School of Computer Science and Engineering

Faculty of Engineering

The University of New South Wales

Enhanced Trojan Detection and Localization

by

Meisi Li

Thesis submitted as a requirement for the degree of

Bachelor of Engineering in Computer Engineering

Submitted: August 2020 Student ID: z5119623

Supervisor: Hui Annie Guo

Abstract

The software level security problem of the computer system has been tackled currently. However, due to the continuous increase of network-on-chip (NoC), the design in the hardware system might have different potential problems, especially for the chip from the third-party vendors. The third-party components can be inserted with hardware Trojan without notice. The hardware Trojan may be very small and can escape the security in the circuit-level testing. The hardware Trojan can intercept and modify the passing packets to damage the integrity and reliability of the system.

This thesis aims to detect and locate the hardware Trojan inside the NoC. Once the Trajan is activated, it could modify the data in the packets, change the packet type, and divert the packet to the wrong location. To optimize the consumption and energy of NoC detection, this paper proposes a methodology to against Trojan tampering. This thesis use the progressive packet authentication to detect the Trojan and use an energy-efficient Trojan detection design (EETD), where the system locates the Trojan when detect the tampering.

Unlike traditional authentication detection, progressive packet authentication saves power consumption and cost. Comparing to the normal localization detection, the EETD can achieve efficiency performance. Based on our analysis, we conducted a series of experiments on each design to evaluate the performance of the design. Our experiments show that each of our detection designs can achieve a higher detection rate which proves the effectiveness of our design method.

Acknowledgements

I would like to express my supervisor, Hui Annie Guo, for her insightful suggestions and patient guidance. Without her motivation, this work would have never been complete. In addition, I would like to thank my accessor, Sri Parameswaran, for his valuable comments on this thesis.

Last but not least, special thanks to my parents for their constant support, guidance and encouragement throughout my study.

Abbreviations

AE Authentication and Encryption Design

AU Authentication Unit

CAD Computer Aided Design

DMU Detection Management Unit

EETD Energy Efficient Trojan Detection

EDU End-to-End Trojan Detection Unit

GCM Galois/Counter Mode Operation

IC Integrated Circuit

LU Localization Unit

NI Network Interface

NOC Network-on-Chip

PE Process Element

PU Processing Unit

SoC System-on-Chip

Contents

List of Figures

List of Tables

Chapter 1

Introduction

As multicore computer architectures become more prevalent and the semiconductor technology is globalized, the System-on-Chips (SoCs) are becoming more common in the hardware design. The SoC combines the electronic circuitry required for various computer components onto a single integrated chip (IC). NoC is a network-based communications subsystem on an IC, most commonly between modules in a SoC.

Most NoCs come from the untrusted third-party vendors who may have the hardware Trojans, which can expose the system to various attacks. Encryption can be applied, and authentication is typically used to protect the system from intentional attacks on the packets that are delivered. The use of data tags is a common approach of protecting data integrity, where packet data is appended by a tag at the source node. The tag value is authenticated when the packet data is used. The packet data is typically checked against the tags at the destination node. If the data value is changed, the data is considered to have been tampered with and invalidated.

The authentication of data integrity for ICs have been investigated in the past decade, especially for the data packets transmitted through the network. The Authentication and encryption design (AE) use the Galois/Counter Mode (GCM) operations to avoid the offending to the confidentiality and integrity of memory data [3]. However, if the authentication on the next transmission node does not detect the changed packets, the Trojan will not be detected forever. To make matters worse, each node has a large error-code, which increases the computational burden and reduces the network performance. One study showed that a Trojan can be neutralized by using different pipelines [1].

However, a typical SoC design always uses a single NoC, which means that removing the Trojan is not a simple way to accomplish it. Therefore, a runtime detection technology is the last line of defense in NoC and if it is untrusted, the entire system may crash.

In this paper, we focus on the Trojan in third-party NoC. We assume that the NoC contains a single hardware Trojan. The Trojan can be activated and de-activated on the system. Once it activated, the Trojan starts to attack the passing packets. It is important to use tags for the authentication and it requires a big tag size is necessary. But the bigger of the size is, the more consumption of packet space needs. This will cause the greater reducing of the performance. In addition, if the probability of activation of the Trojan may be pretty low, the unnecessary authentication has a significant effect on the system performance and availability.

In this paper, we aim for a design that is low in overhead, great in performance and effective in verification of the transferring packets over the network.

The main contributions of this paper are:

- Uses a progressive packet authentication of detection units
- A progressive packet authentication to locate the Trojan.

The rest of the paper is organized as follows. Chapter 2 reviews the existing work related to the Trojan detection and localization. Our enhanced detection design approach is presented in Chapter 3. The experiences are introduced in Chapter 4. Chapter 5 describes the result analyzes and evaluation of the enhanced design. Conclusion and future works show the effectiveness and improvement of our approach are given in Chapter 6.

Chapter 2

Background

In this chapter, we aim to give an introduction of background knowledge for the research presented in this thesis. We first state the outline of Network-on-Chip (NoC) concepts in Section 2.1. Section 2.2 underlines the hardware Trojan insert, threats and countermeasures.

2.1 Network-on-Chip

System-on-Chip (SoC) is a chip that implements most or even complete electronic system functions within a single chip. This kind of chip is the core of the high-end electronic systems, and with the development of integrated circuits technology, it is gradually used in the low-end electronic systems. The Network-on-Chip (NoC) is a communication on-chip to solve the problem of data transmission between different cores (the core and un-core hardware units) in the multi-core system. To fully comprehend the NoC, this section first introduces the architecture (section 2.1.1), operations (section 2.1.2) and presents the advantages of NoC.

2.1.1 Architecture

The NoC is a mesh network contained of multiple circuits and routers. A basic structure of the NoC can be seen in Figure 2.1 [4]:

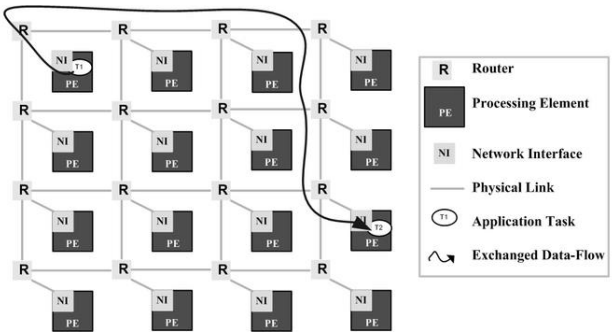

Figure 2.1: 2D mesh network of NoC

- Process Element (PE): is a logical processor core (IP core). The PE generates packets.
- Router: has input ports and output ports corresponding to different port directions. Routers send data to the respective IP core and receive the packet from all IP cores.
- Network Interface (NI): connects to specific IP core. The NI packs the sending data and unpacks the receiving packets by the IP core.
- Physical Link: connects the adjacent routers and links the routers with PEs.

Taking the application task as an example, the NI converts the packets generated from the source IP core into fixed-size flow-control digits (flits). The flits have the major information of the packets. These flights will transfer the packets from the source router to the destination router in a hop-by-hop manner.

2.1.2 Operation

Flow Control

Flow control determines the allocation of resources to packets transferring on a network, such as bandwidth and buffer capacity. It is helpful to avoid buffer overflow and packet loss. The same channel flow control can only assign the channel to one packet and it must consider other packets. Therefore, efficient flow control could bring better network performance.

Buffered flow control techniques can be categorized into two parts: Packet-Buffer flow control and Flit-Buffer flow control [9].

Store and forward flow control, a Packet-Buffer flow control, means that when a node (router) forwards a data packet, it must first store the entire data packet in the buffer,

and then forward it out. Assume a packet of L length is passed through H nodes to the destination node. Its delay is T = (L/BW + R) x H. Among them, L is the packet length, BW is the network bandwidth, R is the delay of each node to find the path, and H is the number of passing nodes.

Wormhole flow control, a kind of Flit-Buffer flow control, splits the packets into several flits and sends them through the network. Supports a packet of length L and reaches to the destination node through H nodes. The delay is T = L/BW + R x H. Among them, L is the packet length, BW is the network bandwidth, R is the delay of each node to find the path, and H is the number of passing nodes.

Routing

Routing is a mechanism to determine the packet transferring from source to destination. According to the changes adaptively with the network traffic or topology, routing algorithms can be divided into static routing algorithms and dynamic routing algorithms. The static routing algorithm of NoC is an active routing algorithm and XY routing algorithm. The source routing algorithm is the route from the source node to the destination node. The XY routing follows the first row moving, then the column moving. The dynamic routing involves dynamic distribution mechanism, based on local link congestion. This algorithm has well adapted to different network status, but the algorithm implementation is complex and expensive.

2.1.3 Advantages

Comparing to the traditional bus structure, the NoC uses data routing and packet switching technology to solve the problems of poor scalability, low communication efficiency and power consumption of the SoC bus structure. The NoC has the following advantages:

- Network bandwidth. The bus structure interconnects multiple IP cores and shares a data bus, which means only one pair of IPs could communication at the same time. With the increase in system scale, the communication efficiency of the bus structure inevitably reduces, and the improvement of system performance is restricted. However, the nodes in NoC can simultaneously use different physical links in the network to transmit packet and support multiple IP cores for communication. Therefore, the NoC has better communication efficiency and higher bandwidth.

When there is a pair of nodes requesting the same physical link, the network will have competition. The NoC nodes use time-division multiplexing physical links to solve this problem and reduce the probability of competition.

- Scalability. The traditional bus structure needs a different design for different system requirements. When the system functions need to expand, the existing design has to redesign, and it would affect the design cycle and cost. However, in the NoC, every node is connected to its local IP core through NI. When the system functions expanding, the newly added processor cores need to connect to the routing nodes in the network through the NI without redesign. As a result, the NoC has good scalability.
- Power consumption. As the continuous increase of the SoC scale, each data transfer on the bus needs to drive the global interconnection line, which significantly increases the power consumption of the bus structure. In the NoC, the power consumption is related to the distance between the communication nodes. The closer of two nodes causes a lower power consumption in communication.
- Signal integrity and signal delay. Since the reduction of the feature size of integrated circuits and the increase of circuit scale, the distance of interconnection lines is reduced and the coupling capacitance between lines increases. In addition, a long global parallel bus brings more serious problems. The integrity of the signal and the correctness of signal transmission would be damaged by the crosstalk noise. The delay on the interconnection line becomes significant in the meantime. These problems make the management of clock offset difficult.

2.2 Hardware Trojan

Once the chip has a potential Trojan, it will leak information and lead to unimaginable disasters. Thus, beginning with the preparation of this project, this section is to fully understand when the Trojans in the chip are inserted and classify the Trojans. Section 2.2.1 gives a concise introduction when are the hardware Trojans inserted. Section 2.2.2 introduces different Trojan type and the corresponding attacks. In Section 2.2.3, it describes some existing countermeasures to detect the hardware Trojans.

2.2.1 Hardware Trojan Insertion

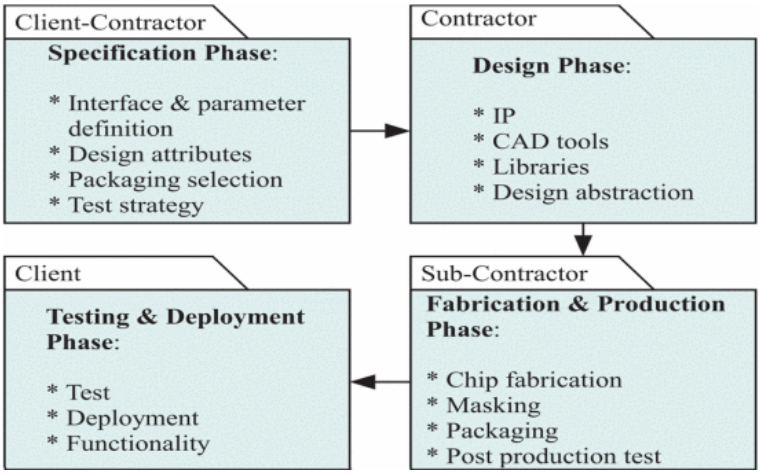

Figure 2.2: Trojan insertion flow

The IC might be modified by an untrusted third-vender party in all steps during the manufacturing process. It is possible to know how to insert the Trojan in the chip production cycle. Figure 2.2 determines four steps in the lift cycle [7]:

Step 1 is to define the requirements and specifications of a chip. The incompatible chips might lead to faults in the verification and revision. In step 2, the manufactory design and produces the target chips. After this, the product is packaged. Step 3 is complex and time-consuming since it has testing and repeating. In this step, computer aided design (CAD) tools define the necessary components (e.g. doping, metallization, and glass area), and then transfer the design to a silicon wafer. After this, this step tests each layer function and repeat the previous works. At last, the wafer is cut into individual integrated circuits (IC) and the ICs are packaged. The last step is testing and deploying the ICs in the clients.

2.2.2 Hardware Trojan Attacks

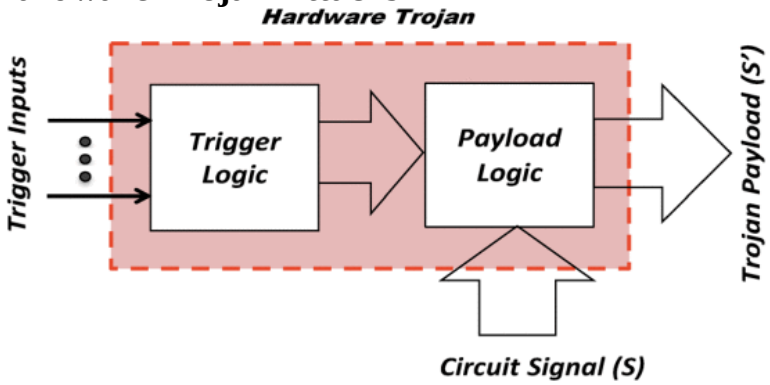

Figure 2.3: Trigger in Trojan

The IC was modified by the untrusted factory during the fabrication, which causes undesired results. The hardware Trojan has different attack in the chip lift cycle [2]. From the figure 2.3, we found that the Trojan could be activated went the trigger is activated. The system may happen unrecoverable failures. The Trojan has two main features:

- evades the testing and development process
- has malicious intentions and attacks

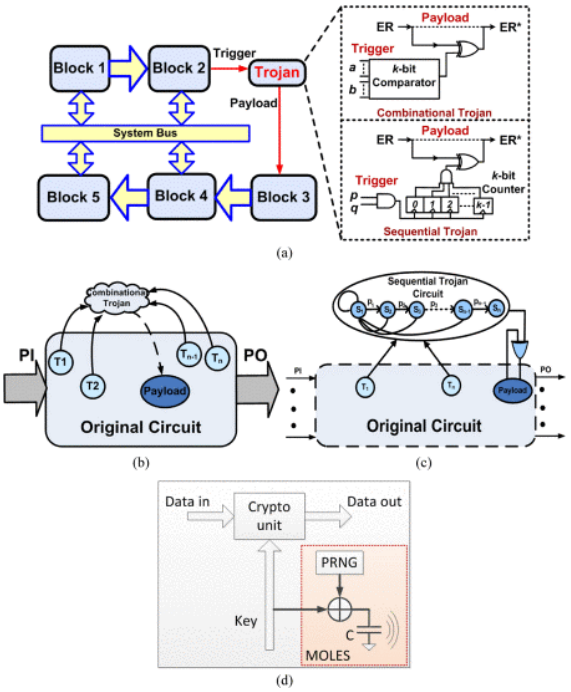

Figure 2.4: Trojan Categories

Different type of Trojan has different kinds of attacks. Figure 2.4 mentions three forms of Trojan attack.

In Figure 2.4(a), a kind of Trojan is the combinational Trojan and the other type is the sequential Trojan. These two types of Trojan have unique characterizes as follows:

- The combinational Trojan: (general model is shown in figure2.4 (b))
    - No state elements, such as flip-flop
    - Controls by the input values, node a and node b
    - The flipping signal ER depends on the input values

- The sequential Trojan: (general model is shown in figure 2.4(c))
    - Has a series of state transistors

- Has k-bit counter and once it reaches to 2k-1, the trojan activates
- ER depends on the counter

Figure 2.4 (d) is the encryption engine Trojans. This type of Trojan uses off-chip leakage technology to leak the hardware keys inside the network and attempt to bypass the security mechanism by side-channels.

2.2.3 Hardware Trojan Countermeasures

Survey on Hardware Trojan Detection Technology

After an understanding of the damages and different types of Trojans, the most important thing is to find a method to detect the Trojan on the NoC. Before starting to think about countermeasures, it should study the existing research on Trojans detections. A survey of hardware trojan detection techniques points out five different types of Trojan detection [8]. They are:

- Optical Inspection Based Detection Techniques
  - reverses engineering to detect Trojans
  - the test picture of the circuit is obtained by removing each layer to destroy the chip
  - compares the layout of the testing circuit with the picture of the manufactured testing circuit
  - is a powerful technology that suitable for detecting hardware Trojans inserting in the manufacturing process
- Testing Based Detection Techniques
  - usually applies to the chip process before shipment.
  - can also be used to detect the presence of hardware Trojans
  - not invasive
  - can identify hardware Trojans applied at different levels of the design process, including malicious modifications applied in the IP core
- Side Channel Based Detection Techniques
  - a powerful technique for detecting malicious modifications in ICs
  - two main limitations:
    - physical characteristics can also be modified by other factors, not just the hardware Trojan
    - it is difficult to extract the correct timing required for a specific path

- Run Time Detection Techniques
    - bypasses the detected Trojan
    - can only detect certain types of Trojans
- Invasive Detection Techniques
    - modifies the IC structure in the design to avoid inserting hardware Trojans
    - modifies the IP and use the voluntary change as a fingerprint to identify the IP after manufacturing
    - reduces overhead and bypass standard detection techniques
    - difficult to implement in complex ICs

Progressive Authentication Detection

After a general understanding of the hardware Trojan and the numerous detection technology, progressive authentication detection is a suitable method in my thesis to detect the Trojan on a NoC. This detection uses progressive authentication for a set of small tag segments to identify data [5].

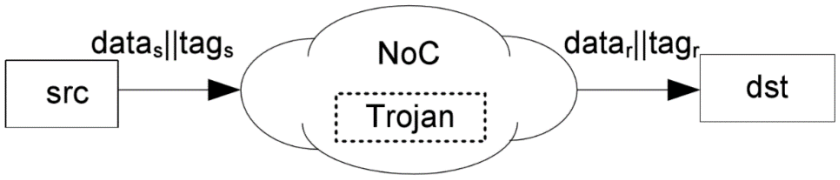

Figure 2.5: Traditional Authentication

Firstly, it is necessary to know about how the traditional authentication works. A general authentication method uses tags to determine the data packet. The source generates the tags and attaches them to the packet. The target compares the data with the tags to determine if the packets have tampered. The figure 2.5 shows the packets transferring. The packets are marked at the source node and the packets with tags pass through the network to the destination node. The data in the target node is modified when the tag is invalid.

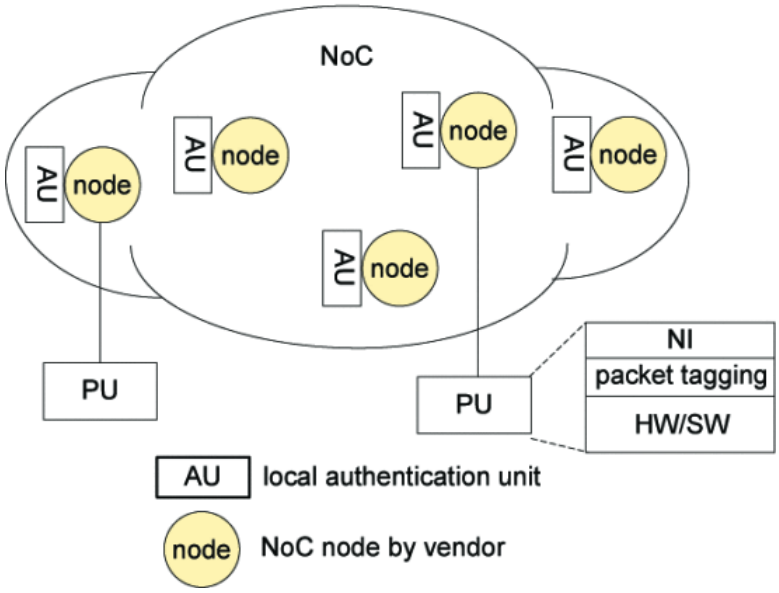

Figure 2.6: Progressive Packet Authentication

A very large size tags for Trojan detection in the authentication would bring a higher bandwidth consumption and limit the performance improvement. Progressive packet data authentication deals with this problem by creating a set of tag segments of each node.

In progressive authentication, the source generates the tag segments for every node and stores them into a tag table. During the transmission, the data packets are identified at the passing nodes. Different node has its specific tag segment in the tag table. The tag value is calculated by the dedicated key of node and each node uses the same key for the local tag generation.

In figure 2.6, the designer's processing unit (PU) decides the transmit data and target node id. Each node connects to a PU and the PU in the source node generates a set of tag segments. For the incoming packet, the secure authentication unit (AU), attached to each node, calculates the local tag and compares it to the corresponding tag segment in the tag table. The packets would be moved to the next node when they are the same. Otherwise, the AU drops the packets.

Efficient Energy Trojan Detection

Progressive authentication is a method to detect the Trojan but it does not determine the Trojan location. Efficient energy Trojan detection (EETD) is a method to locate the hardware Trojan on the NoC. Figure 2.7 is the structure of EETD [6].

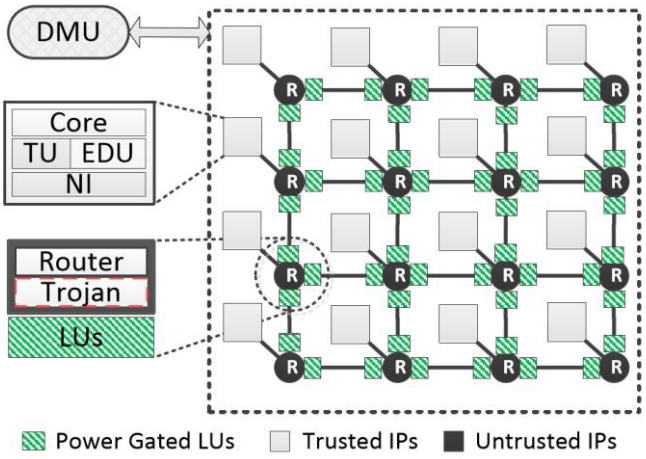

Figure 2.7: Architecture Overview of EETD

From the figure, there are two types of detection units. The first detection unit is end-to-end Trojan detection unit (EDU). The EDUs are activated all the time and connects to the IP cores. The incoming packets are identified by the EDUs. When the tampered packet is detected, the EDU sends the tampered signals to the detection management unit (DMU). The DMU receivers the signal and actives the localization algorithm to locate the Trojan. The second detection unit is localization unit (LU). The LUs are activated by the DMU and after the activation, they are attached to the nodes to find the Trojan location.

input : The node location where the first tampered packet is detected, (x_f, y_f);
NoC design parameters;
Application parameters;
output: Trojan location (x_h, y_h)

1 $T_t = calThresholdTime(x_f)$;
2 $T_s = calRowWaitTime()$;
3 $(x, y) = initWormLoc(x_f, y_f)$;
4 $activateLU(Column(x_f))$;
5 /* --- Step I: Column Movement ---- */
6 **while** *waitTime* $< T_t$ *and TrojanRow is not found* **do**
7 $y = wormMoveAlong(Column(x_f))$;
8 **end while**
9 /* --- Step II: Row Activation ---- */
10 **if** *TrojanRow found* **then**
11 **if** *Detected at right port* **then**
12 $activateLU(RightRowSec(y))$;
13 **else**
14 $activateLU(LeftRowSec(y))$;
15 **end if**
16 **else**
17 $activateLU(Row(y))$;
18 **end if**
19 /* --- Step III: Row Movement ---- */
20 **while** *waitTime* $< T_s$ **do**
21 $x = wormMoveAlong(Row(y))$;
22 **end while**
23 /* --- Step IV: Trojan Location --- */
24 $(x_h, y_h) = (x, y)$;

Figure 2.8: Localization Algorithm

Figure 2.8 is the localization algorithm to locate the Trojan on the network. In this

algorithm, (x_f, y_f) is the node id that the AU detected the first tampered packet. (x, y) is the worm location and initial value is the equal to (x_f, y_f). For the NoC design input parameters, the maximum threshold value (T_t) and maximum waiting time (T_s) are calculated in line 1 and line 2 for the worm movement. There are four steps to locate the Trojan by the worm. Step one is activating all LUs in the column x_f and move the worm along vertically (line 7). When a LU captures the tampered packet, the worm would stop in its node. Step two is doing the row moving. The worm moves horizontally and if a LU detects tampered packet, the worm will turn to relative row. If no tampered packets are detected, the current node of the worm is treats as the Trajan position. Step three performs row movement and the worm position is updated all the time. Finally, the worm position is regarded as the Trojan location on the network.

Chapter 3

Methodology and Implementation

As discussed in the previous chapters, progressive data packet authentication could detect the Trojan in low overhead and EETD could locate the Trojan. In this Chapter, we introduce a new design that combines the advantages of these two methods and a brief detail of the implementation.

3.1 System Overview

The new design has a similar system overview of EETD. The solution also has two detection units. One is AU in progressive authentication and the other one is LU described in EETD. The detection theory of EDU could be considered as traditional authentication and this unit causes higher overhead for the large tags.

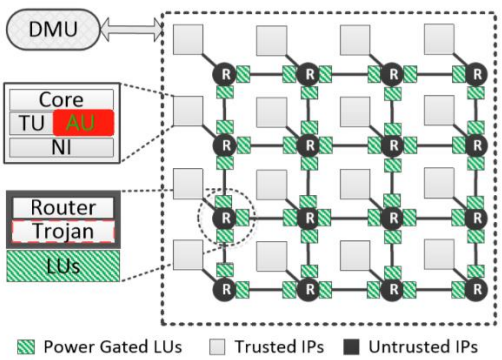

Figure 3.1: Structure Overview of Solution

The components in figure 3.1 remain the same except the AU. This design has the same

detection process. The AUs are attached to all nodes and activated all the time. The AUs calculate the tag based on the incoming packets. Once they determine the tampered packets, a tampered signal is sent to the DMU and the LUs start to locate the Trojan on the network. Once the LUs are activated, the localization algorithm starts to search the Trojan location. The LUs in all rows and all columns capture the tampered packets within the threshold time. The intersection of the detection tampered packets is tread as the Trojan location.

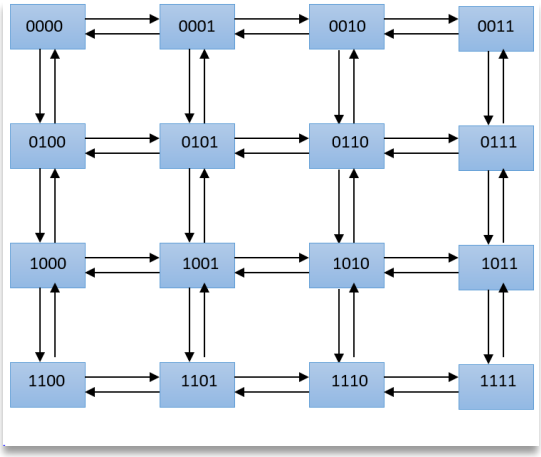

Figure 3.2: 16-nodes NoC

The network is 16 nodes (Figure 3.2) and the nodes connects to the IP core, a network adapter and a router. The network adapter links a PU and a router. The router connects the node to the whole network.

3.2 Architecture

3.2.1 Packet

The packet type in the network is write packet. This type of packet consists of the write data and write address from the PU. The destination node would receive the packet and store the data into the memory.

48 - 43	42 - 40	39-36	35 - 28	27-0
x/y	flags	Target ID	address	data

Figure 3.3: Packet Format

The first 6 bits are for router counting. The first 3 bits (bit-48 to bit-46) are for y directions of the grid and the next 3 bits (bit-45 to bit-43) are for x directions. The values are calculated in the network adapters and decreased in the routers. One bit

decides the direction for each axis and the other two bits determine the distance between the current node and the target node. The network is 4 by 4 mesh network and the number of bits would be increased when a larger network is implemented.

The packet has a fixed length of 49 bits (Figure 3.3). The flag is 3 bits from bit-42 to bit-40 that indicates the type of packet. Bit-42 is set to 1 when the packet is a write packet and the flags are all zero for the empty packet.

The packet needs to send to a specific location and a destination address is necessary. There are 12 bits address space where the first four bits (bit-39 to bit-36) are used for the target node id and the rest bits are used for the local address. If a larger memory is required, the size of the address have to be increased.

A write packet has 28 bits space for the write data to make sure the integrity of data. The packet does not contain any additional information so that the routers decode the packet without other effects.

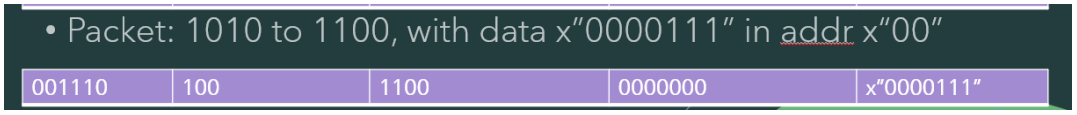

Figure 3.4: Packet Example

Figure 3.4 is an example of the write packet. Assume a packet is sent from node 1010 to node 1100 with the data "0000111" in hexadecimal. The first six bits are router counting and determined by the network adapter in node 1010. The next 3 bits are flags "100" to indicate the packet type is write packet. The next 12 bit contains address and the first four bits are the target id, which is 1100. The rest 28 bits consist of write data.

3.2.2 Network Adapeters

3.2.2.1 Master

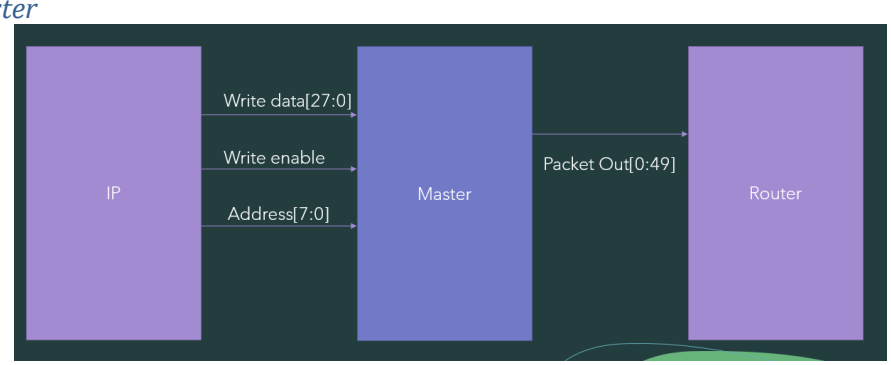

Figure 3.5: Master Network Adapter

Figure 3.4 is the structure of master network adapter that receives the write data, write enable and address signals from the IP core. It connects to a router and sends the packet

out to this interface.

3.2.2.2 Slave

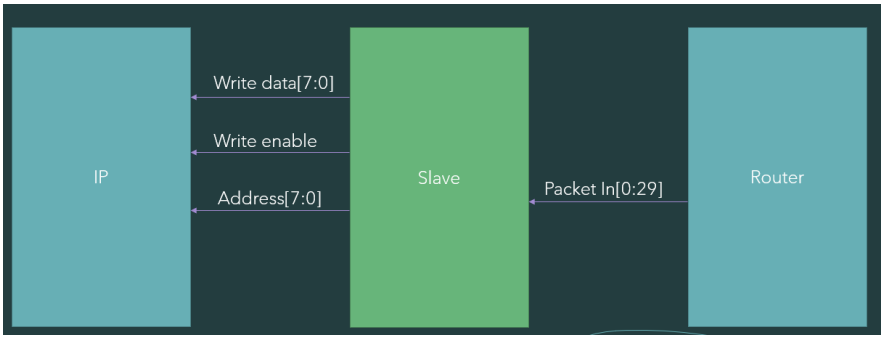

Figure 3.6: Slave Network Adapter

Figure 3.5 is the structure of slave network adapter that sends the write data, write enable and address signals to the IP core. It connects to a router and receive the packet in form this interface.

3.2.3 Interface

This section represents the code for master interface and slave interface. The interface in the network adapter has to connect to the IP core. Figure 3.6 shows how the interface in the master network adapter works. The example is sending a write packet from node 1111 to node 0000. The write enable signal ‘wr’ is set to ‘1’ and the write data ‘wr_data’ is “01110011”. The address is ‘targetid’ and ‘addr’ show in the figure.

```
-- write packet from 1111 to 0000 in addr x"00"
wr <= '1';
targetid <= "0000";
wr_data <= "01110011";
addr <= x"00";
```

Figure 3.7: Master Interface

The interface in the slave network adapter decodes the packet and stores the ‘wr_data’to the ‘addr’ address in the memory. “to_integer(unsigned())” is converting the ‘std_logic_vector’ into ‘integer’ in VHDL code.

```
if wr = '1' then
    register_array(to_integer(unsigned(addr(7 downto 0)))) <= wr_data(7 downto 0);
end if;
```

Figure 3.8: Slave Interface

3.2.4 Router

Routers have five inputs ports and five output ports. It has four directions to identify in the network: north, east, south and west. The additional port is the local channel to the local network adapter. The router has one level buffer that is used to store the information of one of the packets when there are a pair of packets requesting the same router.

3.2.4.1 Router Calculation

The router counting determining the transferring path is calculated by the network adapter. The values are determined by the target node id from the incoming packets and the unique node id of the local network adapter. To get the values of 'y_count', the first two bits in the target node id subtracts to the first two bits in its own node id. The 'y_sign' is set to '1' when the result is negative. The 'y_sign' is '0' if the result is positive. The value of 'x_count' and 'x_sign' have the same calculation but using the last two bits of the target node id and current node id.

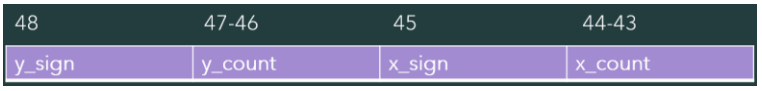

Figure 3.9: 6 bits Router Counting

Using the example packet in figure 3.4, the target id is "1100" and assume the current node is source node. The first two bits of target node id is "11" and then subtracts to the first two bits in current node id, "10". The result is positive, therefore, the 'y_sign' is set to '0' and the 'y_count' is the result, which is "10". The last two bits of target node, "00", minus to the that of current node, "10" to produce a negative result, The 'x_sign' is set to '1' and the result 'x_count' is "11".

3.2.4.2 Router Algorithm

In order to move the packets from the source node to the destination node, the router lokks at the router counting to determine the next movement.

The 'y_sign' and 'x_sign' indicates the direction of the routing. From the figure 3.10, if the 'y_count' has values (line 3), the router checks the 'y_sign'. If the 'y_sign' is '1', the router passes the packets to the north direction. The packets move to the south direction when the 'y_sign' is '0' (line 4 to line 8). When the 'y_count' is all zero and the router would determine the horizontal direction. If the 'x_count' has values (line 9), the router checks the 'x_sign'. If the 'x_sign' is '1', the router passes the packets to the west direction. The

packets move to the east direction when the ‘x_sign’ is ‘0’ (line 10 to line 14). The packets stay in the current router when the ‘y_count’ and ‘x_count’ both are zero. Line 1 indicates the packet is an empty packet and has nothing changes in the network.

```
1  if pkt_in(42) = '0' and pkt_in(41) = '0' and pkt_in(40) = '0' then
2    direction := 6;    --  null
3  elsif pkt(47 downto 46) /= "00" then
4    if pkt_in(48) = '1' then
5      direction := 1;    -- north
6    else
7      direction := 3;    -- south
8    end if;
9  elsif pkt(44 downto 43) /= "00" then
10   if pkt_in(45) = '1' then
11     direction := 4;    -- west
12   else
13     direction := 2;    -- east
14   end if;
15 else
16   direction := 0;      -- local
17 end if;
```

Figure 3.10: Routing Algorithm VHDL code

3.2.5 Localization Algorithm

The localization algorithm of the detection unit is explained in this section. From the figure 3.11, the maximum threshold time is calculated based on a given NoC input design parameters in this algorithm (line 8). When the first tampered packet is detected at node (x_f, y_f), the AU in that node send the tampered signal to the DMU. Then, all LUs in the detection unit are activated by the DMU to supervise the passing packets in all rows and all columns.

If there are no tampered packets are detected and the time has exceeded the maximum threshold time, the node (x_f, y_f) is determined as the Trojan location. The row y and column x are updated for every new detection.

```
Localization Algorithm:
input:  The node location where the first tampered packet
            is detected, (x_f, y_f)
        NoC Network
        NoC Parameters
outputs:Trojan Location(x_h, y_h)

T = calThresholdTime(x_f, y_f)
activateAU()

while waitTime < T and Trojanrow is not found
        and TrojanCol is not found do
    y = wormMoveAlongCol()
    x = wormMoveAlongRow()
end while

(x_h, y_h) = (x, y)
```

Figure 3.11: Localization Algorithm

It is critical to calculate the maximum threshold time. If it is too small, the detection unit stop before the fully searching in the network and might cause a fault positive detection. If it is too long, the consumption is increasing.

Here is the equation to calculate the maximum threshold time:

$$T = (L*a*B + R) \times H$$

Among them, L is the packet length, a is the number of input ports of one node, B is the buffer level, R is the basic packet latency over a node without considering any buffering delay and H is the number of passing node.

3.2.6 Tag Table

The tag table consists of a set of tag segments that generated by the PU in the source node. Every segment is responsible to a specific node. The tag segments are control by a dedicated key from the key table.

```
-- tag table
-- key : 28, 71, 200, 1, 172, 120, 17, 92, 217, 9, 133, 97, 85, 154, 92, 63
tag_0 : tag_array(0)  <= std_logic_vector(TO_UNSIGNED((to_integer(unsigned(sig_flagdata)) + 1 + to_integer(unsigned(key_array(0)))), 8));
tag_1 : tag_array(1)  <= std_logic_vector(TO_UNSIGNED((to_integer(unsigned(sig_flagdata)) + 2 + to_integer(unsigned(key_array(1)))), 8));
tag_2 : tag_array(2)  <= std_logic_vector(TO_UNSIGNED((to_integer(unsigned(sig_flagdata)) + 3 + to_integer(unsigned(key_array(2)))), 8));
tag_3 : tag_array(3)  <= std_logic_vector(TO_UNSIGNED((to_integer(unsigned(sig_flagdata)) + 4 + to_integer(unsigned(key_array(3)))), 8));
tag_4 : tag_array(4)  <= std_logic_vector(TO_UNSIGNED((to_integer(unsigned(sig_flagdata)) + 5 + to_integer(unsigned(key_array(4)))), 8));
tag_5 : tag_array(5)  <= std_logic_vector(TO_UNSIGNED((to_integer(unsigned(sig_flagdata)) + 6 + to_integer(unsigned(key_array(5)))), 8));
tag_6 : tag_array(6)  <= std_logic_vector(TO_UNSIGNED((to_integer(unsigned(sig_flagdata)) + 7 + to_integer(unsigned(key_array(6)))), 8));
tag_7 : tag_array(7)  <= std_logic_vector(TO_UNSIGNED((to_integer(unsigned(sig_flagdata)) + 8 + to_integer(unsigned(key_array(7)))), 8));
tag_8 : tag_array(8)  <= std_logic_vector(TO_UNSIGNED((to_integer(unsigned(sig_flagdata)) + 9 + to_integer(unsigned(key_array(8)))), 8));
tag_9 : tag_array(9)  <= std_logic_vector(TO_UNSIGNED((to_integer(unsigned(sig_flagdata)) + 10 + to_integer(unsigned(key_array(9)))), 8));
tag_10 : tag_array(10)  <= std_logic_vector(TO_UNSIGNED((to_integer(unsigned(sig_flagdata)) + 11 + to_integer(unsigned(key_array(10)))), 8));
tag_11 : tag_array(11) <= std_logic_vector(TO_UNSIGNED((to_integer(unsigned(sig_flagdata)) + 12 + to_integer(unsigned(key_array(11)))), 8));
tag_12 : tag_array(12)  <= std_logic_vector(TO_UNSIGNED((to_integer(unsigned(sig_flagdata)) + 13 + to_integer(unsigned(key_array(12)))), 8));
tag_13 : tag_array(13)  <= std_logic_vector(TO_UNSIGNED((to_integer(unsigned(sig_flagdata)) + 14 + to_integer(unsigned(key_array(13)))), 8));
tag_14 : tag_array(14)  <= std_logic_vector(TO_UNSIGNED((to_integer(unsigned(sig_flagdata)) + 15 + to_integer(unsigned(key_array(14)))), 8));
tag_15 : tag_array(15)  <= std_logic_vector(TO_UNSIGNED((to_integer(unsigned(sig_flagdata)) + 16 + to_integer(unsigned(key_array(15)))), 8));
```

Figure 3.12: Tag Table

The formula to obtain 'sig_flagdata' is here:

sig_flagdata = flag + target_id + data

The flag, target_id and data are from the network adapter in the source node. All tag segments has a fixed length of 8 bits.

3.2.7 Key Table

Each node has a dedicated key for the local tag generation. The key table and tag table are only accessible via processing unit in the node and the AUs.

The keys are in 8 bits, and here are the key values in hex used in the thesis.

```
key_0 : k0    <= x"1C";
key_1 : k1    <= x"47";
key_2 : k2    <= x"C8";
key_3 : k3    <= x"01";
key_4 : k4    <= x"AC";
key_5 : k5    <= x"78";
key_6 : k6    <= x"11";
key_7 : k7    <= x"5C";
key_8 : k8    <= x"D9";
key_9 : k9    <= x"09";
key_10 : k10  <= x"85";
key_11 : k11  <= x"55";
key_12 : k12  <= x"9A";
key_13 : k13  <= x"5C";
key_14 : k14  <= x"3F";
key_15 : k15  <= x"E7";
```

Figure 3.13: Key Table

3.3 Python

In order to observe the results more intuitively, this thesis established a python code to draw the transmission path of the data packet in the NoC. The VHDL code collects the status signal of all physical links (1:busy, 0: idle) and save into "output-results.txt" file. The 'c_WIDTH' is set to 24 since there are totally 24 physical links in the network.

```
process
    variable v_OLINE     : line;
begin
    file_open(file_RESULTS, "output_results.txt", write_mode);
    wait for 1000 ns;
    for k in 0 to c_WIDTH loop
        write(v_OLINE, network(c_WIDTH - k - 1), right, 1);
        writeline(file_RESULTS, v_OLINE);
    end loop;
    file_close(file_RESULTS);
    wait;
end process;
```

Figure 3.14: Status Signal of Physical Links

The python code draws the packets path in the network.

From figure 3.15, a simple 4 by 4 grid is created in line 7 to line 9. Then, the unique node id is assigned to each node (line 11 to line 14).

A new library 'df' is initial to zeros in line 17 and allocated the values from the "output_result.txt" text (line 25). After all information preparing, line 29 draws out the 2D mesh network and the transmission path of the packets.

```
 7 N=4
 8 G = nx.grid_2d_graph(N,N)
 9 labels=dict(((i,j),i + (N-1-j)*N) for i, j in G.nodes())
10 # node id
11 nx.relabel_nodes(G,labels,False)
12 inds=labels.keys()
13 vals=labels.values()
14 grid_pos=dict(zip(vals,inds)) #Format: {node ID:(i,j)}
15 plt.figure()
16 # initial value
17 df = pd.DataFrame({
18         'value':[0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0,0]})
19 # read busy from file
20 f = open("output_results.txt", 'r')
21 nums = f.readlines()
22 nums = [int(i) for i in nums]
23
24 # update value
25 df['value'] = nums
26
27 print("Second Packet from 0100 to 1110")
28 # draw the network
29 nx.draw(G,pos=grid_pos,with_labels=True,
30                 edge_color=df['value'],width=5.0,node_size=800)
31 plt.show()
```

Figure 3.15: Python Code

Chapter 4

Testing

All parts of the Trojan detection were tested and analyzed concurrently during the establishment process. The tests are focus on collecting the results of Trojan detection and localization methods to ensure the expected behavior. This chapter presents the test results of this thesis and gives the evaluation. All tests were run in ISE Design Suite platform.

To test the correctness of our algorithm, we had three tests in different aspects:

- A test ensures that the system can correctly detect and locate Trojan
- A test that can determine whether a data packet is attacked by a Trojan
- A test that can detect different types of Trojan attacks

4.1 Test 1: Simple Test

The first test is a simple test that the detection methodology could detect and location the hardware Trojan.

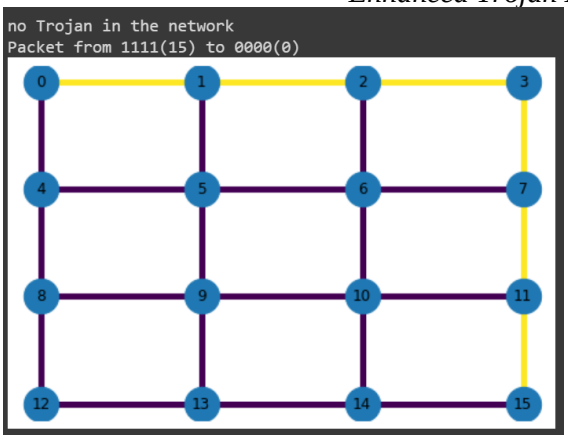

Figure 4.1: No Trojan In Test 1

Figure 4.1 is an example network implementation by sending a write packet from node 1111 to node 0000. Since the Trojans were not activated in this case, the packet reached to the destination node without maliciously attacks. The routers first moved the packet vertically and then delivered it to the node 0000 horizontally.

In figure 4.2, when the Trojan in node 0011 (node 3) was activated and the Trojan had abnormal intention to the passing packet, we found that the packet was disappear in node 2, which means the AU in node 2 detected the tampered packet and dropped it. After that, the AU sent a signal to the DMU and then the DMU activated all the LUs in the detection unit. In the detection unit, the LUs captured the packets from all directions and locate the Trojan in the network. In figure 4.2, the color of node 3 was emphasized to indicate the Trojan location.

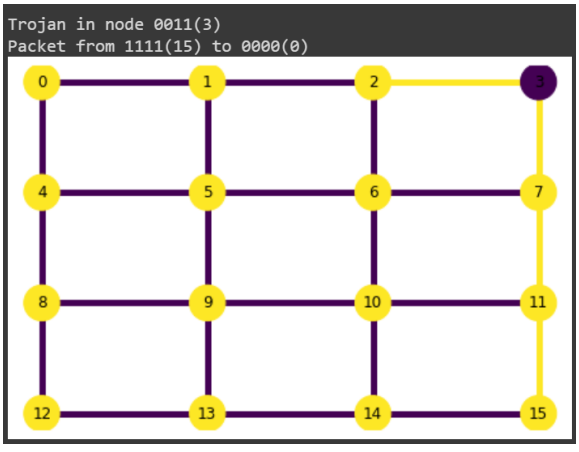

Figure 4.2: Trojan in node 3 – Test 1

The Trojan detection used the progressive packet authentication, so the tampered packet was detected immediately once the packet was modified by the Trojan. These two examples explained the detection method could correctly detect and locate the Trojan in the network.

Figure 4.3 is a simulation process of the example in test 1. The parameters 'trojan_activated' is a signal that the AUs detect the first tampered packet in the network and the DMU makes a decision to active the localization unit. The signal 'sig_reset' is a new reset signal in the detection unit (localization unit) to make sure the results are updated in every new detection within the threshold time. The result is collected when the 'catch_res' is set '1' since one localization finishes at this time. 'sig_stop' is the length of the maximum threshold time. 'sig_row_count' and 'sig_col_count' determine the result 'trojan_pos'. The 'trojan_location' is the Trojan location and this parameter is only for comparison.

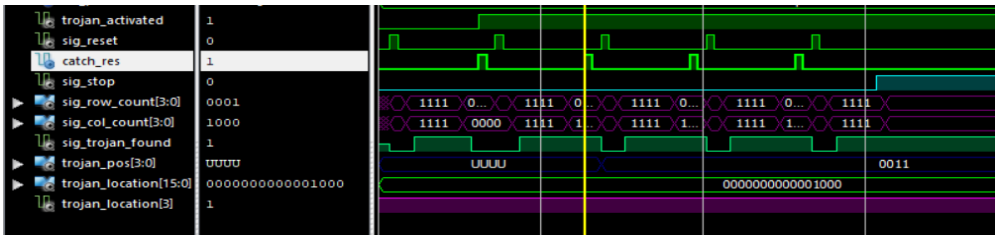

Figure 4.3: Simulation of Test 1

From the figure, the localization has four stages to locate the Trojan in the network. Stage 1 is the network detects the Trojan. The 'trojan_activated' is set to 1. Even though the 'sig_done' and 'catch_res' are activated, the detection unit does not start to find the Trojan location because the detection unit is deactivated. Stage 2 is activating the detection unit and it gets the 'trojan_pos' in the second 'catch_res' is 1. Stage 3 is keeping checking within the threshold time. Stage 4 is stopping the detection and localization and then deactivating the 'catch_res' signal to make sure the trojan_pos will not change after the termination.

For the example in test 1, the 'trojan_pos' shows that the localization unit found the Trojan locates at node 0011, which is node 3. The result is consistent with the expected Trojan location.

4.2 Test 2: Packets from Various Nodes

4.2.1 Trojan in Source node

The second test is a comparing the packets from the various nodes and some packets pass through the Trojan.

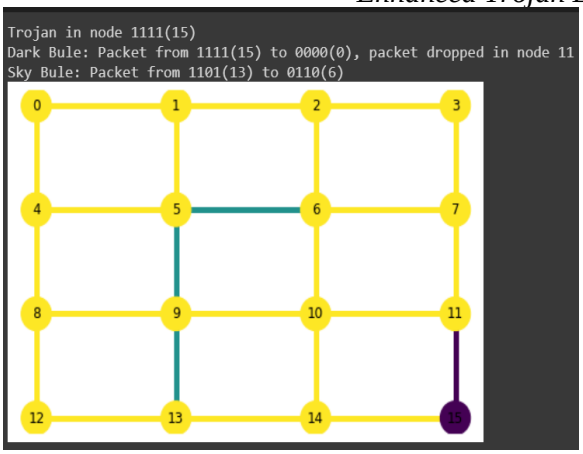

Figure 4.4: Example Case in Test 2

The example in figure 4.4 has two different packets transmission. The dark bule path is the packet that is sent from node 1111 (node 15) to node 0000 (node 0). The packet in sky bule moves from node 1101 (node 13) to node 0110 (node 6). In this case, the Trojan is located in node 1111 (node 15), which is the source node of the packet in the dark bule path.

The packet in dark bule path was dropped in node 11 as the AU in node 11 detected the tampered packet. The DMU received the signal from the AU in node 11 and activated the LU to locate the Trojan position. Therefore, node 15 in the figure was underlined and treated as the Trojan location.

The packet in sky bule path had no effect on the Trojan and reached the destination node, which is node 6. The packet moved vertically and then turned to east in node 5.

Here is the simulation process in this case. Figure 4.5 shows the localization process of packet in the dark bule path. The ‘pkt_drop’ is set to ‘1’ when the AU in node 11 dropped the packet and sent the tampered signal to the DMU. The localization unit is activated and the AUs at all nodes captured the tampered packets from all directions. The result in ‘trojan_pos’ was “1111”, which means the Trojan location was in node 15. The Trojan location was matched to the expected value.

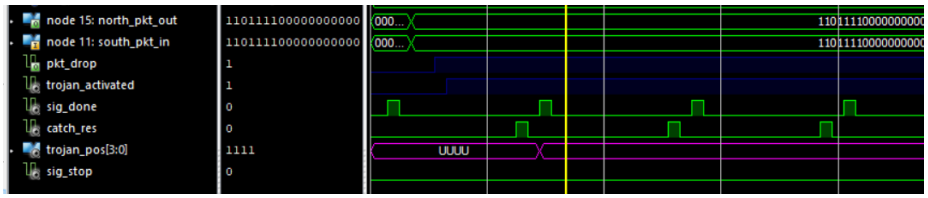

Figure 4.5: Simulation of Dark Bule Path – Case 1

Figure 4.6 is the simulation of the packet in sky blue path. The localization process is the same. However, no signals were sent from the AUs in the passing nodes and the LU remained the deactivate status.

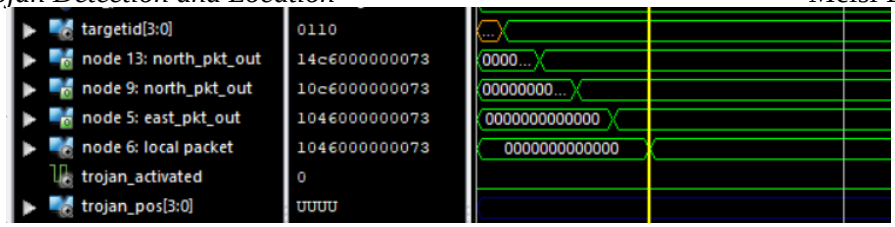

Figure 4.5: Simulation of Sky Bule Path – Case 1 & 2

4.2.2 Trojan in Passing Node

Here is another example containing two different packets. These two packets used the same information as mention in section 4.2.1. The different is the Trojan location. The Trojan in node 3 was activated at this time. From the figure, we found that the packet in sky bule path reached to the destination node without Trojan attacks and the packet in dark bule path was dropped in node 2. At this time, the LU found that node 3 was the Trojan location, which was the expected result.

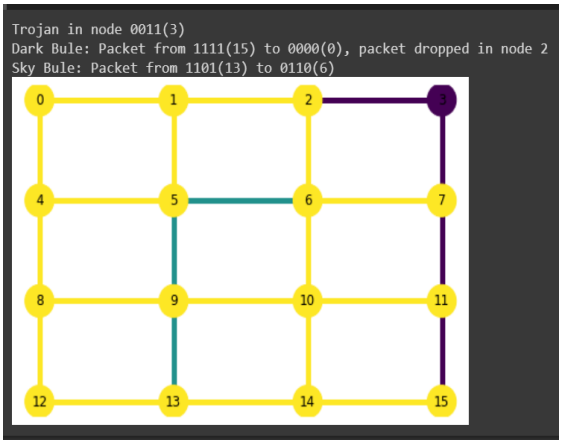

Figure 4.6: Trojan in Passing Node

The simulation of packet in sky bule path was described in previous section (figure 4.5). The simulation of packet in dark bule path shows in figure 4.7. We found that the packet was sent from node 15 to node 11. And node 11 passed the packet to node 7. The Trojan in node 3 modified the packet and the AU in node 2 detect the tampered information. The LU was activated by the DMU to locate the Trojan, which is “0011”. The Trojan was in node 3 in the detection unit.

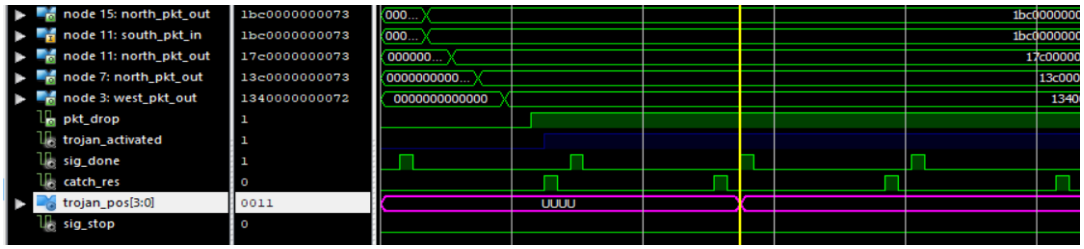

Figure 4.7: Simulation in Dark Blue Path – Case 2

4.3 Test 3: Different Type of Trojan

The last test was using different type of Trojans in the network and observer the different effects. The Trojan could modify the data in the packet, change the packet type and divert the packet the wrong position.

Figure 4.8 shows an example packet from node 0100 (node 4) to node 1110 (node 14). The Trojan was inserted in node 1100 (node 12). The packet was dropped in node 13 and node 12 was in bule color that was treated as the Trojan location.

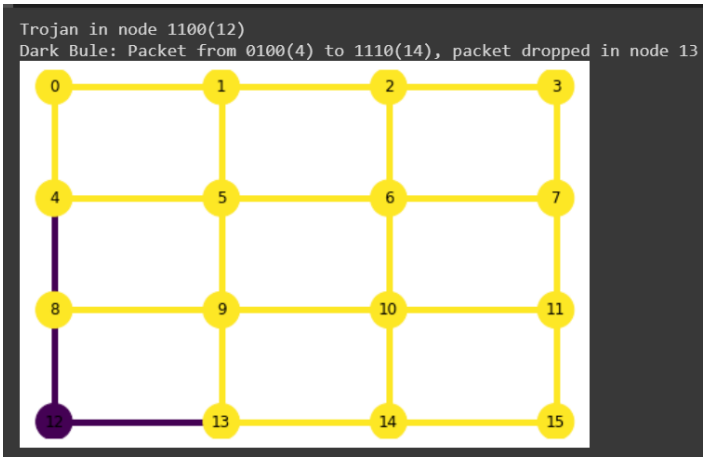

Figure 4.8: Test Example – Test 3

Figure 4.9 is the Trojan that modifies the data in the packet. From line 2 to line 5, the first 48 bits remained the same, but the last bit was changed from ‘1’ to ‘0’. The last 28 bits contained the data and the Trojan changed the data in this case.

The LU was activated by the DMU as previous mentions and found the result ‘trojan_pos’ was “1100”, which is node 12 as expected.

Figure 4.9: Trojan Modifies the Data

The flags in figure 4.10 was different from the value in node 8 and that in node 12. The Trojan in node 12 tended to change the packet type. In this thesis, there is only one type of packet. In this case, if the flags was not equals to "100", the packet was regarded as a invalid packet. The result from the LU was "1100" as well.

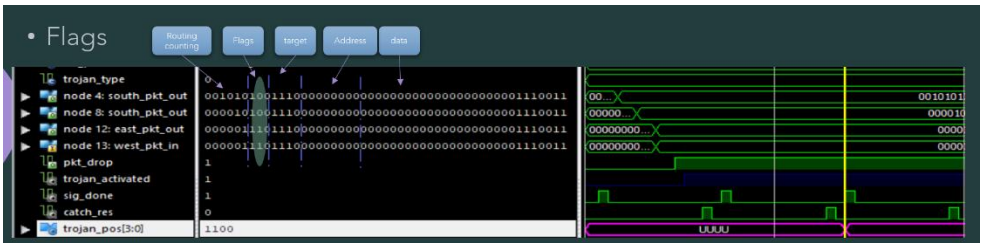

Figure 4.10: Trojan Changes the Packet Type

The last Trojan type is modifying the target node id and diverts the packet to other nodes. We could see the target part in the figure 4.11 was "1111" at the last. The expected target node (node 14) would not receive the packet and the write data would be stay in the node 1111 (node 15).

Figure 4.11: Trojan Diverts the Packet

From above three cases, all kinds of Trojan are detected in this thesis and the LU could locate the correct the position of Trojan. Any types of Trojan cannot escape the security detection in the system.

Chapter 5

Evaluation

This thesis was combined the design ideas of progressive packet authentication and EETD. Therefore, the new design could detect the Trojan in a low overhead and locate the Trojan accurately. This chapter shows the discussing to the last chapter and the evaluation of the new design.

5.1 Results

The results from the previous chapter shows that the detection and localization result in a great outcome wherever the Trojan is and whatever the Trojan type. We have successfully coded the proof methodology from Section 2.2.3.
The module has the needed number of logic units required for the design. The table 1 states the system utilization and the statistics information of the device.
The routers occupy the most part of the logic unit. The size of the logic unit is depending on the size of the NoC. The Flip-Flop are used in the routing algorithm, the tag generator, and the tag segments comparison. The LUT circuit is used in the router algorithm. The system outputs the status of all physical links which size is 24.

Slice Logic Distribution	
Number of LUT Flip Flop pairs used	4
Number with an unused Flip Flop	1 out of 4
Number with an unused LUT	3 out of 4
Number of fully used LUT-FF pairs	0 out of 4
Number of unique control sets	2

IO Utilization	
Number of IOs	26
Number of bonded IOBs	26
Specific Feature Utilization	
Number of BUFG/BUFGCTRLs	1

Table 5.1: Device Utilization Summary

5.2 Discussion

Comparing to the progressive packet authentication, the accuracy and the probability of successful packet transmission are increasing. The LU in the network detect and location the Trojan and then, the following packets might escape the Trojan node. In addition, the localization unit could quickly capture the tampered packets and narrow the attack areas. Comparing to the EETD, the new design decreases the overhead and improved the performance since the detection unit is using the AU stated in the progressive packet authentication. Due to the large tag size, the original detection unit, EDU, might have high power consumption and compute the fault results in some cases.

The new design could deal with the power consumption problem and precisely locate the Trojan in the network.

Chapter 6

Conclusion

This thesis mainly addresses the detection and localization of Trojan. We have successfully designed, implemented and tested a NoC in ISE Design Suite. We learned how to search for relevant documents and extract the key information. At the same time, we gained knowledge about compiling the design concepts in VHDL code flexibly. To choose one packet in a network makes the design simple and avoids unnecessary conflicts.

6.1 Future Work

This thesis has three main improvements that we consider currently. It is necessary to implement when the network transfer two more packets at the same time.

- Tag table improvement.
  The tag table could hold tag for maximally L alive packets in the network. There are L entries in the tag table and every entry has N tag segments with a status bit. N is the network size. The status bit controls the use of the tag entry in the table for a packet. The status bit is '1' when the tag segments in such entry are valid. This kind of tag table could deal with the misdetection when the router receives the different packets.
- Higher level buffer in routers

A router could process one packet at one cycle. When the router receive more than one packet, some packets might be misjudged and dropped. The router should have maximum L level to store the incoming packet.

- Packet queue

  It is expensive to create buffer levels for numerous packets. A queue is a good choice to deal with in the real-application.

Bibliography

[1] A. Malekpour, R. Ragel, A. Ignjatovic, S. Parameswaran, "Trojanguard: Simple and effective hardware trojan mitigation techniques for pipelined mpsocs", Proceedings of the 54th Annual Design Automation Conference 2017 DAC '17, pp. 19:1-19:6, 2017.

[2] Bhunia, S.; Hsiao, M.; Banga, M.; Narasimhan, S. Hardware Trojan Attacks: Threat Analysis and Countermeasures. Proc. IEEE 2014, 102, 1229–1247.

[3] Chenyu Yan, B Rogers, D Englender, D Solihin, M Prvulovic, "Improving cost performance and security of memory encryption and authentication", Proceedings. 33rd International Symposium on Computer Architecture, 2006. Jonathan Frey, Qiaoyan Yu, "A hardened network-on-chip design using runtime hardware trojan mitigation methods", Integration the VLSI Journal, vol. 56, pp. 15-31, 2017.

[4] Frey, J., 2015. Mitigation Of Hardware Trojan Attacks On Networks-On-Chip. [online] Scholars.unh.edu. Available at: <https://scholars.unh.edu/cgi/viewcontent.cgi?article=2069&context=thesis> [Accessed 12 August 2020].

[5] M Hussain and H. Guo, “A Bandwidth-Aware Authentication Scheme for Packet Integrity Attack Detection on Trojan Infected NoC” - IEEE Conference Publication. [online] Available at: https://ieeexplore.ieee.org/document/8645051 [8-10 Oct. 2018].

[6] M Hussain ; A Malekpour ; H Guo ; S Parameswaran , “EETD: An Energy Efficient Design for Runtime Hardware Trojan Detection in Untrusted Network-on-Chip” - IEEE Conference Publication. [online] Available at: https://ieeexplore.ieee.org/document/8429391 [8-11 July 2018].

[7] S. Moein, S. Khan, T. A. Gulliver, F. Gebali, and M. W. El-Kharashi, ‘‘An attribute based classification of hardware Trojans,’’ in Proc. Int. Conf. Comput. Eng. Syst., Cairo, Egypt, Dec. 2015, pp. 351–356.

[8] S. Bhasin and F. Regazzoni, “A survey on hardware trojan detection techniques,” in Circuits and Systems (ISCAS), 2015 IEEE International Symposium on. IEEE, 2015, pp. 2021–2024.

[9] Tsai, W., Lan, Y., Hu, Y. and Chen, S., 2012. Networks on Chips: Structure and Design Methodologies. Journal of Electrical and Computer Engineering, 2012, pp.1-1