Req_V_N[0], Req_V_W[0], Req_V_S[0], Req_V_E[0] }),
                .CReq_V({ CReq_V_L[0], CReq_V_N[0], CReq_V_W[0], CReq_V_S[0], CReq_V_E[0] }),
                .TableTag_pre(TableTag_pre_oE),
                .RRReg_pre(RRReg_pre_oE),
                .NACK(NACK_oE),
```

```
        .BufFull_in(BufFull_in_oE),
        //output
        //.Port_Ad({Port_Ad_oE[0], Port_Ad_oS[0], Port_Ad_oW[0],
Port_Ad_oN[0], Port_Ad_oL[0]}), //the direction with 'o' means that it is the output
direction
        .Port_Ad({ Port_Ad_oL[0], Port_Ad_oN[0], Port_Ad_oW[0],
Port_Ad_oS[0], Port_Ad_oE[0] }), //Port_Ad_oX[Y]: Y is the output port direction, X is
the data source direction
        .TableTag_new(TableTag_new_oE),
        .RRReg_new(RRReg_new_oE));
 PortAdmission PortAd_S(
        .Req_V({ Req_V_L[1], Req_V_N[1], Req_V_W[1], Req_V_S[1],
Req_V_E[1] }),
        .CReq_V({ CReq_V_L[1], CReq_V_N[1], CReq_V_W[1], CReq_V_S[1],
CReq_V_E[1] }),
        .TableTag_pre(TableTag_pre_oS),
        .RRReg_pre(RRReg_pre_oS),
        .NACK(NACK_oS),
        .BufFull_in(BufFull_in_oS),
        //output
        .Port_Ad({ Port_Ad_oL[1], Port_Ad_oN[1], Port_Ad_oW[1],
Port_Ad_oS[1], Port_Ad_oE[1] }),
        .TableTag_new(TableTag_new_oS),
```

```
                .RRReg_new(RRReg_new_oS));
  PortAdmission PortAd_W(
                .Req_V({ Req_V_L[2], Req_V_N[2], Req_V_W[2], Req_V_S[2],
Req_V_E[2] }),
                .CReq_V({ CReq_V_L[2], CReq_V_N[2], CReq_V_W[2], CReq_V_S[2],
CReq_V_E[2] }),
                .TableTag_pre(TableTag_pre_oW),
                .RRReg_pre(RRReg_pre_oW),
                .NACK(NACK_oW),
                .BufFull_in(BufFull_in_oW),
                //output
                .Port_Ad({ Port_Ad_oL[2], Port_Ad_oN[2], Port_Ad_oW[2],
Port_Ad_oS[2], Port_Ad_oE[2] }),
                .TableTag_new(TableTag_new_oW),
                .RRReg_new(RRReg_new_oW));
  PortAdmission PortAd_N(
                .Req_V({ Req_V_L[3], Req_V_N[3], Req_V_W[3], Req_V_S[3],
Req_V_E[3] }),
                .CReq_V({ CReq_V_L[3], CReq_V_N[3], CReq_V_W[3], CReq_V_S[3],
CReq_V_E[3] }),
                .TableTag_pre(TableTag_pre_oN),
                .RRReg_pre(RRReg_pre_oN),
                .NACK(NACK_oN),
```

```
                .BufFull_in(BufFull_in_oN),
                //output
                .Port_Ad({ Port_Ad_oL[3], Port_Ad_oN[3], Port_Ad_oW[3],
Port_Ad_oS[3], Port_Ad_oE[3] }),
                .TableTag_new(TableTag_new_oN),
                .RRReg_new(RRReg_new_oN));
 PortAdmission PortAd_L(
                .Req_V({ Req_V_L[4], Req_V_N[4], Req_V_W[4], Req_V_S[4],
Req_V_E[4] }),
                .CReq_V({ CReq_V_L[4], CReq_V_N[4], CReq_V_W[4], CReq_V_S[4],
CReq_V_E[4] }),
                .TableTag_pre(TableTag_pre_oL),
                .RRReg_pre(RRReg_pre_oL),
                .NACK(NACK_oL),
                .BufFull_in(BufFull_in_oL),
                //output
                .Port_Ad({ Port_Ad_oL[4], Port_Ad_oN[4], Port_Ad_oW[4],
Port_Ad_oS[4], Port_Ad_oE[4] }),
                .TableTag_new(TableTag_new_oL),
                .RRReg_new(RRReg_new_oL));
 //use Port_Ad_oE,S,W,N,L to make connection
 always
  @(Port_Ad_oE or
```

```
    Port_Ad_oS or
    Port_Ad_oW or
    Port_Ad_oN or
    Port_Ad_oL or
    FlitIn_E or
    FlitIn_S or
    FlitIn_W or
    FlitIn_N or
    FlitIn_L)
  begin
    case ({ Port_Ad_oE[0], Port_Ad_oS[0], Port_Ad_oW[0], Port_Ad_oN[0],
Port_Ad_oL[0] })
      5'b00001:
        FlitOut_E_tmp <= FlitIn_L;
      5'b00010:
        FlitOut_E_tmp <= FlitIn_N;
      5'b00100:
        FlitOut_E_tmp <= FlitIn_W;
      5'b01000:
        FlitOut_E_tmp <= FlitIn_S;
      5'b10000:
        FlitOut_E_tmp <= FlitIn_E;
      default:
```

```
        FlitOut_E_tmp <= 0;
    endcase
    case ({ Port_Ad_oE[1], Port_Ad_oS[1], Port_Ad_oW[1], Port_Ad_oN[1],
Port_Ad_oL[1] })
      5'b00001:
        FlitOut_S_tmp <= FlitIn_L;
      5'b00010:
        FlitOut_S_tmp <= FlitIn_N;
      5'b00100:
        FlitOut_S_tmp <= FlitIn_W;
      5'b01000:
        FlitOut_S_tmp <= FlitIn_S;
      5'b10000:
        FlitOut_S_tmp <= FlitIn_E;
      default:
        FlitOut_S_tmp <= 0;
    endcase
    case ({ Port_Ad_oE[2], Port_Ad_oS[2], Port_Ad_oW[2], Port_Ad_oN[2],
Port_Ad_oL[2] })
      5'b00001:
        FlitOut_W_tmp <= FlitIn_L;
      5'b00010:
        FlitOut_W_tmp <= FlitIn_N;
```

```
        5'b00100:
          FlitOut_W_tmp <= FlitIn_W;
        5'b01000:
          FlitOut_W_tmp <= FlitIn_S;
        5'b10000:
          FlitOut_W_tmp <= FlitIn_E;
        default:
          FlitOut_W_tmp <= 0;
      endcase
      case ({ Port_Ad_oE[3], Port_Ad_oS[3], Port_Ad_oW[3], Port_Ad_oN[3],
Port_Ad_oL[3] })
        5'b00001:
          FlitOut_N_tmp <= FlitIn_L;
        5'b00010:
          FlitOut_N_tmp <= FlitIn_N;
        5'b00100:
          FlitOut_N_tmp <= FlitIn_W;
        5'b01000:
          FlitOut_N_tmp <= FlitIn_S;
        5'b10000:
          FlitOut_N_tmp <= FlitIn_E;
        default:
          FlitOut_N_tmp <= 0;
```

```
    endcase
    case ({ Port_Ad_oE[4], Port_Ad_oS[4], Port_Ad_oW[4], Port_Ad_oN[4],
Port_Ad_oL[4] })
      5'b00001:
        FlitOut_L_tmp <= FlitIn_L;
      5'b00010:
        FlitOut_L_tmp <= FlitIn_N;
      5'b00100:
        FlitOut_L_tmp <= FlitIn_W;
      5'b01000:
        FlitOut_L_tmp <= FlitIn_S;
      5'b10000:
        FlitOut_L_tmp <= FlitIn_E;
      default:
        FlitOut_L_tmp <= 0;
    endcase
  end
 always
  @(Reset or
    Port_Ad_oE or
    Port_Ad_oS or
    Port_Ad_oW or
    Port_Ad_oN or
```

```
    Port_Ad_oL)
  begin
    if (Reset)
      Port_Ad <= 25'b0;
    else
      begin
        Port_Ad[4 : 0] <= { Port_Ad_oE[0], Port_Ad_oS[0], Port_Ad_oW[0],
Port_Ad_oN[0], Port_Ad_oL[0] };
        Port_Ad[9 : 5] <= { Port_Ad_oE[1], Port_Ad_oS[1], Port_Ad_oW[1],
Port_Ad_oN[1], Port_Ad_oL[1] };
        Port_Ad[14 : 10] <= { Port_Ad_oE[2], Port_Ad_oS[2], Port_Ad_oW[2],
Port_Ad_oN[2], Port_Ad_oL[2] };
        Port_Ad[19 : 15] <= { Port_Ad_oE[3], Port_Ad_oS[3], Port_Ad_oW[3],
Port_Ad_oN[3], Port_Ad_oL[3] };
        Port_Ad[24 : 20] <= { Port_Ad_oE[4], Port_Ad_oS[4], Port_Ad_oW[4],
Port_Ad_oN[4], Port_Ad_oL[4] };
      end
  end
 /* */
 always
  @(posedge CLK)
  begin
    if (Reset)
```

```
  begin
    DestPort_E <= 3'b0;
    DestPort_S <= 3'b0;
    DestPort_W <= 3'b0;
    DestPort_N <= 3'b0;
    DestPort_L <= 3'b0;
    TableTag_cur <= 25'b0;
    RRReg_cur <= 25'b0;
    //Port_Ad <= 25'b0;
    FlitOut_E <= 0;
    FlitOut_S <= 0;
    FlitOut_W <= 0;
    FlitOut_N <= 0;
    FlitOut_L <= 0;
  end
else
  begin
    DestPort_E <= DestPort_E_tmp;
    DestPort_S <= DestPort_S_tmp;
    DestPort_W <= DestPort_W_tmp;
    DestPort_N <= DestPort_N_tmp;
    DestPort_L <= DestPort_L_tmp;
```

```
            TableTag_cur <= { TableTag_new_oL, TableTag_new_oN, TableTag_new_oW,
        TableTag_new_oS, TableTag_new_oE };
            RRReg_cur <= { RRReg_new_oL, RRReg_new_oN, RRReg_new_oW,
        RRReg_new_oS, RRReg_new_oE };
            FlitOut_E <= FlitOut_E_tmp;
            FlitOut_S <= FlitOut_S_tmp;
            FlitOut_W <= FlitOut_W_tmp;
            FlitOut_N <= FlitOut_N_tmp;
            FlitOut_L <= FlitOut_L_tmp;
          end
        end
endmodule
```

e. Info Extraction

```
`timescale 1ns/1ns
module InfoExtract4(Flit,
                    CurrentID,
                    DestPort_pre,
                    select,
                    //output
                    DestPort,
                    Req_V,
                    CReq_V,
                    ErrDetectionOut);
  parameter FLITWIDTH = 36;
  parameter NOCSIZE = 16;
  input[FLITWIDTH - 1 : 0] Flit;
  input[3 : 0] CurrentID;
  input[2 : 0] DestPort_pre;
  input[2 : 0] select;
  output[2 : 0] DestPort;
  output[4 : 0] Req_V;
  output[4 : 0] CReq_V;
  wire[FLITWIDTH - 1 : 0] Flit;
  wire[3 : 0] CurrentID;
  wire[2 : 0] DestPort_pre;
```

```
reg[2 : 0] DestPort;
reg[4 : 0] Req_V;
reg[4 : 0] CReq_V;
wire[14 : 0] Info_enc;
wire[10 : 0] Info;
//internal signal
reg Req;
reg CReq;
reg[3 : 0] DestID;
reg[1 : 0] X_C;
reg[1 : 0] X_D;
reg[1 : 0] Y_C;
reg[1 : 0] Y_D;
output ErrDetectionOut;
PFDS p0(.inputa(Flit[FLITWIDTH - 1 : 0]),
        .select(select),
        .outputb(Info_enc[14 : 0]));
HammingDecoder15_11_Partial D0(.DataIno(Info_enc[14 : 0]),
                               .DataOut(Info[10 : 0]),
                               .ErrDetection(ErrDetectionOut));

always
  @(Info or
```

```
  Flit)
 begin
  if ((Info[0] == 1'b1) & (Info[1] == 1'b0))
    begin
      Req = 1'b1;
      CReq = 1'b0;
      DestID = Info[9 : 6];
    end
  else
    if ((Info[0] == 1'b0) & (Info[1] == 1'b1))
      begin
        Req = 1'b0;
        CReq = 1'b1;
        DestID = 4'b0;
      end
    else
      begin
        Req = 1'b0;
        CReq = 1'b0;
        DestID = 4'b0;
      end
 end
//Compute Destport
```

```
always
  @(CurrentID or
    DestID) //X_C[0] is gMSB, same with Y_C[0]
  begin
    //NoC size 4x4
    if (NOCSIZE == 9)
      begin
        case (CurrentID)
          4'b0000:
            begin
              Y_C = 2'b00;
              X_C = 2'b00;
            end
          4'b0001:
            begin
              Y_C = 2'b00;
              X_C = 2'b01;
            end
          4'b0010:
            begin
              Y_C = 2'b00;
              X_C = 2'b10;
            end
```

```
4'b0011:
  begin
    Y_C = 2'b01;
    X_C = 2'b00;
  end
4'b0100:
  begin
    Y_C = 2'b01;
    X_C = 2'b01;
  end
4'b0101:
  begin
    Y_C = 2'b01;
    X_C = 2'b10;
  end
4'b0110:
  begin
    Y_C = 2'b10;
    X_C = 2'b00;
  end
4'b0111:
  begin
    Y_C = 2'b10;
```

```
      X_C = 2'b01;
    end
  4'b1000:
    begin
      Y_C = 2'b10;
      X_C = 2'b10;
    end
  default:
    begin
      Y_C = 2'b00;
      X_C = 2'b00;
    end
endcase
case (DestID)
  4'b0000:
    begin
      Y_D = 2'b00;
      X_D = 2'b00;
    end
  4'b0001:
    begin
      Y_D = 2'b00;
      X_D = 2'b01;
```

```
  end
4'b0010:
  begin
    Y_D = 2'b00;
    X_D = 2'b10;
  end
4'b0011:
  begin
    Y_D = 2'b01;
    X_D = 2'b00;
  end
4'b0100:
  begin
    Y_D = 2'b01;
    X_D = 2'b01;
  end
4'b0101:
  begin
    Y_D = 2'b01;
    X_D = 2'b10;
  end
4'b0110:
  begin
```

```
        Y_D = 2'b10;
        X_D = 2'b00;
      end
    4'b0111:
      begin
        Y_D = 2'b10;
        X_D = 2'b01;
      end
    4'b1000:
      begin
        Y_D = 2'b10;
        X_D = 2'b10;
      end
    default:
      begin
        Y_D = 2'b00;
        X_D = 2'b00;
      end
  endcase
 end
else //NOCSIZE==16
 begin
  case (CurrentID)
```

```
4'b0000:
  begin
    Y_C = 2'b00;
    X_C = 2'b00;
  end
4'b0001:
  begin
    Y_C = 2'b00;
    X_C = 2'b01;
  end
4'b0010:
  begin
    Y_C = 2'b00;
    X_C = 2'b10;
  end
4'b0011:
  begin
    Y_C = 2'b00;
    X_C = 2'b11;
  end
4'b0100:
  begin
    Y_C = 2'b01;
```

```
    X_C = 2'b00;
  end
4'b0101:
  begin
    Y_C = 2'b01;
    X_C = 2'b01;
  end
4'b0110:
  begin
    Y_C = 2'b01;
    X_C = 2'b10;
  end
4'b0111:
  begin
    Y_C = 2'b01;
    X_C = 2'b11;
  end
4'b1000:
  begin
    Y_C = 2'b10;
    X_C = 2'b00;
  end
4'b1001:
```

```
  begin
    Y_C = 2'b10;
    X_C = 2'b01;
  end
4'b1010:
  begin
    Y_C = 2'b10;
    X_C = 2'b10;
  end
4'b1011:
  begin
    Y_C = 2'b10;
    X_C = 2'b11;
  end
4'b1100:
  begin
    Y_C = 2'b11;
    X_C = 2'b00;
  end
4'b1101:
  begin
    Y_C = 2'b11;
    X_C = 2'b01;
```

```
    end
  4'b1110:
    begin
      Y_C = 2'b11;
      X_C = 2'b10;
    end
  4'b1111:
    begin
      Y_C = 2'b11;
      X_C = 2'b11;
    end
  default:
    begin
      Y_C = 2'b00;
      X_C = 2'b00;
    end
endcase
case (DestID)
  4'b0000:
    begin
      Y_D = 2'b00;
      X_D = 2'b00;
    end
```

```
4'b0001:
  begin
    Y_D = 2'b00;
    X_D = 2'b01;
  end
4'b0010:
  begin
    Y_D = 2'b00;
    X_D = 2'b10;
  end
4'b0011:
  begin
    Y_D = 2'b00;
    X_D = 2'b11;
  end
4'b0100:
  begin
    Y_D = 2'b01;
    X_D = 2'b00;
  end
4'b0101:
  begin
    Y_D = 2'b01;
```

```
    X_D = 2'b01;
  end
4'b0110:
  begin
    Y_D = 2'b01;
    X_D = 2'b10;
  end
4'b0111:
  begin
    Y_D = 2'b01;
    X_D = 2'b11;
  end
4'b1000:
  begin
    Y_D = 2'b10;
    X_D = 2'b00;
  end
4'b1001:
  begin
    Y_D = 2'b10;
    X_D = 2'b01;
  end
4'b1010:
```

```
  begin
    Y_D = 2'b10;
    X_D = 2'b10;
  end
4'b1011:
  begin
    Y_D = 2'b10;
    X_D = 2'b11;
  end
4'b1100:
  begin
    Y_D = 2'b11;
    X_D = 2'b00;
  end
4'b1101:
  begin
    Y_D = 2'b11;
    X_D = 2'b01;
  end
4'b1110:
  begin
    Y_D = 2'b11;
    X_D = 2'b10;
```

```
            end
          4'b1111:
            begin
              Y_D = 2'b11;
              X_D = 2'b11;
            end
          default:
            begin
              Y_D = 2'b00;
              X_D = 2'b00;
            end
        endcase
      end
      end
always
  @(X_C or
    Y_C or
    X_D or
    Y_D or
    Req or
    DestPort_pre)
  begin
    if (Req)
```

```
if (X_C == X_D) //move on y-direction, 3'b100, local port. DestPort[2] is MSB
  begin
    if (Y_C == Y_D)
      DestPort = 3'b100;
    else
      begin

        case ({ Y_C, Y_D }) //design for 3x3
          /*4'b0001, 4'b0110,*/
          4'b1000, 4'b0100, 4'b1001, 4'b1100, 4'b1101, 4'b1110:
            DestPort = 3'b011; //go through north
          /*4'b0100, 4'b1001,*/
          4'b0010, 4'b0001, 4'b0110, 4'b0011, 4'b1011, 4'b0111:
            DestPort = 3'b001; //go through south
          default:
            DestPort = 3'b100; //go through local
        endcase


      end
  end
else //move on x-direction
  begin
```

```
        case ({ X_C, X_D })
          4'b0001, 4'b0110, 4'b0010, 4'b0011, 4'b0111, 4'b1011:
            DestPort = 3'b000; //go through east
          4'b0100, 4'b1001, 4'b1000, 4'b1100, 4'b1101, 4'b1110:
            DestPort = 3'b010; //go through west
          default:
            DestPort = 3'b100; // go through local
        endcase
      end
    else
      DestPort = DestPort_pre;
  end
always
  @(DestPort or
    DestPort_pre or
    Req or
    CReq)
  begin
    if (Req)
      case (DestPort)
        3'b000:
          Req_V = { 4'b0, 1'b1 };
        3'b001:
```

```
      Req_V = { 3'b0, 1'b1, 1'b0 };
    3'b010:
      Req_V = { 2'b0, 1'b1, 2'b0 };
    3'b011:
      Req_V = { 1'b0, 1'b1, 3'b0 };
    3'b100:
      Req_V = { 1'b1, 4'b0 };
    default:
      Req_V = { 4'b0, 1'b1 };
  endcase
else
  Req_V = 5'b00000;
if (CReq)
  case (DestPort_pre)
    3'b000:
      CReq_V = { 4'b0, 1'b1 };
    3'b001:
      CReq_V = { 3'b0, 1'b1, 1'b0 };
    3'b010:
      CReq_V = { 2'b0, 1'b1, 2'b0 };
    3'b011:
      CReq_V = { 1'b0, 1'b1, 3'b0 };
    3'b100:
```

```
            CReq_V = { 1'b1, 4'b0 };
          default:
            CReq_V = { 4'b0, 1'b1 };
        endcase
      else
        CReq_V = 5'b00000;
    end
endmodule
```

f. Port Admission

```
`timescale 1ns/1ns
module PortAdmission(Req_V,
                     CReq_V,
                     TableTag_pre,
                     RRReg_pre,
                     NACK,
                     BufFull_in,
                     //output
                     Port_Ad, //the direction with 'o' means that it is the output direction
                     TableTag_new,
                     RRReg_new);
  input[4 : 0] Req_V; //changed msb
  input[4 : 0] CReq_V; //changed msb
  input[4 : 0] TableTag_pre;
  input[4 : 0] RRReg_pre;
  input NACK;
  input BufFull_in;
  output[4 : 0] Port_Ad;
  output[4 : 0] TableTag_new;
  output[4 : 0] RRReg_new;
  wire[4 : 0] Req_V; //changed msb
  wire[4 : 0] CReq_V; //changed msb
```

```
wire[4 : 0] TableTag_pre;
wire[4 : 0] RRReg_pre;
wire NACK;
wire BufFull_in;
reg[4 : 0] Port_Ad;
reg[4 : 0] TableTag_new;
reg[4 : 0] RRReg_new;
//internal signals
wire[4 : 0] RR_Port_Ad;
wire[4 : 0] RR_RRReg;
wire[4 : 0] RR_TableTag;
RoundRobin RR(.Req_V(Req_V),
              .RRReg_in(RRReg_pre),
              .TableTag_in(TableTag_pre),
              //output
              .RRReg_out(RR_RRReg),
              .TableTag_out(RR_TableTag),
              .Port_Ad(RR_Port_Ad));
always
  @(NACK or
    BufFull_in or
    TableTag_pre or
    RRReg_pre or
```

```
    CReq_V or
    RR_Port_Ad or
    RR_RRReg or
    RR_TableTag)
  begin
    if (NACK | BufFull_in)
      begin
        TableTag_new <= TableTag_pre;
        RRReg_new <= RRReg_pre;
        Port_Ad <= 5'b0;
      end
    else
      if (TableTag_pre[0] | TableTag_pre[1] | TableTag_pre[2] | TableTag_pre[3] |
TableTag_pre[4])
        begin
          Port_Ad <= RRReg_pre;
          RRReg_new <= RRReg_pre;
          if (CReq_V[0] | CReq_V[1] | CReq_V[2] | CReq_V[3] | CReq_V[4])
            TableTag_new <= 5'b0;
          else
            TableTag_new <= TableTag_pre;
        end
      else
```

```
                begin
                  Port_Ad <= RR_Port_Ad;
                  RRReg_new <= RR_RRReg;
                  TableTag_new <= RR_TableTag;
                end
            end
endmodule
```

g. Hamming Decoder

```
`timescale 1ns/1ns
module HammingDecoder15_11(DataIno,
                           DataOut,
                           ErrDetection);
  input[35 : 0] DataIno;
  output[31 : 0] DataOut;
  wire[35 : 0] DataIno;
  wire[3 : 0] ECC;
  reg[31 : 0] DataOut;
  output reg ErrDetection;
  //internal reg
  reg[3 : 0] syndrome;
  assign ECC[3 : 0] = DataIno[14 : 11];
  always
    @(DataIno or
      ECC)
    begin
      syndrome[0] <= ECC[0] ^ DataIno[0] ^ DataIno[3] ^ DataIno[4] ^ DataIno[6] ^
DataIno[8] ^ DataIno[9] ^ DataIno[10];
      syndrome[1] <= ECC[1] ^ DataIno[0] ^ DataIno[1] ^ DataIno[3] ^ DataIno[5] ^
DataIno[6] ^ DataIno[7] ^ DataIno[8];
```

```
    syndrome[2] <= ECC[2] ^ DataIno[1] ^ DataIno[2] ^ DataIno[4] ^ DataIno[6] ^
DataIno[7] ^ DataIno[8] ^ DataIno[9];
    syndrome[3] <= ECC[3] ^ DataIno[2] ^ DataIno[3] ^ DataIno[5] ^ DataIno[7] ^
DataIno[8] ^ DataIno[9] ^ DataIno[10];
  end
 always
  @(syndrome or
    DataIno)
  begin
    ErrDetection <= (syndrome[3] | syndrome[2] | syndrome[1] | syndrome[0]);
    DataOut[10] <= DataIno[10] ^ ((syndrome[0]) & (~syndrome[1]) & (~syndrome[2]) &
(syndrome[3])); //1001 syn[0]-->syn[3]
    DataOut[9] <= DataIno[9] ^ ((syndrome[0]) & (~syndrome[1]) & (syndrome[2]) &
(syndrome[3])); //1011
    DataOut[8] <= DataIno[8] ^ ((syndrome[0]) & (syndrome[1]) & (syndrome[2]) &
(syndrome[3])); //1111
    DataOut[7] <= DataIno[7] ^ ((~syndrome[0]) & (syndrome[1]) & (syndrome[2]) &
(syndrome[3])); //0111
    DataOut[6] <= DataIno[6] ^ ((syndrome[0]) & (syndrome[1]) & (syndrome[2]) &
(~syndrome[3])); //1110
    DataOut[5] <= DataIno[5] ^ ((~syndrome[0]) & (syndrome[1]) & (~syndrome[2]) &
(syndrome[3])); //0101
```

```
        DataOut[4] <= DataIno[4] ^ ((syndrome[0]) & (~syndrome[1]) & (syndrome[2]) &
    (~syndrome[3])); //1010
        DataOut[3] <= DataIno[3] ^ ((syndrome[0]) & (syndrome[1]) & (~syndrome[2]) &
    (syndrome[3])); //1101
        DataOut[2] <= DataIno[2] ^ ((~syndrome[0]) & (~syndrome[1]) & (syndrome[2]) &
    (syndrome[3])); //0011
        DataOut[1] <= DataIno[1] ^ ((~syndrome[0]) & (syndrome[1]) & (syndrome[2]) &
    (~syndrome[3])); //0110
        DataOut[0] <= DataIno[0] ^ ((syndrome[0]) & (syndrome[1]) & (~syndrome[2]) &
    (~syndrome[3])); //1100
        DataOut[31 : 11] <= DataIno[35 : 15];
      end
endmodule
```

h. Hamming Encoder

```
`timescale 1ns/1ns
module HammingEncoder15_11(DataIn,
                           DataOut);
  input[31 : 0] DataIn;
  output[35 : 0] DataOut;
  reg[35 : 0] DataOut;
  always
    @(DataIn)
    begin
      DataOut[10 : 0] <= DataIn[10 : 0];
      DataOut[35 : 15] <= DataIn[31 : 11];
      DataOut[11] <= DataIn[0] ^ DataIn[3] ^ DataIn[4] ^ DataIn[6] ^ DataIn[8] ^ DataIn[9]
^ DataIn[10];
      DataOut[12] <= DataIn[0] ^ DataIn[1] ^ DataIn[3] ^ DataIn[5] ^ DataIn[6] ^ DataIn[7]
^ DataIn[8];
      DataOut[13] <= DataIn[1] ^ DataIn[2] ^ DataIn[4] ^ DataIn[6] ^ DataIn[7] ^ DataIn[8]
^ DataIn[9];
      DataOut[14] <= DataIn[2] ^ DataIn[3] ^ DataIn[5] ^ DataIn[7] ^ DataIn[8] ^ DataIn[9]
^ DataIn[10];
    end
endmodule
```

i. Mux8

```
`timescale 1ns/1ns
module mux8(select,
            d,
            q);
  input[2 : 0] select;
  input[7 : 0] d;
  output q;
  reg q;
  wire[2 : 0] select;
  wire[7 : 0] d;
  always
    @(select or
      d)
    begin
      case (select)
        3'b000:
          q = d[0];
        3'b001:
          q = d[1];
        3'b010:
          q = d[2];
        3'b011:
```

```
            q = d[3];
          3'b100:
            q = d[4];
          3'b101:
            q = d[5];
          3'b110:
            q = d[6];
          3'b111:
            q = d[7];
        endcase
      end
endmodule
```

j. Partial Flit De-Permutation

```
`timescale 1ns/1ns
module PFDS(inputa,
            select,
            outputb);
  input[35 : 0] inputa;
  input[2 : 0] select;
  output[14 : 0] outputb;
  wire[7 : 0] mux0input,
              mux1input,
              mux2input,
              mux3input,
              mux4input,
              mux5input,
              mux6input,
              mux7input,
              mux8input,
              mux9input,
              mux10input,
              mux11input,
              mux12input,
              mux13input,
              mux14input;
```

```
  assign mux0input = { inputa[0], inputa[34], inputa[19], inputa[20], inputa[10], inputa[2],
inputa[1], inputa[10] };
  assign mux1input = { inputa[1], inputa[35], inputa[0], inputa[21], inputa[11], inputa[3],
inputa[2], inputa[19] };
  assign mux2input = { inputa[2], inputa[0], inputa[20], inputa[22], inputa[12], inputa[4],
inputa[0], inputa[12] };
  assign mux3input = { inputa[3], inputa[1], inputa[1], inputa[23], inputa[13], inputa[5],
inputa[35], inputa[17] };
  assign mux4input = { inputa[4], inputa[2], inputa[21], inputa[24], inputa[14], inputa[6],
inputa[34], inputa[14] };
  assign mux5input = { inputa[5], inputa[3], inputa[2], inputa[25], inputa[15], inputa[7],
inputa[33], inputa[15] };
  assign mux6input = { inputa[6], inputa[4], inputa[22], inputa[26], inputa[16], inputa[8],
inputa[32], inputa[16] };
  assign mux7input = { inputa[7], inputa[5], inputa[3], inputa[27], inputa[17], inputa[9],
inputa[31], inputa[13] };
  assign mux8input = { inputa[8], inputa[6], inputa[23], inputa[28], inputa[18], inputa[10],
inputa[30], inputa[18] };
  assign mux9input = { inputa[9], inputa[7], inputa[4], inputa[29], inputa[19], inputa[11],
inputa[29], inputa[11] };
  assign mux10input = { inputa[10], inputa[8], inputa[24], inputa[30], inputa[20],
inputa[12], inputa[28], inputa[0] };
```

```
  assign mux11input = { inputa[11], inputa[9], inputa[5], inputa[1], inputa[29], inputa[13], inputa[27], inputa[9] };
  assign mux12input = { inputa[12], inputa[10], inputa[25], inputa[32], inputa[30], inputa[14], inputa[26], inputa[2] };
  assign mux13input = { inputa[13], inputa[11], inputa[6], inputa[3], inputa[31], inputa[15], inputa[25], inputa[7] };
  assign mux14input = { inputa[14], inputa[12], inputa[26], inputa[34], inputa[32], inputa[16], inputa[24], inputa[4] };
  mux8 dm0(.select(select),
          .d(mux0input),
          .q(outputb[0]));
  mux8 dm1(.select(select),
          .d(mux1input),
          .q(outputb[1]));
  mux8 dm2(.select(select),
          .d(mux2input),
          .q(outputb[2]));
  mux8 dm3(.select(select),
          .d(mux3input),
          .q(outputb[3]));
  mux8 dm4(.select(select),
          .d(mux4input),
          .q(outputb[4]));
```

```
mux8 dm5(.select(select),
        .d(mux5input),
        .q(outputb[5]));
mux8 dm6(.select(select),
        .d(mux6input),
        .q(outputb[6]));
mux8 dm7(.select(select),
        .d(mux7input),
        .q(outputb[7]));
mux8 dm8(.select(select),
        .d(mux8input),
        .q(outputb[8]));
mux8 dm9(.select(select),
        .d(mux9input),
        .q(outputb[9]));
mux8 dm10(.select(select),
         .d(mux10input),
         .q(outputb[10]));
mux8 dm11(.select(select),
         .d(mux11input),
         .q(outputb[11]));
mux8 dm12(.select(select),
         .d(mux12input),
```

```
        .q(outputb[12]));
    mux8 dm13(.select(select),
        .d(mux13input),
        .q(outputb[13]));
    mux8 dm14(.select(select),
        .d(mux14input),
        .q(outputb[14]));
endmodule
```

k. Partial Hamming Decoder

```
`timescale 1ns/1ns
module HammingDecoder15_11_Partial(DataIno,
                                   DataOut,
                                   ErrDetection);
  input[14 : 0] DataIno;
  output[10 : 0] DataOut;
  wire[14 : 0] DataIno;
  wire[3 : 0] ECC;
  reg[10 : 0] DataOut;
  output reg ErrDetection;
  //internal reg
  reg[3 : 0] syndrome;
  //assign DataIn[10:0] = DataIno[10:0];
  assign ECC[3 : 0] = DataIno[14 : 11];
  always
    @(DataIno or
      ECC)
    begin
      syndrome[0] <= ECC[0] ^ DataIno[0] ^ DataIno[3] ^ DataIno[4] ^ DataIno[6] ^
DataIno[8] ^ DataIno[9] ^ DataIno[10];
      syndrome[1] <= ECC[1] ^ DataIno[0] ^ DataIno[1] ^ DataIno[3] ^ DataIno[5] ^
DataIno[6] ^ DataIno[7] ^ DataIno[8];
```

```
    syndrome[2] <= ECC[2] ^ DataIno[1] ^ DataIno[2] ^ DataIno[4] ^ DataIno[6] ^
DataIno[7] ^ DataIno[8] ^ DataIno[9];
    syndrome[3] <= ECC[3] ^ DataIno[2] ^ DataIno[3] ^ DataIno[5] ^ DataIno[7] ^
DataIno[8] ^ DataIno[9] ^ DataIno[10];
  end
 always
  @(syndrome or
    DataIno)
  begin
    ErrDetection <= (syndrome[3] | syndrome[2] | syndrome[1] | syndrome[0]);
    DataOut[10] <= DataIno[10] ^ ((syndrome[0]) & (~syndrome[1]) & (~syndrome[2]) &
(syndrome[3])); //1001 syn[0]-->syn[3]
    DataOut[9] <= DataIno[9] ^ ((syndrome[0]) & (~syndrome[1]) & (syndrome[2]) &
(syndrome[3])); //1011
    DataOut[8] <= DataIno[8] ^ ((syndrome[0]) & (syndrome[1]) & (syndrome[2]) &
(syndrome[3])); //1111
    DataOut[7] <= DataIno[7] ^ ((~syndrome[0]) & (syndrome[1]) & (syndrome[2]) &
(syndrome[3])); //0111
    DataOut[6] <= DataIno[6] ^ ((syndrome[0]) & (syndrome[1]) & (syndrome[2]) &
(~syndrome[3])); //1110
    DataOut[5] <= DataIno[5] ^ ((~syndrome[0]) & (syndrome[1]) & (~syndrome[2]) &
(syndrome[3])); //0101
```

```
        DataOut[4] <= DataIno[4] ^ ((syndrome[0]) & (~syndrome[1]) & (syndrome[2]) &
    (~syndrome[3])); //1010
        DataOut[3] <= DataIno[3] ^ ((syndrome[0]) & (syndrome[1]) & (~syndrome[2]) &
    (syndrome[3])); //1101
        DataOut[2] <= DataIno[2] ^ ((~syndrome[0]) & (~syndrome[1]) & (syndrome[2]) &
    (syndrome[3])); //0011
        DataOut[1] <= DataIno[1] ^ ((~syndrome[0]) & (syndrome[1]) & (syndrome[2]) &
    (~syndrome[3])); //0110
        DataOut[0] <= DataIno[0] ^ ((syndrome[0]) & (syndrome[1]) & (~syndrome[2]) &
    (~syndrome[3])); //1100
      end
endmodule
```

l. Round Robin

```
`timescale 1ns/1ns //edited by Jonathan to remove Branch for FPGA
module RoundRobin(Req_V,
                  RRReg_in,
                  TableTag_in,
                  //output
                  RRReg_out,
                  TableTag_out,
                  Port_Ad);
 input[4 : 0] Req_V;
 input[4 : 0] RRReg_in;
 input[4 : 0] TableTag_in;
 output[4 : 0] RRReg_out;
 output[4 : 0] TableTag_out;
 output[4 : 0] Port_Ad;
 wire[4 : 0] Req_V;
 wire[4 : 0] RRReg_in;
 wire[4 : 0] TableTag_in;
 reg[4 : 0] RRReg_out;
 reg[4 : 0] TableTag_out;
 reg[4 : 0] Port_Ad;
 //internal signals
```

```
  reg[4 : 0] Req_priority;
  reg[4 : 0] Port_ad_tmp0;
  reg[4 : 0] Port_ad_tmp1;
  reg[4 : 0] Port_ad_tmp2;
  reg[4 : 0] Port_ad_tmp3;
  reg[4 : 0] Port_ad_tmp4;
  always
    @(RRReg_in or
      Req_V)
    begin
      /*East output port*/
      /*round robin prioirty*/
      Port_ad_tmp0[1] <= RRReg_in[0] & Req_V[1];
      Port_ad_tmp0[2] <= !(RRReg_in[0] & Req_V[1]) & Req_V[2];
      Port_ad_tmp0[3] <= !(RRReg_in[0] & Req_V[1]) & (!Req_V[2]) & Req_V[3];
      Port_ad_tmp0[4] <= !(RRReg_in[0] & Req_V[1]) & (!Req_V[2]) & (!Req_V[3]) &
Req_V[4];
      Port_ad_tmp0[0] <= !(RRReg_in[0] & Req_V[1]) & (!Req_V[2]) & (!Req_V[3]) &
(!Req_V[4]) & Req_V[0];
      /*South output port*/
      Port_ad_tmp1[2] <= RRReg_in[1] & Req_V[2];
      Port_ad_tmp1[3] <= !(RRReg_in[1] & Req_V[2]) & Req_V[3];
      Port_ad_tmp1[4] <= !(RRReg_in[1] & Req_V[2]) & (!Req_V[3]) & Req_V[4];
```

```
    Port_ad_tmp1[0] <= !(RRReg_in[1] & Req_V[2]) & (!Req_V[3]) & (!Req_V[4]) &
Req_V[0];
    Port_ad_tmp1[1] <= !(RRReg_in[1] & Req_V[2]) & (!Req_V[3]) & (!Req_V[4]) &
(!Req_V[0]) & Req_V[1];
    /*West output port*/
    Port_ad_tmp2[3] <= RRReg_in[2] & Req_V[3];
    Port_ad_tmp2[4] <= !(RRReg_in[2] & Req_V[3]) & Req_V[4];
    Port_ad_tmp2[0] <= !(RRReg_in[2] & Req_V[3]) & (!Req_V[4]) & Req_V[0];
    Port_ad_tmp2[1] <= !(RRReg_in[2] & Req_V[3]) & (!Req_V[4]) & (!Req_V[0]) &
Req_V[1];
    Port_ad_tmp2[2] <= !(RRReg_in[2] & Req_V[3]) & (!Req_V[4]) & (!Req_V[0]) &
(!Req_V[1]) & Req_V[2];
    /*North output port*/
    Port_ad_tmp3[4] <= RRReg_in[3] & Req_V[4];
    Port_ad_tmp3[0] <= !(RRReg_in[3] & Req_V[4]) & Req_V[0];
    Port_ad_tmp3[1] <= !(RRReg_in[3] & Req_V[4]) & (!Req_V[0]) & Req_V[1];
    Port_ad_tmp3[2] <= !(RRReg_in[3] & Req_V[4]) & (!Req_V[0]) & (!Req_V[1]) &
Req_V[2];
    Port_ad_tmp3[3] <= !(RRReg_in[3] & Req_V[4]) & (!Req_V[0]) & (!Req_V[1]) &
(!Req_V[2]) & Req_V[3];
    /*Local output port*/
    Port_ad_tmp4[0] <= RRReg_in[4] & Req_V[0];
    Port_ad_tmp4[1] <= !(RRReg_in[4] & Req_V[0]) & Req_V[1];
```

```
    Port_ad_tmp4[2] <= !(RRReg_in[4] & Req_V[0]) & (!Req_V[1]) & Req_V[2];
    Port_ad_tmp4[3] <= !(RRReg_in[4] & Req_V[0]) & (!Req_V[1]) & (!Req_V[2]) &
Req_V[3];
    Port_ad_tmp4[4] <= !(RRReg_in[4] & Req_V[0]) & (!Req_V[1]) & (!Req_V[2]) &
(!Req_V[3]) & Req_V[4];
  end
 always
  @(Req_V)
  /*priority: Local, East, South, West, North. The reason to make local input port the
highest priority is to inject flits to network as soon as possible. Alternatively, we can put the
request from the local input port the lowest priority to prevent increasing network
congestion*/
  begin
    Req_priority[4] = Req_V[4];
    Req_priority[0] = !Req_V[4] & Req_V[0];
    Req_priority[1] = !Req_V[4] & !Req_V[0] & Req_V[1];
    Req_priority[2] = !Req_V[4] & !Req_V[0] & !Req_V[1] & Req_V[2];
    Req_priority[3] = !Req_V[4] & !Req_V[0] & !Req_V[1] & !Req_V[2] & Req_V[3];
  end
 always
  @(RRReg_in or
    TableTag_in or
    Port_ad_tmp0 or
```

```
    Port_ad_tmp1 or
    Port_ad_tmp2 or
    Port_ad_tmp3 or
    Port_ad_tmp4 or
    Req_priority or
    Req_V)
  begin
    case (RRReg_in) //the output port is only occupied  by input port at one time, so all possible non-zero RRReg_in  listed are five.
      5'b00001:
        begin
          Port_Ad <= Port_ad_tmp0;
          RRReg_out <= Port_ad_tmp0;
          TableTag_out <= Port_ad_tmp0;
        end
      5'b00010:
        begin
          Port_Ad <= Port_ad_tmp1;
          RRReg_out <= Port_ad_tmp1;
          TableTag_out <= Port_ad_tmp1;
        end
      5'b00100:
        begin
```

```
    Port_Ad <= Port_ad_tmp2;
    RRReg_out <= Port_ad_tmp2;
    TableTag_out <= Port_ad_tmp2;
   end
 5'b01000:
   begin
    Port_Ad <= Port_ad_tmp3;
    RRReg_out <= Port_ad_tmp3;
    TableTag_out <= Port_ad_tmp3;
   end
 5'b10000:
   begin
    Port_Ad <= Port_ad_tmp4;
    RRReg_out <= Port_ad_tmp4;
    TableTag_out <= Port_ad_tmp4;
   end
 default:
   begin
    case (Req_priority)
      5'b00001:
        begin
          Port_Ad <= Port_ad_tmp0;
          RRReg_out <= Port_ad_tmp0;
```

```
    TableTag_out <= Port_ad_tmp0;
  end
5'b00010:
  begin
    Port_Ad <= Port_ad_tmp1;
    RRReg_out <= Port_ad_tmp1;
    TableTag_out <= Port_ad_tmp1;
  end
5'b00100:
  begin
    Port_Ad <= Port_ad_tmp2;
    RRReg_out <= Port_ad_tmp2;
    TableTag_out <= Port_ad_tmp2;
  end
5'b01000:
  begin
    Port_Ad <= Port_ad_tmp3;
    RRReg_out <= Port_ad_tmp3;
    TableTag_out <= Port_ad_tmp3;
  end
5'b10000:
  begin
    Port_Ad <= Port_ad_tmp4;
```

```
                RRReg_out <= Port_ad_tmp4;
                TableTag_out <= Port_ad_tmp4;
              end
            default:
              begin
                Port_Ad <= 5'b0;
                RRReg_out <= RRReg_in;
                TableTag_out <= TableTag_in;
              end
          endcase
        end //the intended output port is not occupied by any packet yet
    endcase
  end
endmodule
```

m. Circular Output FIFO

```
`timescale 1ns/1ns
//Use FPGA InputChannel to create a circular FIFO
module CircularFIFO4(clk,
                     reset,
                     valid_in,
                     datain,
                     NACK,
                     almost_full,
                     dataout,
                     wrreq,
                     select);
  parameter FLITWIDTH = 32;
  parameter CodedFLITWIDTH = 36;
  input[2 : 0] select;
  input clk;
  input reset;
  input valid_in;
  input[CodedFLITWIDTH - 1 : 0] datain;
  input NACK;
  input almost_full;
  output[CodedFLITWIDTH - 1 : 0] dataout;
  output wrreq;
```

```
wire clk;

wire reset;

wire valid_in;

wire[CodedFLITWIDTH - 1 : 0] datain;

wire NACK;

wire almost_full;

reg[CodedFLITWIDTH - 1 : 0] dataout;

reg wrreq;

//added to fix problem with wrreq

wire[10 : 0] Info;

wire[14 : 0] fifo_output_s1_enc;

wire ErrDetection;

//reg[FLITWIDTH-1:0]  dataout_uncoded;

//intermediate signals

wire[CodedFLITWIDTH - 1 : 0] fifo_input_s1;

wire[CodedFLITWIDTH - 1 : 0] fifo_input_s2;

wire[CodedFLITWIDTH - 1 : 0] fifo_input_s3;

wire[CodedFLITWIDTH - 1 : 0] fifo_input_s4;

wire[CodedFLITWIDTH - 1 : 0] fifo_output_s1;

wire[CodedFLITWIDTH - 1 : 0] fifo_output_s2;

wire[CodedFLITWIDTH - 1 : 0] fifo_output_s3;

wire[CodedFLITWIDTH - 1 : 0] fifo_output_s4;

wire[CodedFLITWIDTH - 1 : 0] fifo_input_s5;
```

```
  wire[CodedFLITWIDTH - 1 : 0] fifo_input_s6;
  wire[CodedFLITWIDTH - 1 : 0] fifo_input_s7;
  wire[CodedFLITWIDTH - 1 : 0] fifo_input_s8;
  wire[CodedFLITWIDTH - 1 : 0] fifo_output_s5;
  wire[CodedFLITWIDTH - 1 : 0] fifo_output_s6;
  wire[CodedFLITWIDTH - 1 : 0] fifo_output_s7;
  wire[CodedFLITWIDTH - 1 : 0] fifo_output_s8;
  //assign fifo_input_s1 = almost_full?(fifo_output_s1&ErrDetection):(NACK?fifo_output_s2:datain); detection method, changed 10/7/2015 for correction
  assign fifo_input_s1 = almost_full ? fifo_output_s1 : (NACK ? fifo_output_s2 : datain);
  assign fifo_input_s2 = almost_full ? fifo_output_s2 : fifo_output_s3;
  assign fifo_input_s3 = almost_full ? fifo_output_s3 : fifo_output_s4; //if insert more stages, follow this rule. Just make the first and last stages different
  assign fifo_input_s4 = almost_full ? fifo_output_s4 : fifo_output_s5;
  assign fifo_input_s5 = almost_full ? fifo_output_s5 : fifo_output_s6;
  assign fifo_input_s6 = almost_full ? fifo_output_s6 : fifo_output_s7;
  assign fifo_input_s7 = almost_full ? fifo_output_s7 : fifo_output_s8;
  assign fifo_input_s8 = almost_full ? fifo_output_s8 : fifo_output_s1;
  stagereg31_Proposed_EX circular_s1(.Preset_act_low(1'b1),
                                     .Clear_act_low(!reset),
                                     .CLK(clk),
                                     .DataIn(fifo_input_s1),
```

```
                .DataOut(fifo_output_s1));
stagereg31_Proposed_EX circular_s2(.Preset_act_low(1'b1),
                .Clear_act_low(!reset),
                .CLK(clk),
                .DataIn(fifo_input_s2),
                .DataOut(fifo_output_s2));
stagereg31_Proposed_EX circular_s3(.Preset_act_low(1'b1),
                .Clear_act_low(!reset),
                .CLK(clk),
                .DataIn(fifo_input_s3),
                .DataOut(fifo_output_s3));
stagereg31_Proposed_EX circular_s4(.Preset_act_low(1'b1),
                .Clear_act_low(!reset),
                .CLK(clk),
                .DataIn(fifo_input_s4),
                .DataOut(fifo_output_s4));
stagereg31_Proposed_EX circular_s5(.Preset_act_low(1'b1),
                .Clear_act_low(!reset),
                .CLK(clk),
                .DataIn(fifo_input_s5),
                .DataOut(fifo_output_s5));
stagereg31_Proposed_EX circular_s6(.Preset_act_low(1'b1),
                .Clear_act_low(!reset),
```

```
                        .CLK(clk),
                        .DataIn(fifo_input_s6),
                        .DataOut(fifo_output_s6));
stagereg31_Proposed_EX circular_s7(.Preset_act_low(1'b1),
                        .Clear_act_low(!reset),
                        .CLK(clk),
                        .DataIn(fifo_input_s7),
                        .DataOut(fifo_output_s7));
stagereg31_Proposed_EX circular_s8(.Preset_act_low(1'b1),
                        .Clear_act_low(!reset),
                        .CLK(clk),
                        .DataIn(fifo_input_s8),
                        .DataOut(fifo_output_s8));
always
  @(fifo_output_s1)
  begin
    dataout <= fifo_output_s1;
  end
PFDS p0(.inputa(fifo_output_s1[CodedFLITWIDTH - 1 : 0]),
       .select(select[2 : 0]),
       .outputb(fifo_output_s1_enc[14 : 0]));
HammingDecoder15_11_Partial D0(.DataIno(fifo_output_s1_enc[14 : 0]),
                        .DataOut(Info[10 : 0]),
```

```
                    .ErrDetection(ErrDetection));


        always
          @(reset or
            Info[0] or
            Info[1] or
            almost_full) //or ErrDetection) 10/7/2015
          begin
            if (reset) //| ErrDetection)//or ErrDetection
              wrreq <= 1'b0;
            else
              if ((Info[0] | Info[1]) & (~almost_full)) //changed from fifo_output_s1 since
        scrambling etc. added almost full fix //added almost_full to fix stuck data on links
                wrreq <= 1'b1;
              else
                wrreq <= 1'b0;
          end
endmodule
```

n. Stagereg31

```
`timescale 1ns/1ns
// Copyright (C) 1991-2009 Altera Corporation
// Your use of Altera Corporation's design tools, logic functions
// and other software and tools, and its AMPP partner logic
// functions, and any output files from any of the foregoing
// (including device programming or simulation files), and any
// associated documentation or information are expressly subject
// to the terms and conditions of the Altera Program License
// Subscription Agreement, Altera MegaCore Function License
// Agreement, or other applicable license agreement, including,
// without limitation, that your use is for the sole purpose of
// programming logic devices manufactured by Altera and sold by
// Altera or its authorized distributors.  Please refer to the
// applicable agreement for further details.
// PROGRAM		"Quartus II"
// VERSION		"Version 9.1 Build 222 10/21/2009 SJ Web Edition"
// CREATED		"Fri Jan 15 13:55:43 2010"
module stagereg31_Proposed_EX(Preset_act_low,
                              Clear_act_low,
                              CLK,
                              DataIn,
                              DataOut);
```

```
input Preset_act_low;
input Clear_act_low;
input CLK;
input[35 : 0] DataIn;
output[35 : 0] DataOut;
reg[35 : 0] DataOut_ALTERA_SYNTHESIZED;
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[23] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[23] = 1;
        end
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[23] = DataIn[23];
```

```
        end
    end
  always
    @(posedge CLK or
      negedge Clear_act_low or
      negedge Preset_act_low)
    begin
      if (!Clear_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[22] = 0;
        end
      else
        if (!Preset_act_low)
          begin
            DataOut_ALTERA_SYNTHESIZED[22] = 1;
          end
        else
          begin
            DataOut_ALTERA_SYNTHESIZED[22] = DataIn[22];
          end
    end
  always
    @(posedge CLK or
```

```
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[21] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[21] = 1;
        end
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[21] = DataIn[21];
        end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
```

```
      begin
        DataOut_ALTERA_SYNTHESIZED[20] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[20] = 1;
        end
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[20] = DataIn[20];
        end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[15] = 0;
      end
    else
```

```
    if (!Preset_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[15] = 1;
      end
    else
      begin
        DataOut_ALTERA_SYNTHESIZED[15] = DataIn[15];
      end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[14] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[14] = 1;
        end
```

```
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[14] = DataIn[14];
        end
    end
  always
    @(posedge CLK or
      negedge Clear_act_low or
      negedge Preset_act_low)
    begin
      if (!Clear_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[13] = 0;
        end
      else
        if (!Preset_act_low)
          begin
            DataOut_ALTERA_SYNTHESIZED[13] = 1;
          end
        else
          begin
            DataOut_ALTERA_SYNTHESIZED[13] = DataIn[13];
          end
```

```
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[12] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[12] = 1;
        end
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[12] = DataIn[12];
        end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
```

```
  negedge Preset_act_low)
 begin
  if (!Clear_act_low)
   begin
    DataOut_ALTERA_SYNTHESIZED[7] = 0;
   end
  else
   if (!Preset_act_low)
    begin
     DataOut_ALTERA_SYNTHESIZED[7] = 1;
    end
   else
    begin
     DataOut_ALTERA_SYNTHESIZED[7] = DataIn[7];
    end
 end
always
 @(posedge CLK or
  negedge Clear_act_low or
  negedge Preset_act_low)
 begin
  if (!Clear_act_low)
   begin
```

```
      DataOut_ALTERA_SYNTHESIZED[6] = 0;
    end
  else
    if (!Preset_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[6] = 1;
      end
    else
      begin
        DataOut_ALTERA_SYNTHESIZED[6] = DataIn[6];
      end
end
always
 @(posedge CLK or
  negedge Clear_act_low or
  negedge Preset_act_low)
 begin
  if (!Clear_act_low)
    begin
      DataOut_ALTERA_SYNTHESIZED[5] = 0;
    end
  else
    if (!Preset_act_low)
```

```
      begin
        DataOut_ALTERA_SYNTHESIZED[5] = 1;
      end
    else
      begin
        DataOut_ALTERA_SYNTHESIZED[5] = DataIn[5];
      end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[4] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[4] = 1;
        end
      else
```

```
      begin
        DataOut_ALTERA_SYNTHESIZED[4] = DataIn[4];
      end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[19] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[19] = 1;
        end
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[19] = DataIn[19];
        end
  end
```

```
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[18] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[18] = 1;
        end
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[18] = DataIn[18];
        end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
```

```
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[17] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[17] = 1;
        end
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[17] = DataIn[17];
        end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[16] = 0;
```

```
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[16] = 1;
        end
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[16] = DataIn[16];
        end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[11] = 0;
      end
    else
      if (!Preset_act_low)
        begin
```

```
        DataOut_ALTERA_SYNTHESIZED[11] = 1;
      end
    else
      begin
        DataOut_ALTERA_SYNTHESIZED[11] = DataIn[11];
      end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[10] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[10] = 1;
        end
      else
        begin
```

```
        DataOut_ALTERA_SYNTHESIZED[10] = DataIn[10];
      end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[9] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[9] = 1;
        end
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[9] = DataIn[9];
        end
  end
always
```

```
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[8] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[8] = 1;
        end
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[8] = DataIn[8];
        end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
```

```
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[3] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[3] = 1;
        end
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[3] = DataIn[3];
        end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[2] = 0;
      end
```

```
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[2] = 1;
        end
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[2] = DataIn[2];
        end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[1] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[1] = 1;
```

```
      end
    else
      begin
        DataOut_ALTERA_SYNTHESIZED[1] = DataIn[1];
      end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[0] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[0] = 1;
        end
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[0] = DataIn[0];
```

```
      end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[24] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[24] = 1;
        end
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[24] = DataIn[24];
        end
  end
always
  @(posedge CLK or
```

```
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[31] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[31] = 1;
        end
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[31] = DataIn[31];
        end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
```

```
      begin
        DataOut_ALTERA_SYNTHESIZED[30] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[30] = 1;
        end
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[30] = DataIn[30];
        end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[29] = 0;
      end
    else
```

```
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[29] = 1;
        end
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[29] = DataIn[29];
        end
    end
  always
    @(posedge CLK or
      negedge Clear_act_low or
      negedge Preset_act_low)
    begin
      if (!Clear_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[28] = 0;
        end
      else
        if (!Preset_act_low)
          begin
            DataOut_ALTERA_SYNTHESIZED[28] = 1;
          end
```

```
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[28] = DataIn[28];
        end
    end
  always
    @(posedge CLK or
      negedge Clear_act_low or
      negedge Preset_act_low)
    begin
      if (!Clear_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[27] = 0;
        end
      else
        if (!Preset_act_low)
          begin
            DataOut_ALTERA_SYNTHESIZED[27] = 1;
          end
        else
          begin
            DataOut_ALTERA_SYNTHESIZED[27] = DataIn[27];
          end
```

```
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[26] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[26] = 1;
        end
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[26] = DataIn[26];
        end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
```

```
  negedge Preset_act_low)
 begin
  if (!Clear_act_low)
   begin
    DataOut_ALTERA_SYNTHESIZED[25] = 0;
   end
  else
   if (!Preset_act_low)
    begin
     DataOut_ALTERA_SYNTHESIZED[25] = 1;
    end
   else
    begin
     DataOut_ALTERA_SYNTHESIZED[25] = DataIn[25];
    end
 end
always
 @(posedge CLK or
  negedge Clear_act_low or
  negedge Preset_act_low)
 begin
  if (!Clear_act_low)
   begin
```

```
      DataOut_ALTERA_SYNTHESIZED[35] = 0;
    end
  else
    if (!Preset_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[35] = 1;
      end
    else
      begin
        DataOut_ALTERA_SYNTHESIZED[35] = DataIn[35];
      end
end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[34] = 0;
      end
    else
      if (!Preset_act_low)
```

```
      begin
        DataOut_ALTERA_SYNTHESIZED[34] = 1;
      end
    else
      begin
        DataOut_ALTERA_SYNTHESIZED[34] = DataIn[34];
      end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[33] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[33] = 1;
        end
      else
```

```
      begin
        DataOut_ALTERA_SYNTHESIZED[33] = DataIn[33];
      end
  end
always
  @(posedge CLK or
    negedge Clear_act_low or
    negedge Preset_act_low)
  begin
    if (!Clear_act_low)
      begin
        DataOut_ALTERA_SYNTHESIZED[32] = 0;
      end
    else
      if (!Preset_act_low)
        begin
          DataOut_ALTERA_SYNTHESIZED[32] = 1;
        end
      else
        begin
          DataOut_ALTERA_SYNTHESIZED[32] = DataIn[32];
        end
  end
```

```
        assign DataOut = DataOut_ALTERA_SYNTHESIZED;

endmodule
```

o. Permutation

```
`timescale 1ns/1ns
module Perm32plus4(
        //inputs
        inputa,
        select,
        //outputs
        outputb);
 input[35 : 0] inputa;
 wire[7 : 0] mux0input,
        mux1input,
        mux2input,
        mux3input,
        mux4input,
        mux5input,
        mux6input,
        mux7input,
        mux8input,
        mux9input,
        mux10input,
        mux11input,
        mux12input,
        mux13input,
```

```
        mux14input,
        mux15input,
        mux16input,
        mux17input,
        mux18input,
        mux19input,
        mux20input,
        mux21input,
        mux22input,
        mux23input,
        mux24input,
        mux25input,
        mux26input,
        mux27input,
        mux28input,
        mux29input,
        mux30input,
        mux31input,
        mux32input,
        mux33input,
        mux34input,
        mux35input;
input[2 : 0] select;
```

```
  wire[2 : 0] select;
  reg[35 : 0] q;
  output[35 : 0] outputb;
  wire[35 : 0] inputa;
  //reg[38:0] outputb;
  assign mux0input = { inputa[0], inputa[2], inputa[1], inputa[20], inputa[20], inputa[34],
inputa[2], inputa[10] };
  assign mux1input = { inputa[1], inputa[3], inputa[3], inputa[11], inputa[21], inputa[35],
inputa[0], inputa[19] };
  assign mux2input = { inputa[2], inputa[4], inputa[5], inputa[22], inputa[22], inputa[0],
inputa[1], inputa[12] };
  assign mux3input = { inputa[3], inputa[5], inputa[7], inputa[13], inputa[23], inputa[1],
inputa[35], inputa[17] };
  assign mux4input = { inputa[4], inputa[6], inputa[9], inputa[24], inputa[24], inputa[2],
inputa[34], inputa[14] };
  assign mux5input = { inputa[5], inputa[7], inputa[11], inputa[15], inputa[25], inputa[3],
inputa[33], inputa[15] };
  assign mux6input = { inputa[6], inputa[8], inputa[13], inputa[26], inputa[26], inputa[4],
inputa[32], inputa[16] };
  assign mux7input = { inputa[7], inputa[9], inputa[15], inputa[17], inputa[27], inputa[5],
inputa[31], inputa[13] };
  assign mux8input = { inputa[8], inputa[10], inputa[17], inputa[28], inputa[28], inputa[6],
inputa[30], inputa[18] };
```

```
  assign mux9input = { inputa[9], inputa[11], inputa[19], inputa[19], inputa[29], inputa[7], inputa[29], inputa[11] };
  assign mux10input = { inputa[10], inputa[12], inputa[21], inputa[30], inputa[0], inputa[8], inputa[28], inputa[0] };
  assign mux11input = { inputa[11], inputa[13], inputa[23], inputa[31], inputa[1], inputa[9], inputa[27], inputa[9] };
  assign mux12input = { inputa[12], inputa[14], inputa[25], inputa[32], inputa[2], inputa[10], inputa[26], inputa[2] };
  assign mux13input = { inputa[13], inputa[15], inputa[27], inputa[33], inputa[3], inputa[11], inputa[25], inputa[7] };
  assign mux14input = { inputa[14], inputa[16], inputa[29], inputa[34], inputa[4], inputa[12], inputa[24], inputa[4] };
  assign mux15input = { inputa[15], inputa[17], inputa[31], inputa[35], inputa[5], inputa[13], inputa[23], inputa[5] };
  assign mux16input = { inputa[16], inputa[18], inputa[33], inputa[29], inputa[6], inputa[14], inputa[22], inputa[6] };
  assign mux17input = { inputa[17], inputa[19], inputa[35], inputa[27], inputa[7], inputa[15], inputa[21], inputa[3] };
  assign mux18input = { inputa[18], inputa[20], inputa[34], inputa[16], inputa[8], inputa[16], inputa[20], inputa[8] };
  assign mux19input = { inputa[19], inputa[21], inputa[0], inputa[18], inputa[9], inputa[17], inputa[19], inputa[1] };
```

```
  assign mux20input = { inputa[20], inputa[22], inputa[2], inputa[0], inputa[10], inputa[18],
inputa[18], inputa[30] };
  assign mux21input = { inputa[21], inputa[23], inputa[4], inputa[1], inputa[31], inputa[19],
inputa[17], inputa[29] };
  assign mux22input = { inputa[22], inputa[24], inputa[6], inputa[2], inputa[32], inputa[20],
inputa[16], inputa[35] };
  assign mux23input = { inputa[23], inputa[25], inputa[8], inputa[3], inputa[33], inputa[21],
inputa[15], inputa[33] };
  assign mux24input = { inputa[24], inputa[26], inputa[10], inputa[4], inputa[34],
inputa[22], inputa[14], inputa[34] };
  assign mux25input = { inputa[25], inputa[27], inputa[12], inputa[5], inputa[35],
inputa[23], inputa[13], inputa[24] };
  assign mux26input = { inputa[26], inputa[28], inputa[14], inputa[6], inputa[18],
inputa[24], inputa[12], inputa[27] };
  assign mux27input = { inputa[27], inputa[29], inputa[16], inputa[7], inputa[19],
inputa[25], inputa[11], inputa[22] };
  assign mux28input = { inputa[28], inputa[30], inputa[18], inputa[8], inputa[30],
inputa[26], inputa[10], inputa[32] };
  assign mux29input = { inputa[29], inputa[31], inputa[20], inputa[9], inputa[11],
inputa[27], inputa[9], inputa[31] };
  assign mux30input = { inputa[30], inputa[32], inputa[22], inputa[10], inputa[12],
inputa[28], inputa[8], inputa[20] };
```

```
 assign mux31input = { inputa[31], inputa[33], inputa[24], inputa[21], inputa[13], inputa[29], inputa[7], inputa[21] };
 assign mux32input = { inputa[32], inputa[34], inputa[26], inputa[12], inputa[14], inputa[30], inputa[6], inputa[28] };
 assign mux33input = { inputa[33], inputa[35], inputa[28], inputa[23], inputa[15], inputa[31], inputa[5], inputa[23] };
 assign mux34input = { inputa[34], inputa[0], inputa[30], inputa[14], inputa[16], inputa[32], inputa[4], inputa[26] };
 assign mux35input = { inputa[35], inputa[1], inputa[32], inputa[25], inputa[17], inputa[33], inputa[3], inputa[25] };
  mux8 m0(.select(select),
        .d(mux0input),
        .q(outputb[0]));
  mux8 m1(.select(select),
        .d(mux1input),
        .q(outputb[1]));
  mux8 m2(.select(select),
        .d(mux2input),
        .q(outputb[2]));
  mux8 m3(.select(select),
        .d(mux3input),
        .q(outputb[3]));
  mux8 m4(.select(select),
```

```
        .d(mux4input),
        .q(outputb[4]));
mux8 m5(.select(select),
        .d(mux5input),
        .q(outputb[5]));
mux8 m6(.select(select),
        .d(mux6input),
        .q(outputb[6]));
mux8 m7(.select(select),
        .d(mux7input),
        .q(outputb[7]));
mux8 m8(.select(select),
        .d(mux8input),
        .q(outputb[8]));
mux8 m9(.select(select),
        .d(mux9input),
        .q(outputb[9]));
mux8 m10(.select(select),
         .d(mux10input),
         .q(outputb[10]));
mux8 m11(.select(select),
         .d(mux11input),
         .q(outputb[11]));
```

```
mux8 m12(.select(select),
        .d(mux12input),
        .q(outputb[12]));
mux8 m13(.select(select),
        .d(mux13input),
        .q(outputb[13]));
mux8 m14(.select(select),
        .d(mux14input),
        .q(outputb[14]));
mux8 m15(.select(select),
        .d(mux15input),
        .q(outputb[15]));
mux8 m16(.select(select),
        .d(mux16input),
        .q(outputb[16]));
mux8 m17(.select(select),
        .d(mux17input),
        .q(outputb[17]));
mux8 m18(.select(select),
        .d(mux18input),
        .q(outputb[18]));
mux8 m19(.select(select),
        .d(mux19input),
```

```
        .q(outputb[19]));
mux8 m20(.select(select),
        .d(mux20input),
        .q(outputb[20]));
mux8 m21(.select(select),
        .d(mux21input),
        .q(outputb[21]));
mux8 m22(.select(select),
        .d(mux22input),
        .q(outputb[22]));
mux8 m23(.select(select),
        .d(mux23input),
        .q(outputb[23]));
mux8 m24(.select(select),
        .d(mux24input),
        .q(outputb[24]));
mux8 m25(.select(select),
        .d(mux25input),
        .q(outputb[25]));
mux8 m26(.select(select),
        .d(mux26input),
        .q(outputb[26]));
mux8 m27(.select(select),
```

```
        .d(mux27input),
        .q(outputb[27]));
mux8 m28(.select(select),
        .d(mux28input),
        .q(outputb[28]));
mux8 m29(.select(select),
        .d(mux29input),
        .q(outputb[29]));
mux8 m30(.select(select),
        .d(mux30input),
        .q(outputb[30]));
mux8 m31(.select(select),
        .d(mux31input),
        .q(outputb[31]));
mux8 m32(.select(select),
        .d(mux32input),
        .q(outputb[32]));
mux8 m33(.select(select),
        .d(mux33input),
        .q(outputb[33]));
mux8 m34(.select(select),
        .d(mux34input),
        .q(outputb[34]));
```

```
        mux8 m35(.select(select),
                .d(mux35input),
                .q(outputb[35]));
endmodule
```

p. De-Permutation

```
`timescale 1ns/1ns
module DePerm32plus4(
                //inputs
                inputa,
                select,
                //outputs
                outputb);
 input[35 : 0] inputa;
 wire[7 : 0] mux0input,
          mux1input,
          mux2input,
          mux3input,
          mux4input,
          mux5input,
          mux6input,
          mux7input,
          mux8input,
          mux9input,
          mux10input,
          mux11input,
          mux12input,
          mux13input,
```

```
        mux14input,
        mux15input,
        mux16input,
        mux17input,
        mux18input,
        mux19input,
        mux20input,
        mux21input,
        mux22input,
        mux23input,
        mux24input,
        mux25input,
        mux26input,
        mux27input,
        mux28input,
        mux29input,
        mux30input,
        mux31input,
        mux32input,
        mux33input,
        mux34input,
        mux35input;
input[2 : 0] select;
```

```
  wire[2 : 0] select;
  reg[35 : 0] q;
  output[35 : 0] outputb;
  wire[35 : 0] inputa;
  //reg[38:0] outputb;
  assign mux0input = { inputa[0], inputa[34], inputa[19], inputa[20], inputa[10], inputa[2], inputa[1], inputa[10] };
  assign mux1input = { inputa[1], inputa[35], inputa[0], inputa[21], inputa[11], inputa[3], inputa[2], inputa[19] };
  assign mux2input = { inputa[2], inputa[0], inputa[20], inputa[22], inputa[12], inputa[4], inputa[0], inputa[12] };
  assign mux3input = { inputa[3], inputa[1], inputa[1], inputa[23], inputa[13], inputa[5], inputa[35], inputa[17] };
  assign mux4input = { inputa[4], inputa[2], inputa[21], inputa[24], inputa[14], inputa[6], inputa[34], inputa[14] };
  assign mux5input = { inputa[5], inputa[3], inputa[2], inputa[25], inputa[15], inputa[7], inputa[33], inputa[15] };
  assign mux6input = { inputa[6], inputa[4], inputa[22], inputa[26], inputa[16], inputa[8], inputa[32], inputa[16] };
  assign mux7input = { inputa[7], inputa[5], inputa[3], inputa[27], inputa[17], inputa[9], inputa[31], inputa[13] };
  assign mux8input = { inputa[8], inputa[6], inputa[23], inputa[28], inputa[18], inputa[10], inputa[30], inputa[18] };
```

```
  assign mux9input = { inputa[9], inputa[7], inputa[4], inputa[29], inputa[19], inputa[11], inputa[29], inputa[11] };
  assign mux10input = { inputa[10], inputa[8], inputa[24], inputa[30], inputa[20], inputa[12], inputa[28], inputa[0] };
  assign mux11input = { inputa[11], inputa[9], inputa[5], inputa[1], inputa[29], inputa[13], inputa[27], inputa[9] };
  assign mux12input = { inputa[12], inputa[10], inputa[25], inputa[32], inputa[30], inputa[14], inputa[26], inputa[2] };
  assign mux13input = { inputa[13], inputa[11], inputa[6], inputa[3], inputa[31], inputa[15], inputa[25], inputa[7] };
  assign mux14input = { inputa[14], inputa[12], inputa[26], inputa[34], inputa[32], inputa[16], inputa[24], inputa[4] };
  assign mux15input = { inputa[15], inputa[13], inputa[7], inputa[5], inputa[33], inputa[17], inputa[23], inputa[5] };
  assign mux16input = { inputa[16], inputa[14], inputa[27], inputa[18], inputa[34], inputa[18], inputa[22], inputa[6] };
  assign mux17input = { inputa[17], inputa[15], inputa[8], inputa[7], inputa[35], inputa[19], inputa[21], inputa[3] };
  assign mux18input = { inputa[18], inputa[16], inputa[28], inputa[19], inputa[26], inputa[20], inputa[20], inputa[8] };
  assign mux19input = { inputa[19], inputa[17], inputa[9], inputa[9], inputa[27], inputa[21], inputa[19], inputa[1] };
```

```
  assign mux20input = { inputa[20], inputa[18], inputa[29], inputa[0], inputa[0], inputa[22], inputa[18], inputa[30] };
  assign mux21input = { inputa[21], inputa[19], inputa[10], inputa[31], inputa[1], inputa[23], inputa[17], inputa[31] };
  assign mux22input = { inputa[22], inputa[20], inputa[30], inputa[2], inputa[2], inputa[24], inputa[16], inputa[27] };
  assign mux23input = { inputa[23], inputa[21], inputa[11], inputa[33], inputa[3], inputa[25], inputa[15], inputa[33] };
  assign mux24input = { inputa[24], inputa[22], inputa[31], inputa[4], inputa[4], inputa[26], inputa[14], inputa[25] };
  assign mux25input = { inputa[25], inputa[23], inputa[12], inputa[35], inputa[5], inputa[27], inputa[13], inputa[35] };
  assign mux26input = { inputa[26], inputa[24], inputa[32], inputa[6], inputa[6], inputa[28], inputa[12], inputa[34] };
  assign mux27input = { inputa[27], inputa[25], inputa[13], inputa[17], inputa[7], inputa[29], inputa[11], inputa[26] };
  assign mux28input = { inputa[28], inputa[26], inputa[33], inputa[8], inputa[8], inputa[30], inputa[10], inputa[32] };
  assign mux29input = { inputa[29], inputa[27], inputa[14], inputa[16], inputa[9], inputa[31], inputa[9], inputa[21] };
  assign mux30input = { inputa[30], inputa[28], inputa[34], inputa[10], inputa[28], inputa[32], inputa[8], inputa[20] };
```

```
 assign mux31input = { inputa[31], inputa[29], inputa[15], inputa[11], inputa[21], inputa[33], inputa[7], inputa[29] };
 assign mux32input = { inputa[32], inputa[30], inputa[35], inputa[12], inputa[22], inputa[34], inputa[6], inputa[28] };
 assign mux33input = { inputa[33], inputa[31], inputa[16], inputa[13], inputa[23], inputa[35], inputa[5], inputa[23] };
 assign mux34input = { inputa[34], inputa[32], inputa[32], inputa[14], inputa[24], inputa[0], inputa[4], inputa[24] };
 assign mux35input = { inputa[35], inputa[33], inputa[17], inputa[15], inputa[25], inputa[1], inputa[3], inputa[22] };
 mux8 dm0(.select(select),
        .d(mux0input),
        .q(outputb[0]));
 mux8 dm1(.select(select),
        .d(mux1input),
        .q(outputb[1]));
 mux8 dm2(.select(select),
        .d(mux2input),
        .q(outputb[2]));
 mux8 dm3(.select(select),
        .d(mux3input),
        .q(outputb[3]));
 mux8 dm4(.select(select),
```

```
        .d(mux4input),
        .q(outputb[4]));
mux8 dm5(.select(select),
        .d(mux5input),
        .q(outputb[5]));
mux8 dm6(.select(select),
        .d(mux6input),
        .q(outputb[6]));
mux8 dm7(.select(select),
        .d(mux7input),
        .q(outputb[7]));
mux8 dm8(.select(select),
        .d(mux8input),
        .q(outputb[8]));
mux8 dm9(.select(select),
        .d(mux9input),
        .q(outputb[9]));
mux8 dm10(.select(select),
         .d(mux10input),
         .q(outputb[10]));
mux8 dm11(.select(select),
         .d(mux11input),
         .q(outputb[11]));
```

```
mux8 dm12(.select(select),
        .d(mux12input),
        .q(outputb[12]));
mux8 dm13(.select(select),
        .d(mux13input),
        .q(outputb[13]));
mux8 dm14(.select(select),
        .d(mux14input),
        .q(outputb[14]));
mux8 dm15(.select(select),
        .d(mux15input),
        .q(outputb[15]));
mux8 dm16(.select(select),
        .d(mux16input),
        .q(outputb[16]));
mux8 dm17(.select(select),
        .d(mux17input),
        .q(outputb[17]));
mux8 dm18(.select(select),
        .d(mux18input),
        .q(outputb[18]));
mux8 dm19(.select(select),
        .d(mux19input),
```

```
        .q(outputb[19]));
mux8 dm20(.select(select),
        .d(mux20input),
        .q(outputb[20]));
mux8 dm21(.select(select),
        .d(mux21input),
        .q(outputb[21]));
mux8 dm22(.select(select),
        .d(mux22input),
        .q(outputb[22]));
mux8 dm23(.select(select),
        .d(mux23input),
        .q(outputb[23]));
mux8 dm24(.select(select),
        .d(mux24input),
        .q(outputb[24]));
mux8 dm25(.select(select),
        .d(mux25input),
        .q(outputb[25]));
mux8 dm26(.select(select),
        .d(mux26input),
        .q(outputb[26]));
mux8 dm27(.select(select),
```

```
        .d(mux27input),
        .q(outputb[27]));
mux8 dm28(.select(select),
        .d(mux28input),
        .q(outputb[28]));
mux8 dm29(.select(select),
        .d(mux29input),
        .q(outputb[29]));
mux8 dm30(.select(select),
        .d(mux30input),
        .q(outputb[30]));
mux8 dm31(.select(select),
        .d(mux31input),
        .q(outputb[31]));
mux8 dm32(.select(select),
        .d(mux32input),
        .q(outputb[32]));
mux8 dm33(.select(select),
        .d(mux33input),
        .q(outputb[33]));
mux8 dm34(.select(select),
        .d(mux34input),
        .q(outputb[34]));
```

```
        mux8 dm35(.select(select),
                .d(mux35input),
                .q(outputb[35]));
endmodule
```