

**INSERTION AND DETECTION OF HARDWARE TROJANS USING SIDE-
CHANNEL ANALYSIS**

*A Project report submitted in partial fulfillment of the requirements for
the award of the degree of*

BACHELOR OF TECHNOLOGY

IN

ELECTRONICS AND COMMUNICATION ENGINEERING

Submitted by

P.Akhil(316126512045)

Ch.Shalini(316126512010)

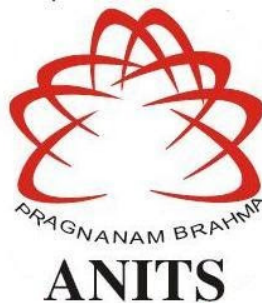
G.S.P.M.Manohar rao(316126512018)

Ch.Viswa Yadeedya(316126512070)

Under the guidance of

Mr.P.Devi Pradeep

Assistant Professor



DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES

(Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC

with 'A' Grade)

Sangivalasa, Bheemili mandal, Visakhapatnam Dist.(A.P)-531162

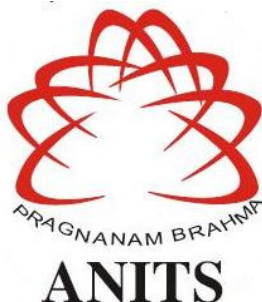
2019-2020

DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING

ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES

*(Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC
with 'A' Grade)*

Sangivalasa, Bheemili Mandal, Visakhapatnam dist.(A.P)-531162



CERTIFICATE

*This is to certify that the project report entitled “**Insertion and Detection of Hardware Trojans**” submitted by **Pilaka Akhil (316126512045), Chinta Shalini (316126512010), G.S.P.M.Manohar Rao (316126512018), Ch.Viswa Yadeeya (316126512070)** in partial fulfillment of the requirements for the award of the degree of **Bachelor of Technology in Electronics & Communication Engineering** of Andhra University, Visakhapatnam is a record of bonafide work carried out under my guidance and supervision.*

Project Guide

Mr.P.Devi Pradeep
Asst. Professor
Department of E.C.E

Head of the Department

Dr. V.Rajyalakshmi
Department of E.C.E
ANITS

ANITS

ACKNOWLEDGEMENT

We would like to express our deep gratitude to our project guide **Mr.P.Devi Pradeep Asst.Professor**, Department of Electronics and Communication Engineering, ANITS, for his guidance with unsurpassed knowledge and immense encouragement. We are grateful to **Dr. V. Rajyalakshmi**, Head of the Department, Electronics and Communication Engineering, for providing us with the required facilities for the completion of the project work.

We are very much thankful to the **Principal and Management, ANITS, Sangivalasa**, for their encouragement and cooperation to carry out this work.

We express our thanks to all **teaching faculty** of Department of ECE, whose suggestions during reviews helped us in accomplishment of our project. We would like to thank **all non-teaching staff** of the Department of ECE, ANITS for providing great assistance in accomplishment of our project.

We would like to thank our parents, friends, and classmates for their encouragement throughout our project duration. At last but not the least, we thank everyone for supporting us directly or indirectly in completing this project successfully.

PROJECT BATCH STUDENTS

P.Akhil(316126512045)

Ch.Shalini(316126512010)

G.S.P.M.ManoharRao(316126512018)

Ch.Viswa Yadeedy(316126512070)

CONTENTS

ABSTRACT	vi
LIST OF FIGURES	vii
LIST OF TABLES	x
LIST OF ABBREVIATIONS	xi
CHAPTER 1 INTRODUCTION	01
1.1 Project Objective	01
1.2 Background	01
1.3 Motivation	03
1.4 Project Outline	06
CHAPTER 2 HARDWARE SECURITY	07
2.1 Introduction	07
2.2 Attacks on Hardware	08
2.3 Attack Circuits	09
2.4 Counter Measures	10
2.5 Hardware Trojan	10
2.5.1 Action Characteristics	11
2.5.2 Payload Characteristics	14
2.5.3 Trojans in CLB/EMB FPGA	15
2.5.4 Types of Hardware Trojans	17
2.5.5 Example Attack	17

2.6 Real Life Implementation	18
CHAPTER 3 INTRODUCTION TO FPGA	20
3.1 What is FPGA?	20
3.2 Theory	20
3.3 Overview of FPGA	26
3.4 Programmability of FPGAs	28
3.5 FPGA Logic Blocks	29
3.6 Modern FPGAs	31
CHAPTER 4 VERILOG HDL	32
4.1 What is HDL?	32
4.2 Importance of HDLs	32
4.3 Introduction to Verilog HDL	32
4.4 Module	33
4.5 Tokens of Verilog	33
4.5.1 Case Sensitivity	34
4.5.2 Keywords	34
4.5.3 Operators	34
4.5.4 Data Types	34
4.5.5 Comments	36
4.5.6 Number Specification	36
4.6 Module declaration	37
4.6.1 Module Instantiation	38
4.7 Flow Chart of Verilog	40
4.8 Software Tools Used	43
CHAPTER 5 HARDWARE TROJAN INSERTION AND DETECTION	51
5.1 Hardware Trojan	51
5.2 Algorithm for Trojan Detection	51
5.3 Hardware Trojan Detection Methodologies	51
5.3.1 Visual Detection	52

5.3.2 Logic Based Testing	52
5.3.3 Side Channel Analysis	54
5.3.4 Ring Oscillators	56
CHAPTER 6 CONCLUSION AND FUTURE SCOPE	60
REFERENCES	61
APPENDIX	63

ABSTRACT

Security of any system has been related to the security of the information being processed. The hardware used for information processing has been considered trusted. This information should not get interfered by the third parties. A trojan horse or trojan is a type of malware that is often disguised as legitimate software. Trojans can be employed by cyber-thieves and hackers trying to gain access to user's systems. A Hardware Trojan (HT) is a malicious modification of the circuitry of an integrated circuit. A hardware trojan is completely characterized by its physical representation and its behaviour. Hardware Trojan Threats (HTTs) are stealthy components embedded inside integrated circuits (ICs) with an intention to attack and cripple the IC similar to viruses infecting the human body. Hardware Trojan horses are a realistic threat for both ASIC and FPGA systems. Visual detection methods, logical based testing, side-channel analysis, ring oscillators etc., can be used to detect the presence of malicious hardware functionality. Unlike software viruses and software trojans, hardware trojans cannot be easily eliminated through firmware by updating, therefore are more harmful to working systems.

Hardware trojans are designed by attackers to add unwanted functionality to the design. There is no a standard procedure to design hardware trojans, as its design is reliant upon the attacker's goals and available resources. This project shows a solution to prevent hardware trojans by detecting them and safeguarding the system with the help of FPGA. Hardware trojan detection methods have been developed to ensure that there is no malicious circuitry embedded in the design. These methods either try to introduce architectural modifications to prevent embedding trojan or try to detect the existence of a trojan which is having a high probability by studying side-channel waveforms. However, these methods mainly depend on comparing the un-trusted chip with a trusted chip. This project is implemented using side channel analysis as it is a compromise of all other methods in parameters like time taken, success rate, infrastructure needed, implementation ability, coverage scope. In addition to these this method has high

performance. The trojan alarm handling unit can partially reconfigure the FPGA to remove the infected core and replace it with a new one while the rest of the FPGA logic is still running. The dynamic trojan detector unit will also mark the anomalous core vendor as a suspected vendor.

LIST OF FIGURES

Fig 1.1	(a) General model of a hardware Trojan circuit realized through malicious modification of a hardware. (b) An example of combinational Trojan. (c) An example of sequential Trojan.	2
Fig 2.1	Block Diagram of Hardware Trojan Inserted Circuit	11
Fig 2.2	Simplified architecture of an FPGA showing the trigger points that a Trojan may use.	13
Fig 2.3	Simplified schematic of digital clock manager (DCM) in Xilinx's Virtex-5 FPGA device	13
Fig 2.4	Diagram showing examples of payloads that can be altered by an implanted Trojan circuit.	14
Fig 2.5	Programmable I/O block containing hardware Trojans to cause logical and electrical malfunction.	16
Fig 2.6	FPGA device with security features for bitstream decryption.	16
Fig 2.7	Hardware Trojan inside: (a) a configurable logic block (CLB) and (b) an embedded memory block (EMB) of FPGA.	17
Fig 2.8	Types of Hardware Trojans	17

Fig 2.9	Simplified diagram of Protocol used	18
Fig 2.10	The Attack	18
Fig 2.11	Real life implementation of Chin & Pin attack	19
Fig 3.1	Xilinx Artix 7 FPGA Trainer kit	21
Fig 3.2	Port numbers of Artix7 FPGA Trainer kit	21
Fig 3.3	FPGA Configure through External PROM	24
Fig 3.4	FPGA Configure through SPI FLASH Memory	24
Fig 3.5	FPGA Configure through Internal FLASH	25
Fig 3.6	Slave Mode	25
Fig 3.7	JTAG connection	26
Fig 3.8	Block diagram of FPGA	28
Fig 3.9	Schematic diagram of FPGA Logic blocks	30
Fig 3.10	Schematic diagram of Coarse grain block	30
Fig 3.11	Schematic diagram of FPGA Design flow	31
Fig 4.1	Representation of a module as black box with its ports.	33
Fig 4.2	Illustration of Scalars and Vectors	35
Fig 4.3	Flowchart representation of Verilog Code	40
Fig 4.4	Opening window of Xilinx 2019.2	44
Fig 4.5	Create project window	45
Fig 4.6	Name and Location entry for project	45
Fig 4.7	Selecting type of the project	46
Fig 4.8	Specifications window for the project	46
Fig 4.9	Summary window of the project	47
Fig 4.10	Add sources	47
Fig 4.11	Creating source	48

Fig 4.12	Creating file window	48
Fig 4.13	Filename creation window	49
Fig 4.14	Sources window	49
Fig 4.15	Sidebar for performing required process.	50
Fig 5.1	The set of Figure	
	(a) From right to left represents the readings for the trojan free, sleeping and active respectively.	
	(b) Demonstrate a close up representation of the trojan free, sleeping and active on the board.	
	(c) Provide close up temperature readings of the trojan active on the board over a period of 10 min.	52
Fig 5.2	Iterative logic-based self-checking test.	53
Fig 5.3	Circuit under test	55
Fig 5.4	Schematic diagram for the circuit in fig 5.3	55
Fig 5.5	Floor planning for trojan free circuit and combinational trojan inserted circuit	56
Fig 5.6	Simulation Results for circuit in fig 5.3	56
Fig 5.7	Ring Oscillator	57
Fig 5.8	Simulation Results for circuit in fig 5.3 using Trojan	57
Fig 5.9	Floor plan of trojan effected circuit	57
Fig 5.10	Schematic of Trojan inserted circuit	58

LIST of TABLES

Table 3.1	Artix-7 FPGA Feature Summary by Device	22
Table 3.2	Artix-7 FPGA Device-Package Combinations and Maximum I/Os	22
Table 3.3	Comparison between programming techniques	29
Table 5.1	Parameters comparison of Trojan Free and Trojan Effected Circuit	58

LIST OF ABBREVIATIONS

HT	Hardware Trojan
HTT	Hardware Trojan Threat
FPGA	Field Programmable Gate Array
MOSFET	Metal Oxide Semiconductor Field Effect Transistor
CLB	Configurable Logic Blocks
EMB	Embedded Memory Blocks
HDL	Hardware Description Language

Chapter 1

INTRODUCTION

Nowadays, hardware trojan protection became a hot topic especially after the horizontal silicon industry business model. Third party IP is the building block of many critical systems and that arise a question of confidentiality and reliability of these blocks. In this work, we present novel methods for system protection and Trojan detection that alleviate the need for a golden chip. In addition, we introduce a scenario to dynamically remove infected IPs embedded on FPGAs. In this project side-channel analysis and ring oscillator based approaches were used to detect Hardware Trojan. Dynamic Trojan detection is done using multiple variant voting. Different methodologies can be implemented for different trojans.

1.1 Project Objective

The main objective of the project is to insert a hardware trojan and detect it using various methods and thereby compare which type of trojan is detected efficiently by which type of methods. The hardware trojan is nothing but a malicious circuit which is placed in the integrated chip without effecting its normal functionality. All these detection methods are followed by a flow of process specified as an algorithm in chapter 5. Everytime the trojan inserted circuit parameters are compared with the golden circuit and then judge if trojan is present or not.

Simulation results are obtained for different circuits which are in Chapter 5.

1.2 Background

Malicious modifications of integrated circuits, referred to as Hardware Trojans, have emerged as a major security threat due to widespread outsourcing of IC manufacturing to un-trusted foundries. An adversary can potentially tamper with a design in these fabrication facilities by inserting malicious circuitry, leading to potentially catastrophic malfunctions in security-critical application domains, such as the military, government, communications, space, and medicine. Conventional post-manufacturing testing, test generation algorithms, and test coverage metrics often fail to detect Hardware Trojans due to their diversity, complexity, and rare triggering conditions. An intelligent adversary can design a Trojan to only trigger under very rare conditions on an internal node, which is unlikely to arise during post-manufacturing test, but can be triggered during long hours of in-field operation.

The detection of Trojans by employing side-channel parameters, such as power trace or delay overhead, is limited due to the large process variations in nano scale IC technologies, detection sensitivities of small Trojans, and measurement noise. Often these issues mask the effect of Trojan circuits, especially for ultra small Trojans. From an adversary's perspective, the desired features for a successful Trojan are as follows: rarely activated to evade logic based testing, low overhead to evade side-channel based detection approach, and low side-channel signature to evade Design for Security (DfS) hardening mechanisms. The

condition of Trojan activation is referred to as the trigger, and the node affected by the Trojan is referred to as its payload. Trojans can be classified based on their triggering conditions or payload mechanisms. The trigger mechanism can be either digital or analog. Digitally triggered Trojans can be classified into combinational and sequential Trojans. Trojan can also be classified into digital and analog based on the payload mechanisms. Digital Trojans invert the logic values at internal nodes or modify the contents of memory locations, while the analog payload Trojans may affect circuit parameters, such as performance, power, and noise margin.

A combinational Trojan is activated on the simultaneous occurrences of a particular condition at certain internal nodes, while a sequential Trojan acts as a time-bomb, exhibiting its malicious effect due to a sequence of rare events after a long period of operation. Fig 1.1(a) illustrates the general scenario of a Trojan attack in a design, where a Trojan is realized through the malicious modification of the circuit with a trigger condition and payload. Fig 1.1 (b) shows an example of combinational Trojan which does not contain any sequential elements, and depends only on the simultaneous occurrence of a set of rare node conditions. Conversely, the sequential Trojans shown in Fig 1.1 (c) undergo a sequence of state transitions before triggering a malfunction. The 3-bit counter causes a malfunction at the node S on reaching a particular count, and the count is increased only when the condition $a = 1, b = 0$ is satisfied at the positive clock-edge. Protection against hardware Trojan has been widely explored by researchers. These approaches are based on the following three approaches:

- (1) Specialized functional testing that rely on triggering an unknown Trojan and observing its effect in output ports of a design;
- (2) side-channel analysis that rely on observing a Trojan effect in physical parameters, such as supply current or path delay and
- (3) design/integration approaches that either prevent a Trojan insertion or facilitate detection during production test.

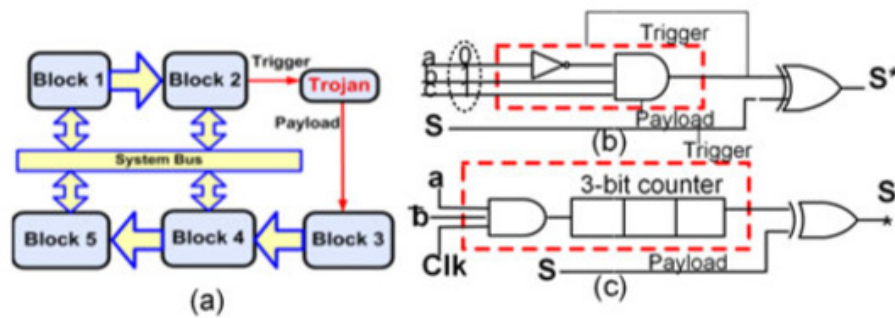


Fig1.1 (a) General model of a hardware Trojan circuit realized through malicious modification of a hardware. (b) An example of combinational Trojan. (c) An example of sequential Trojan.

1.3 Motivation

The main idea for working on hardware trojan detection is to control the cyber crimes to the maximum extent. For example, in the year 2007 the backdoor built into a Syrian radar system was responsible for the system's failure. There are also reports of Trojans being used by the USSR to intercept American communications during the cold war. Time to activate a hardware Trojan circuit is a major concern from the authentication standpoint.

It is also a direct threat to the already vulnerable Internet of Things, meaning that wireless-enabled household devices also become potential targets. The problem is such that even previously 'reputable' factories are vulnerable to attacks, since all that is required is one employee to alter the existing code to include a trojan. As most IC designs are extremely large and contain a huge amount of hardware description, these inclusions are difficult to detect and the sheer size of the code can require many people having access to the code at production level.

Regarding military grade products utilizing ICs, the problem of hardware trojans is critical with the threat level of the trojan being such that it could potentially be catastrophic. Malicious inclusions of code could cause life saving equipment to fail, missiles to lose control, and cryptography keys to be leaked. While incidents of hardware trojans, are not openly discussed there have been a few noted. In 2007, it was assumed that a backdoor built into a Syrian radar system was responsible for the system's failure. There are also reports of trojans being used by the USSR to intercept American communications during the cold war. The problem is aggravated further still when considered in relation to the growth in production of counterfeit goods. Such goods may be produced in less than reputable factories, so the inclusion of malicious code in the production process is far from unrealistic. As counterfeit goods are not generally sold through trustworthy channels, it is impossible to recall products found to be unsafe or indeed to produce updated firmware to deal with emerging threats. This can expose consumers to a plethora of malicious attacks by hackers. For example, a trojan leaking cryptography keys in counterfeit IoT devices could potentially give hackers access to a network of devices that can be utilized in 'Mirai' like attacks and cannot be recalled or patched. In this paper, a hardware trojan is created and emulated on a consumer FPGA board. The experiments to detect the trojan in a dormant and active state are made using off-the-shelf technologies which rely on thermal imaging, power monitoring, and side-channel analysis.

Unfortunately, those theories are not in nature groundless or out-with the realms of the possible. In fact several of them have already been instantiated. One such rumor of 'kill switches' being hidden in commercial processors was confirmed by an anonymous U.S. defence contractor who indicated the culprit to be a 'European Chip Maker'. The potential consequence of the existence of such a switch could be catastrophic. Indeed, as previously highlighted, this particular hardware trojan was blamed for the failure of a Syrian radar to detect an incoming air strike. The Threat of the Hardware Trojan. At Design Level The

complexity and cost of the design of ICs has grown exponentially over the last decade as the semiconductor industry has scaled to sub-micron levels. A typical IC board will go through a rigorous process consisting of several stages. Firstly, the specifications must be translated into a behavioural description, usually in a hardware description language such as Verilog or VHDL. Once this has been completed, the next phase is to perform synthesis to transform the behavioural description into a design implementation using logic gates such as a net-list. Once the synthesis has been completed, the net-list is implemented as a layout design and the digital files are passed to the foundry for fabrication. As well as outsourcing the production of ICs, many companies are also purchasing third party intellectual property (IP) cores, and utilizing third party Electronic Design Automation (EDA) tools. Each use of third-party software presents a new opportunity for attacks such as hardware trojan insertion, IP piracy, IC tampering, and IC cloning. Although these attacks are all of importance, the hardware trojan is by far the most dangerous attack, and, as such, has garnered much attention. At Foundry Level As semiconductor technology has advanced, the cost of owning foundry has increased dramatically. In 2015, the cost was estimated to be in the region of 5 billion USD. As a direct result of this, many companies can no longer afford to fund the production process from start to finish, and are outsourcing their production to cheaper foreign foundries. Whilst undesirable modifications to ICs should ideally be detectable by pre-silicon verification and simulation, this would require a specific model of the entire IC design and this is not always readily available particularly where third party IP cores or EDA tools have been used. In addition, large multi module designs are rarely compliant with exhaustive verification.

Post silicone approaches to design verification include destructive de-packaging and reverse engineering of the IC. However, current techniques do not allow destructive verification of ICs to be scalable. It is also possible for an attacker to infect only a portion of the produced ICs, making these tests futile. Most post silicone logical testing techniques are also unsuitable for detecting hardware trojans. This is attributed to the stealthy nature of the hardware trojan and to the large numbers of differing taxonomies that can be employed by the attackers. Most hardware trojans are programmed to activate under a specific set of conditions, and a skilled attacker would ensure that these conditions were undetectable by the testing routine. This is particularly true of trojans targeting sequential finite state machines. Industries Affected Military Hardware trojans are a huge threat to many industries. However, security conscious industries, such as the military, are in a particularly high risk bracket and defence departments are very aware of this. The U.S. Department of Defense (DoD) has created a “Trusted Foundry Program” to ensure its military equipment remains free of hardware trojans by using only accredited foundries. This means that only American foundries which are located on the American soil and which underwent the strictest vetting process are allowed to work on the chips for the U.S. DoD. In addition to vetting the foundries, close attention is being paid to the other links in the design and supply chain. While this approach may seem effective, it has its limitations. The majority of western foundries are woefully behind their foreign counterparts when it comes

to the level of technology they can provide. This seriously limits access to more advanced chips which are required for modern avionics and weapons systems.

If the attacker creates the trojan through the modification of the existing code, then it will be classified as a parametric. Typically, this can be achieved by thinning wires or weakening transistors and flip flops. This type of trojan is notoriously hard to detect as the alteration can be minuscule. The next physical characteristic the attacker would have to consider would be the size of the hardware trojan. In this context, the size refers to the physical extension of the hardware trojan or the number of components it consists of. In case of a large trojan consisting of many components, an attacker can distribute these across the IC, placing components where they are necessary to execute their payload in accordance with the functions of the hardware trojan. This is known as loose distribution. In contrast, a smaller hardware trojan consisting of only a few components allows for the components to be placed together as they will occupy only a small part of the layout of the IC. This is known as tight distribution.

On rare occasions, a determined attacker could regenerate the layout to encompass the hardware trojan, moving the components of the IC to accommodate the components of the hardware trojan. This is referred to as a structural alteration. **Activation Characteristics** Typically, a hardware trojan will be condition-based, meaning that its activation will be dependent on a trigger defined by the attacker. The trigger itself will generally consist of either a predefined input pattern, or specific internal logic state, or counter value, and can be triggered both internally and externally. An externally triggered hardware trojan will usually consist of malicious logic within the IC that utilizes an external sensor such as a radio antenna. The attacker will then communicate via the compromised component enabling them to trigger the antenna. It is easy to see why this can be extremely dangerous when it comes to security conscious industries such as the military. It is not out-with the realms of the believable to postulate that an attacker could feasibly re-route or switch off a missile via a radio signal as suggested.

Conversely, an internally triggered hardware trojan will look within the circuitry for the set of conditions that will cause it to activate. A typical example of this would be countdown logic. In contrast to the condition-based trojan that will only activate when its trigger conditions are met, the “always-on” trojan is active from the moment of insertion, and relies on internal signals. This type of hardware trojan is generally split into two categories; combinational and sequential. A combinational trojan will activate upon detection of a specific set of circumstances within the internal signals of the IC. Sequential trojans will also monitor the internal signals of the IC. However, instead of looking for a specific condition, they activate when a specific sequence of events occurs.

1.4 Project Outline

The project Hardware Insertion and Detection deals with the hardware security domain where securing information is vital in this sophisticated contemporary world. Day to day technology has been improving along with this the loop holes for hackers are becoming more and causing cyber security problems. In order to prevent and control these problems this project gives a scope to control the problems. By continuous analysis of the circuit the parameters like path delay can be found changed if trojan is present.

A circuit is said to be trojan effected if its parameters shows a significant change with the parameters of trojan free circuit. The trojan free circuit is called the golden circuit. Different trojans can be detected using different methodologies. Most popular and effective methods are side channel analysis and ring oscillator based detection. All these circuits are simulated using Xilinx Vivado and reports were generated and the parameter were compared for trojan free and trojan effected circuit.

CHAPTER 2

Hardware Security

2.1 Introduction

Hardware is a collection of physical elements that constitutes a computer system. Hardware is used by everyone even if they are not aware of it. Hardware in this context might be:

a. Computer Hardware: Some computer hardware are Processors, firmware, memory etc.

i. Processors: is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions

ii. Firmware: is the combination of a hardware device, e.g. an integrated circuit, and computer instructions and data that reside as read only software on that device

iii. Memory: Memory refers to the device used to store information for use in a computer.

b. Mobile Hardware: Sim Card, RFID/Smart Card, Chip and Pin

i. Sim Card: is an integrated circuit that is intended to securely store the international mobile subscriber identity (IMSI) and the related key used to identify and authenticate subscribers on mobile telephony devices (such as mobile phones and computers).

ii. RFID: is the wireless use of electromagnetic fields to transfer data, for the purposes of automatically identifying and tracking tags attached to objects. For example, an RFID tag attached to an automobile during production can be used to track its progress through the assembly line.

iii. Smart Card: is any pocket-sized card with embedded integrated circuits. Smart cards are made of plastic, and can provide strong security authentication for single sign-on (SSO) with large organizations

iv Chip & Pin: "Chip" refers to a computer chip embedded in the smartcard, and "PIN" refers to a personal identification number that the customer must supply. "Chip and PIN" is also used in a generic sense to mean any EMV smart card technology that relies on an embedded chip and a PIN.

c. Future Hardware: PUFs (Physically Unclonable Functions) PUFs have a unique fingerprint in a physical object that means if you have an object with one fingerprint; another object with the same fingerprint cannot be created. It uses challenge/response for its operations. The challenge/response explains that if a message is sent to a physical object and the physical object is changed to another physical object, and same message is

sent to the second physical object, the two physical objects react differently because of their unique fingerprint.

2.2 Attacks on Hardware

1. **Physical Attacks:** The main thing that differentiates hardware attacks from software attacks is the physicality of the attack done with hardware tools. This raises the bar for hardware attacks because any attacker that wants to perform an attack on the hardware needs to have extensive knowledge of the hardware, unlike the software attacks that can be done by just downloading a vulnerability tool on the internet to perform attacks.

2. Generally, the hardware wants to protect a secret so the secret is embedded in a physical object. For example, the bank card wants to protect your pin, the pin is encoded in the card and if the attacker can probe the chip of the card and read the pin then the card is useless. The secret in the hardware should not be writable even though it provides the information on the card when placed on a terminal. If we consider STRIDE, we have to consider two main points: the Information Disclosure (Confidentiality) which means something is hidden, and Tampering (Integrity) which means it is not writable.

3. **Attack Vectors:** The hardware that would be used to protect the secret would be fabricated by someone in a factory. The factory will either program the secret onto the hardware or send the hardware to the company with memory for read access in one component and write access in another component so that the company can program the hardware and destroy the write component to avoid rewriting. This approach can't be used for all hardware. For example, subway cards need to be rewritten to every month so in that case, the terminals have write access to the card but the user doesn't have write access so it can't be overwritten by the user. To fabricate the hardware, the laboratory/factory needs to be trusted. And to prove they are trustworthy, the laboratory gets certified. The certification body sends people in, to audit them, check out the employees and their procedures and conclude whether the laboratory is trusted or not.

4. **Supply Chain:** After the hardware has been made, it will be shipped to stores that will sell it or it is shipped to the consumers from the store. During shipping it could be intercepted by the attackers and tampered with, then re-packaged without the knowledge of the store or the consumers. Also, some attacks can be performed on point of sale terminals when some insider (employees) have access to the terminals and tampered with it in the warehouse

5. **Accidents:** there are a lot of memory based devices (e.g. USB Keys, Digital picture frames) which may contain malware in them accidentally, which could affect the system of the user. The company that created this hardware may not be aware of the malware on the device. Examples of companies that had this kind of

accidents are IBM, Dell, Samsung, HP, Apple, etc. All the attacks stated above are the main reasons for why the user might get a bad hardware.

2.3 Attack Circuits

RFID, Smart Card, Micro- Controller, ASICs, FPGAs RFID is passive, a signal can be sent to it and it responds. It can't be programmed; it can't perform any computations based on the signal sent to it. Smart card, on the other hand, performs computations based on what was sent to it. Micro-controller is like an ID with no chip, ASICs are fabricated circuits that are custom made to do implementations so all your processors and memories are on ASICs. ASICs are expensive because they are custom made, and before the design is committed to ASICs, a lot of testing will be done with FPGAs to make sure that circuit actually works. FPGAs are programmable chips. When the chip is bought, it's blank and it can be programmed with software to do whatever you want. They are not as fast as ASICs because they are general purpose. FPGAs are faster than software but slower than ASICs. Why are we attacking Circuits? The main reason for attacking circuits is to recover a secret that had been encoded on a piece of hardware or for the attacker to program a certain value to the circuit. The secret could be the actual algorithm itself. Some attackers reverse engineer the algorithm of the RFID or Smart Card to find flaws in the algorithm itself. This is because some developer wants to keep the algorithm used to protect the circuit a secret due to the fact that the developers are not using a standard algorithm. The algorithm used should be a standard algorithm so as to know how to better protect it. The Circuit attacks are:

1. **Black Box Testing:** To perform this attack, the attacker sends an input to the circuit and receives an output. Based on the input and output behavior, the attacker will decide what kind of algorithm to use. An example is Speed Gas RFID which are proprietary stream cipher. The attackers found the documentation and modified it to discover the cipher used and break the circuit. This type of attack is non-invasive, meaning that the card/chip will not be destroyed when probed so it can be used another time. Another method that can be used in black boxing is fuzzing in software security which allows large random inputs to the circuit and get strange responses like undocumented features, factory testing, etc.

2. **Physical Probing:** To perform this attack the attacker sticks a probe onto the chip itself and reads data off the chip. Within a circuit, there is a wire that connects components to each other called the bus and the bus is where the information would be read as the data is moving around in the bus. The data can also be read off the memory location in the circuit. The probe can have a submission precision and it's an invasive method. A lot of circuits are driven by a clock, and if the attacker can slow down the clock it gives a lot of time to the attacker to read the voltage of the circuit

3. **Reverse Engineering:** To perform this attack, the attacker must acquire the smart card and physically expose the circuit. The smart card is manufactured with different layers and each layer is removed until the

physical circuit is exposed. The attacker then takes high resolution photographs of the circuit and uploads it to a computer and uses a code and machine learning application to figure out what the actual circuit does. Once the circuit is figured out, then the algorithm used in the smart card also is exposed and the algorithm can be broken. An example of a smart card where the reverse engineering attack was performed on is the Mifare (Subway card).

4. Fault Generation: Some technologies fail. E.g. A TV providers sends a message to the client asking if the clients wants to renew the subscription. If TV providers don't receive a message NO from the clients, they don't disconnect, so the clients are granted access. So some clients can just cut the power at the right time to prevent the response to the TV providers, since they won't disconnect with no response. This is a non-invasive attack. Other things that can be done are modifying the memory contents (non-invasive), glitch (rapid change) the power or clock (non-invasive), heating up components e.g. with a user (semi- invasive), modify chip e.g. cutting wires (invasive) etc.

5. Side Channel Analysis: To perform this attack, the attackers makes use of the hardware normally but makes sensitive measure of certain things and based on the measurements done, the attacker can infer secrets. An example of things to measure is power (the amount of voltage in an ATM), timing analysis in cryptography (software effect), electromagnetic emission, acoustic sounds (performed on RSA). These are called side channel because they are outside the normal channels. They are non-invasive. It is slower than the normal attacks.

2.4 Counter Measures

1. Obfuscate data (Scramble, encrypt) on buses
2. Obfuscate the ASICS layout, 3D stacking
3. Metal mesh on top of the circuit (if the circuit is probed, it causes a short and the memory resets)
4. Side Channel: physical shields, asynchronous circuits.

Also, a decrease in the signals from the circuits of the hardware like the noise or add artificial noise or low the circuit's power **METHODOLOGIES** There is no good methodology for hardware (that means no static analysis or dynamic analysis of hardware). It is an open question that needs to be researched on. Most of it has to do with domain specific knowledge and it is advisable to follow the requirement engineering process. Common criteria/NIST has protection profiles which provide the properties not how to achieve them.

2.5 Hardware Trojan:

A hardware trojan can be described as a malicious alteration or inclusion to an integrated circuit (IC) that will either alter its intended function or cause it to perform an additional malicious function.

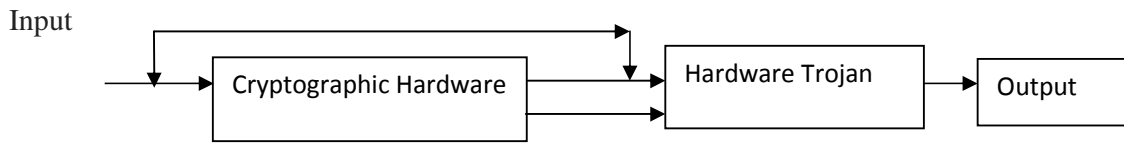


Fig 2.1: Block Diagram of Hardware Trojan Inserted Circuit

These malicious inclusions or alterations are generally programmed to activate only under a specific set of circumstances created by an attacker and are extremely hard to detect when in their dormant state.

Block diagram of typical hardware trojan inserted circuit is shown in fig2. 1.

As technology advances, so does the demand for IC boards leaving many technology companies without the resources to produce secure enough ICs to meet current demands.

This has pushed companies into the ‘fabless’ trend prevalent in today’s semi-conductor industry, where companies are no longer attempting to produce the goods in their own factories, but instead are outsourcing the process to cheaper factories abroad .This growth brings with it a significant rise in the level of threat posed by hardware trojans, a threat that directly affects all companies concerned with products that utilize ICs. This encompasses many different industries, including the military and telecommunications companies, and can potentially affect billions of devices from mobile phones and computers to military grade aviation and detection devices, particularly at a time when wireless devices are being introduced as links in critical infrastructure, compounding trust and security issues even further.

2.5.1 Action Characteristics:

The action characteristics of a hardware trojan refer to the effect the trojan will have on the execution of its payload. Hardware trojans will typically fall into one of two categories: implicit or explicit. Implicit trojans will not change the board’s circuitry of the IC; instead, they will perform their malicious function in tandem with the intended function of the board. This makes these trojans easier to detect as they tend to cause small path delays on activation and consume more power whilst active.

In contrast, an explicit trojan will change the function of the board’s circuitry on activation. This can come in the form of a signal alteration or even leaking of information via predefined board pins. These trojans tend to cause distinct path delays as well as large changes in circuit’s capacity

Hardware Trojan Detection requires overcoming numerous challenges. Namely:

1. Handling large architectures.
2. Being non-destructive to the IC.
3. Being cost effective.
4. Ability to detect trojans of all sizes.

5. Authenticating chips in as small a time frame as possible.
6. Dealing with variations in manufacturing processes.
7. Detecting all trojan classifications.
8. Detecting trojans in a reasonable time frame.

There is no single method capable of detecting all types of hardware trojans, nor overcoming all the challenges described here-above. Over the years, several methods have been developed to detect different types of trojans. These methods are described here-after.

Physical Inspection One of the most obvious method of detection is physical inspection of the board itself. This method is sometimes classified as a failure analysis based technique. Those techniques usually comprise two steps:

- (1) Cutting and lifting the molding coat to expose the circuitry; and
- (2) Performing various scans

Functional Testing Often referred to as Automatic Test Pattern Generation (ATPG) this technique is more commonly used to locate manufacturing faults; it has been shown to be effective in detecting hardware trojans. ATPG involves inputs of ports are stimulated and then the output ports are monitored for variations that may indicate a hardware trojan has been activated. Functional testing techniques can also be useful when attempting to determine the trigger patterns of conditional trojans. **Built-In-Self-Test Techniques** Built-In-Self-Test (BIST) techniques are commonly used to detect manufacturing faults and are present in many chips. If unknown or malicious logic is detected during these tests a bad checksum result is given, although designed to detect manufacturing faults on some occasions these tests can detect hardware trojans

Side channel analysis techniques are some of the most commonly used procedures in hardware trojan detection. These techniques generally measure signals such as power and path delay, looking for fluctuations potentially caused by trojans. Side channel analysis can have a high success rate as even in a dormant state the trojans trigger signal will cause some current leakage

3. Methodology: In order to carry out the investigation our trojan was designed and loaded onto an Basys 3 Artix 7 FPGA board.

Three different detection techniques are demonstrated, the first utilises power analysis techniques as well as side channel analysis, allowing security investigators to measure both the power variance, traces and current leakage, followed by a concentrated heat measurements using an infrared thermometer, and finally a thermal camera test is carried out. The three experiments are carried out using off-the-shelf hardware and are applied to both the trojan-free and trojan-inserted designs. Attempts are then made to detect the trojan in its dormant form. While in in their dormant form trojans do not perform any malicious actions, however, wait

to be activated, through an activation signal, this can be done through the push of a button, or through a specific set of instructions. It is however important to be able to detect trojans in their dormant form, before they activate and perform malicious actions.

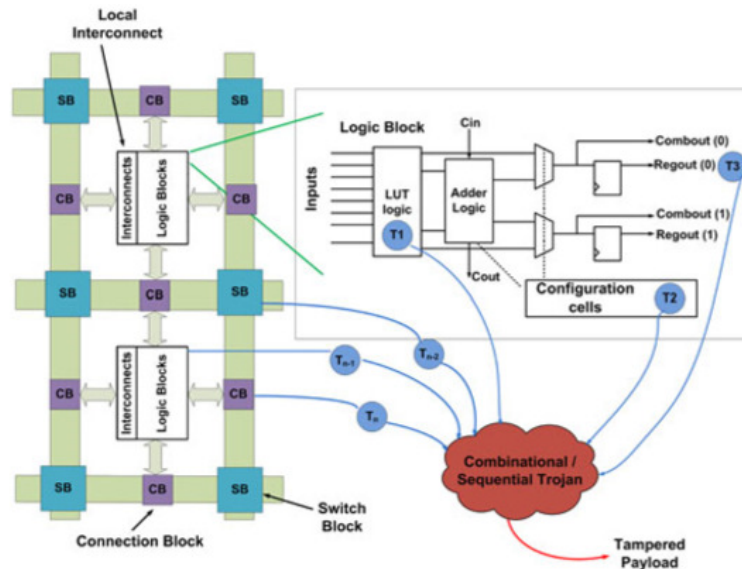


Fig: 2.2 Simplified architecture of an FPGA showing the trigger points that a Trojan may use.

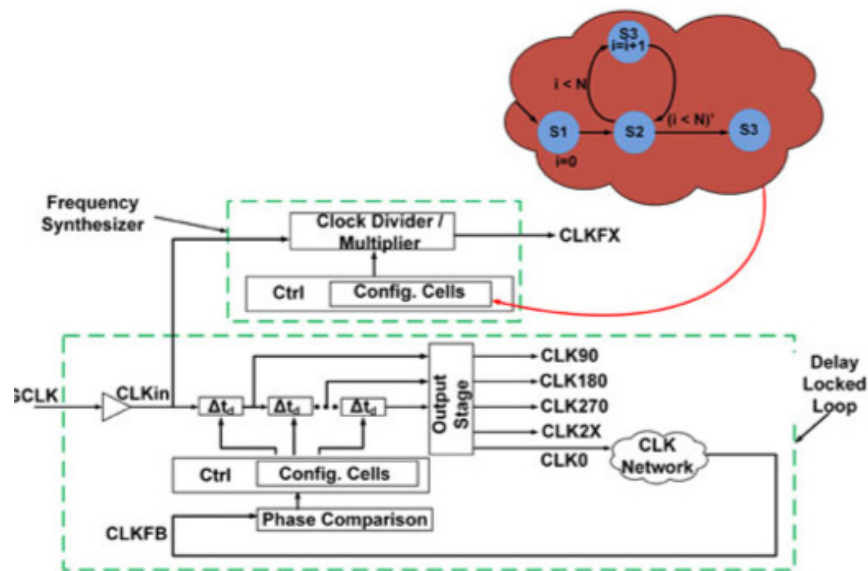


Fig: 2.3 Simplified schematic of digital clock manager (DCM) in Xilinx's Virtex-5 FPGA device

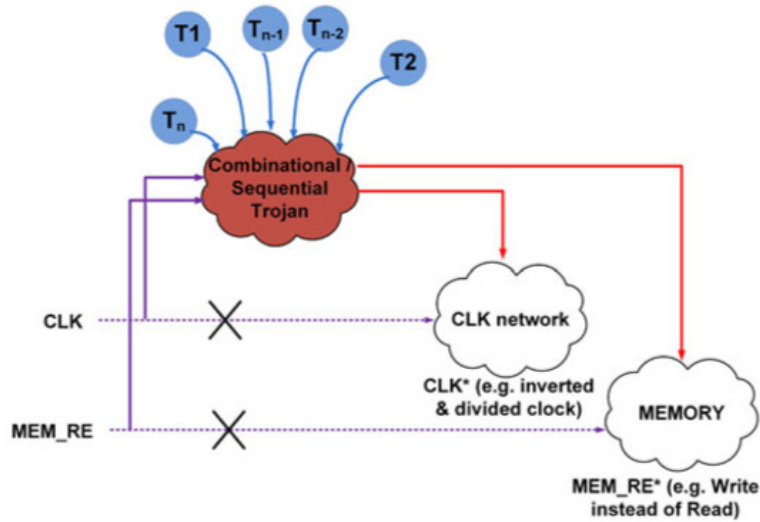


Fig: 2.4 Diagram showing examples of payloads that can be altered by an implanted Trojan circuit.

2.5.2 Payload Characteristics

Hardware Trojans can also be classified based on their intended behaviour. Trojans can be inserted for causing malfunction or for leaking sensitive information. In the former case, Trojans alter the functionality of the design in some way, while Trojans designed for leaking sensitive information may do so without modifying the logic functionality of the design.

Trojans for Malfunction can be further classified into two subcategories based on whether they cause logical malfunction or physical malfunction. Trojans presented in the previous sections cause logic malfunction by modifying the values in the LUTs, causing undesired routing between two logic modules, etc. Fig.2.4 shows additional examples of payloads affected by Trojans.

Trojans intended to cause physical damage can create electrical conflicts at the I/O ports or at the programmable interconnects. Consider the programmable I/O block in Fig. 2.5. When an I/O port is configured to be an input by a design, the configuration cells in the I/O block should disable the output block to prevent internal conflicts. A counter-based Trojan can be inserted in the foundry which detects the state of the I/O port and begins counting. When the counter counts to the final value, the Trojan may enable the output logic when the port is configured as an input. This would cause a high short-circuit current to flow between the FPGA and the external device, possibly damaging the system. These Trojans are similar to the MELT viruses described in except that Trojans causing physical destruction may also be inserted in the foundry.

Since IP designs involve a high development cost and contain sensitive information, security is of utmost importance many high-end FPGAs such as Xilinx's Virtex4 and Virtex5, and Altera's StratixII and StratixIII offer bit-stream encryption to prevent unauthorized cloning of the bit-stream. Fig.2.6 shows the

security features in a generic FPGA device that contains the programmable logic array configuration logic which controls the programming of the SRAM cells in the logic array, interconnect network, and additional modules in the device. The device also contains a decryption module for decrypting the bit-stream using a key stored in a non-volatile memory. Security measures in the device

- (1) Prevent the key from being read and sent to a port by clearing the configuration data and keys when a read attempt is made,
- (2) Prevent read-back of the configuration data, and
- (3) Restrict decryption access after configuration.

However, all of these measures only prevent malicious code in an IP from accessing the key or configuration data. Hardware Trojans can leak the IP in two ways: by leaking the decryption key, or by leaking the design itself. An attacker in the foundry can insert an extraneous circuit (Fig. 2.6) to tap the wires connecting the non-volatile memory and decryption module. Even if the decryption module is implemented in the logic array by using a decryption bit-stream as mentioned in , such an instantiated module must have access to the non-volatile key for decryption. A copy of the key can be stored in the Trojan, which may then leak it through side-channels or covert-channels. Using side-channels, a Trojan can hide the key in the power traces or by emitting electromagnetic radiation containing the information and an attacker can observe these signals to steal the key. For example, the MOLES Trojan presented in uses a spread-spectrum technique to leak the key in the power traces over several clock cycles. Alternatively, a Trojan may also multiplex the JTAG port, USB port, or any other programming port to leak the key through covert channels when the ports are not being used. Since SRAM-based FPGAs are volatile, an external device must be used to store the encrypted design. If an adversary is in possession of the FPGA device loaded with the design, the encrypted bit-stream can be stolen by dropping the connection between an FPGA's programming ports and the external device storing the encrypted bit-stream. In other cases, a Trojan may fake a request to the external device to send the programming data to the FPGA. This time, however, the Trojan MUXes the bit-stream and rather than sending it to the decryption, it may store blocks of the bit-stream at any given time and leak them through side-channels or covert-channels.

2.5.3 Trojans in CLB/EMB FPGA

Configurable logic blocks (CLBs) and embedded memory blocks are highly flexible, but require significant configuration to implement the desired functions. This severely harms the memory or logic integration density in FPGA which makes it more amenable for Trojan insertion. Fig.2.7 shows a FPGA CLB, which can act as a 2-input look up table, a 4-bit Random Access Memory (RAM), or a 4-bit shift register. In Fig2.7a the inserted Trojan has been shown in red: the trigger condition is derived from the memory content of two consecutive RAM locations, and can harm the shift register functionality or the write

enable functionality of the memory block at run-time. The trigger condition can also be generated from the output of other CLBs, or alternatively can be derived from the output of other functional units. Fig.2.7b shows a Trojan instance inserted inside an embedded memory block in a commercial FPGA device. Similar to a CLB, an EMB is also capable of executing functionalities like shift register, FIFO etc. in addition to acting as Random Access Memory. The control circuitry shown in Fig2.7b decides between normal read operation and shift register operation inside the EMBs. The inserted logic or Trojan conditionally affects the shift operation inside a EMB by using adjacent memory contents and so can be triggered at run-time. It can be noted that the similar trigger conditions can also be effectively used to leak the contents of the memory to the output port. Such malfunctions can be achieved by changing the address bits of the memory blocks in different clock cycles and reading out the immediate next location of the memory in each and every cycle so as to obtain the complete memory contents stored in a particular EMB or a set of EMBs.

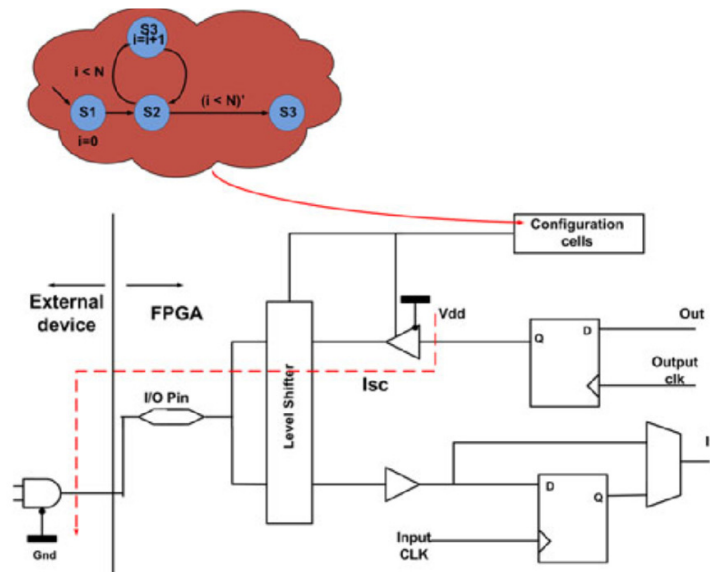


Fig 2.5 Programmable I/O block containing hardware Trojans to cause logical and electrical malfunction.

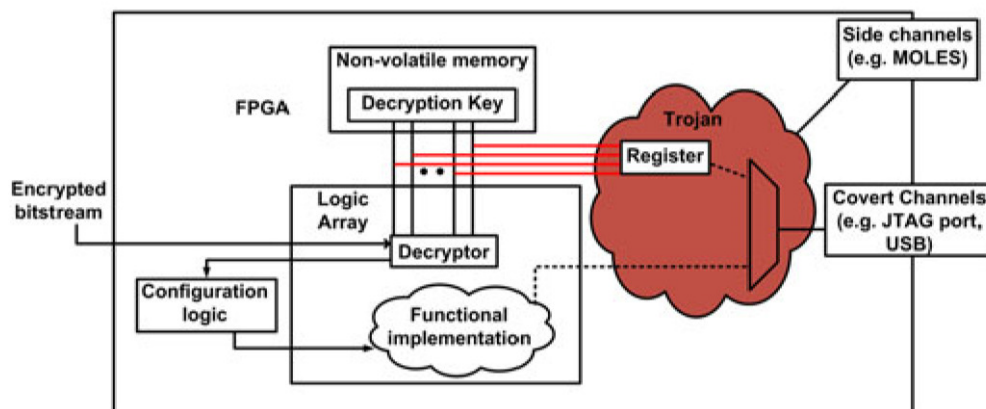


Fig 2.6 FPGA device with security features for bit-stream decryption.

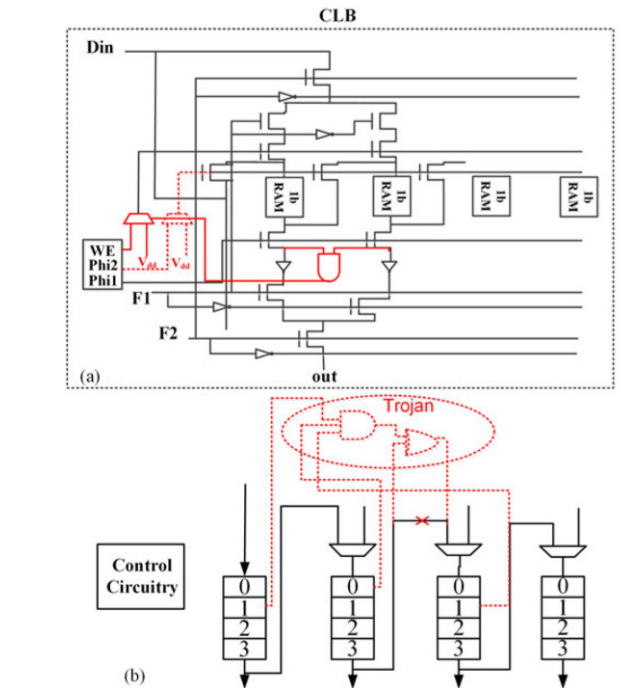


Fig: 2.7 Hardware Trojan inside: (a) a configurable logic block (CLB) and (b) an embedded memory block (EMB) of FPGA.

2.5.4 Types of Hardware Trojans:

There are different types of hardware Trojans like Combinational, Sequential and Hybrid Trojans as shown in fig2.8

Types of Hardware Trojan

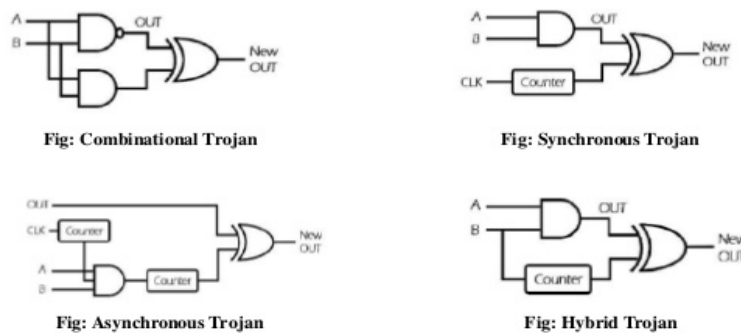


Fig2.8: Types of Hardware Trojans

2.5.5 Example Attack

The example attack is on a CHIP & PIN. The attack is CICA 2000 and it was performed in the United Kingdom. It's actually a protocol attack, but in order to perform it, you need customized hardware. The basic hardware used is the bank card. The bank card might have been stolen and the attacker didn't

know the pin to the card, but wants to buy something with the card without the pin. This attack can't be performed on an ATM because it uses a different protocol, it can only be performed on a handout terminals at a store.

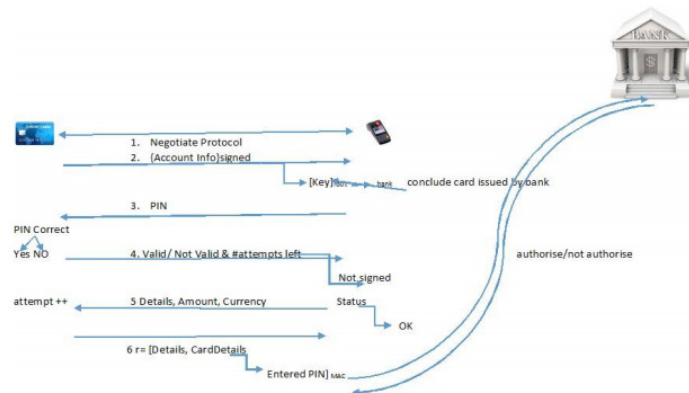


Fig2.9 Simplified diagram of Protocol used

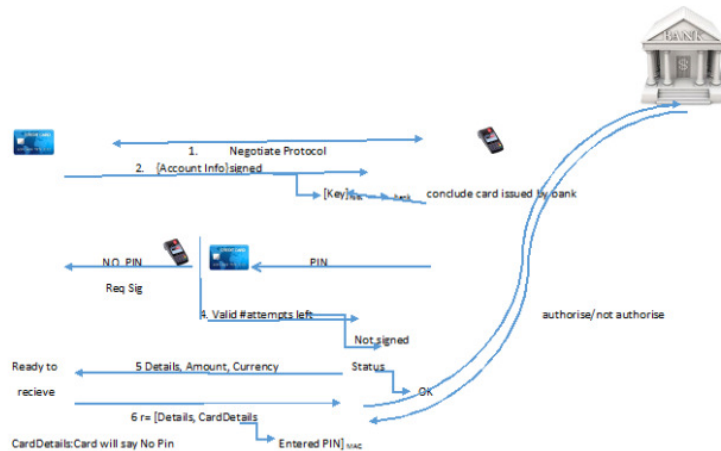


Fig2.10: The Attack

Issues of this attack

1. The Pin (Invalid not signed)
2. Details used in the protocol is not enough
3. Card Detail : Terminal can't parse only bank.

2.6 Real Life Implementation

The attacker would need the stolen card, a card reader, a laptop, a circuit (FPGA), wire and a fake card.

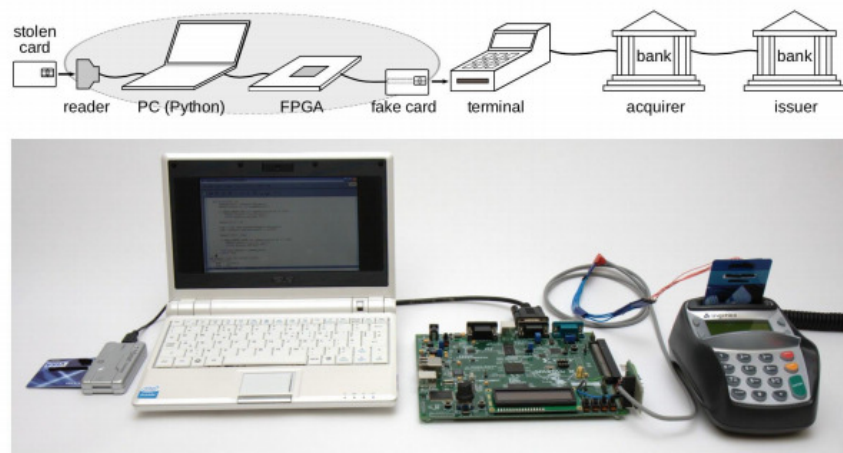


Fig2.11: Real life implementation of Chin & Pin attack

In the diagram above, the attacker would stick the card in a card reader and the card reader is placed in a computer, the computer is connected to a circuit like an FPGA. The attacker then have a wire that runs to a fake bank card that is in the terminal. The wire is attached to the circuit which was attached to the computer that the card reader which contains the real card is inserted. This is the Man-in-the-middle device, the stolen card would transfer all the information on the card to the fake card in the terminal. To perform this attack in real life, all the equipment would be placed in a bag pack and the wire would be passed inside the cloth and since by law the cashier is not supposed to touch the card, the fake card is inserted with the wire on the circuit and the attack is performed. This attacked happened in the UK but this kind of attack cannot happen in Canada.

CHAPTER 3

Introduction to FPGA

3.1 What is FPGA?

A field-programmable gate array (FPGA) is designed to be configured by a customer or a designer after manufacturing – hence the term "field-programmable". The FPGA configuration is generally specified using a hardware description language (HDL), similar to that used for an application-specific integrated circuit (ASIC). Circuit diagrams were previously used to specify the configuration, but this is increasingly rare due to the advent of electronic design automation tools.

FPGAs contain an array of programmable logic blocks, and a hierarchy of reconfigurable interconnects that allow the blocks to be "wired together", like many logic gates that can be inter-wired in different configurations. Logic blockscan be configured to perform complex combinational functions, or merely simple logic gates like AND and XOR. In most FPGAs, logic blocks also include memory elements, which may be simple flip-flops or more complete blocks of memory. Many FPGAs can be reprogrammed to implement different logic functions allowing flexible reconfigurable computing as performed in computer software.

3.2 THEORY

Altera was founded in 1983 and delivered the industry's first reprogrammable logic device in 1984 – the EP300 – which featured a quartz window in the package that allowed users to shine an ultra-violet lamp on the die to erase the EPROM cells that held the device configuration. In December 2015, Intel acquired Altera.

Xilinx co-founders Ross Freeman and Bernard Vonderschmitt invented the first commercially viable field-programmable gate array in 1985 – the XC2064. The XC2064 had programmable gates and programmable interconnects between gates, the beginnings of a new technology and market. The XC2064 had 64 configurable logic blocks (CLBs), with two three-input lookup tables (LUTs). More than 20 years later, Freeman was entered into the National Inventors Hall of Fame for his invention.

Notes:

1. Each 7 series FPGA slice contains four LUTs and eight flip-flops; only some slices can use their LUTs as distributed RAM or SRLs.
2. Each DSP slice contains a pre-adder, a 25 x 18 multiplier, an adder, and an accumulator.

3. Block RAMs are fundamentally 36 Kb in size; each block can also be used as two independent 18 Kb blocks.
4. Each CMT contains one MMCM and one PLL.
5. Artix-7 FPGA Interface Blocks for PCI Express support up to x4 Gen 2.
6. Does not include configuration Bank 0.
7. This number does not include GTP transceivers.

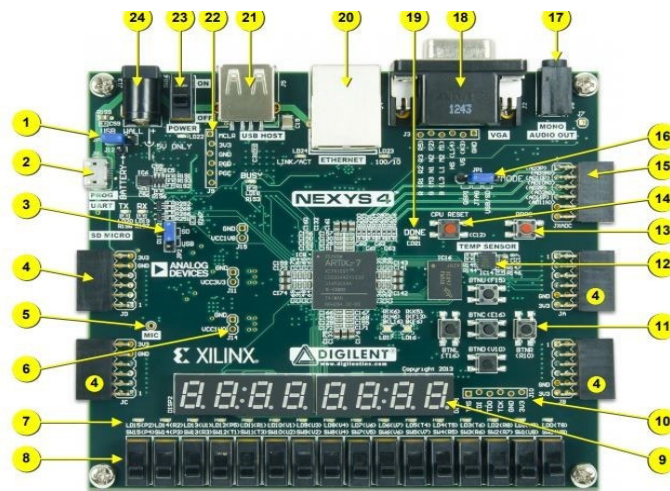


Fig3.1 Xilinx Artix 7 FPGA Trainer kit

Callout	Component Description	Callout	Component Description
1	Power select jumper and battery header	13	FPGA configuration reset button
2	Shared UART/ JTAG USB port	14	CPU reset button (for soft cores)
3	External configuration jumper (SD / USB)	15	Analog signal Pmod port (XADC)
4	Pmod port(s)	16	Programming mode jumper
5	Microphone	17	Audio connector
6	Power supply test point(s)	18	VGA connector
7	LEDs (16)	19	FPGA programming done LED
8	Slide switches	20	Ethernet connector
9	Eight digit 7-seg display	21	USB host connector
10	JTAG port for (optional) external cable	22	PIC24 programming port (factory use)
11	Five pushbuttons	23	Power switch
12	Temperature sensor	24	Power jack

Fig3.2 Port numbers of Artix7 FPGA Trainer kit

Table 3.1: Artix-7 FPGA Feature Summary by Device

		Configurable Logic											
		Slices(1)	Max Distributed	18	36	Max							
XC7A12	12,80	2,000	171	40	40	20	720	3	1	2	1	3	150
XC7A15	16,64	2,600	200	45	50	25	900	5	1	4	1	5	250
XC7A25	23,36	3,650	313	80	90	45	1,620	3	1	4	1	3	150
XC7A35	33,28	5,200	400	90	100	50	1,800	5	1	4	1	5	250
XC7A50	52,16	8,150	600	12	150	75	2,700	5	1	4	1	5	250
XC7A75	75,52	11,800	892	18	210	105	3,780	6	1	8	1	6	300
XC7A10	101,4	15,850	1,188	24	270	135	4,860	6	1	8	1	6	300
XC7A20	215,3	33,650	2,888	74	730	365	13,14	10	1	16	1	10	500

Table3. 2: Artix-7 FPGA Device-Package Combinations and Maximum I/Os

Packa	CPG23		CPG23		CSG32		CSG32		FTG25		SBG48		FGG4		FBG48		FGG6		FBG67		FFG11	
Size	10 x 10		10 x 10		15 x 15		15 x 15		17 x 17		19 x 19		23 x 23		23 x 23		27 x 27		27 x 27		35 x 35	
Ball																						
		I/O HR		I/O HR		I/O HR		I/O HR		I/O HR		I/O HR		I/O HR		I/O HR		I/O HR		I/O HR		I/O HR
XC7A 12T			2	112			2	150														
XC7A	2	106			0	210	4	150	0	170			4	250								
XC7A			2	112			4	150														
XC7A	2	106			0	210	4	150	0	170			4	250								
XC7A	2	106			0	210	4	150	0	170			4	250								
XC7A	2	106			0	210	4	150	0	170			4	250								
XC7A					0	210			0	170			4	285								
XC7A					0	210			0	170	4	285	4	285	8	300						
XC7A											4	285			4	285			8	400	16	500

1. All packages listed are Pb-free (SBG, FBG, FFG with exemption 15). Some packages are available in Pb option.
2. Devices in FGG484 and FBG484 are footprint compatible.
3. Devices in FGG676 and FBG676 are footprint compatible.
4. GTP transceivers in CP, CS, FT, and FG packages support data rates up to 6.25 Gb/s.

5. HR = High-range I/O with support for I/O voltage from 1.2V to 3.3V.

1. Power Supplies:

The Nexys 4 board can receive power from the Digilent USB-JTAG port (J6) or from an external power supply. Jumper JP3 (near the power jack) determines which source is used. All Nexys 4 power supplies can be turned on and off by a single logic-level power switch (SW16). A power-good LED (LD22), driven by the “power good” output of the ADP2118 supply, indicates that the supplies are turned on and operating normally.

2. FPGA Configuration:

After power-on, the Artix-7 FPGA must be configured (or programmed) before it can perform any functions. You can configure the FPGA in one of four ways:

1. A PC can use the Digilent USB-JTAG circuitry (portJ6, labeled “PROG”) to program the FPGA any time the power is on.
2. A file stored in the nonvolatile serial (SPI) flash device can be transferred to the FPGA using the SPI port.
3. A programming file can be transferred to the FPGA from a micro SD card.
4. A programming file can be transferred from a USB memory stick attached to the USB HID port.

The FPGA is made of SRAM (Volatile Memory) so the data configured inside FPGA lost at power Off state. FPGA Configuration is the process of loading the FPGA chip with Configuration data through external devices during power On state.

The Method of Configuring FPGA Can be divided into

- Master Mode
- Slave Mode
- JTAG Mode

Master Modes

In the Master Mode the Configuration data is stored in external nonvolatile memories such as SPI FLASH, Parallel FLASH, PROM and so on. During configuration process the data is loaded in the FPGA Configurable Logic Blocks to operate as a specific application. The configuration clock is provided by FPGA in Master Mode operation.

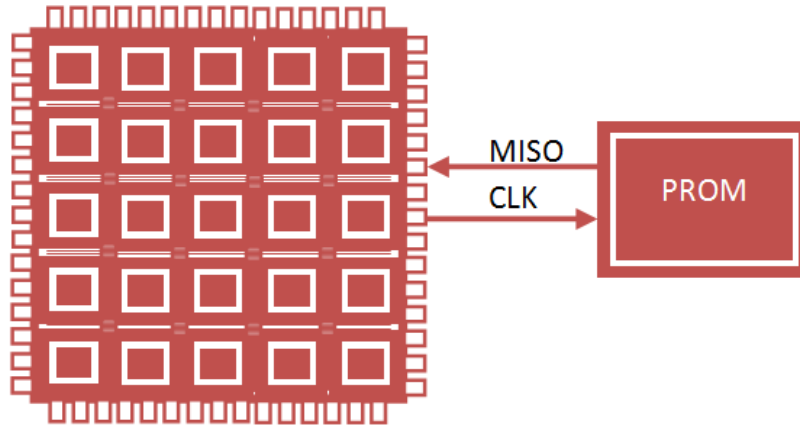


Fig 3.3 FPGA Configure through External PROM

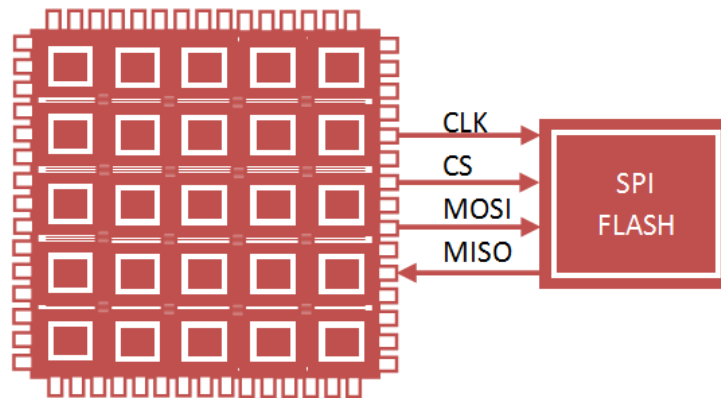


Fig 3.4 FPGA Configure through SPI FLASH Memory

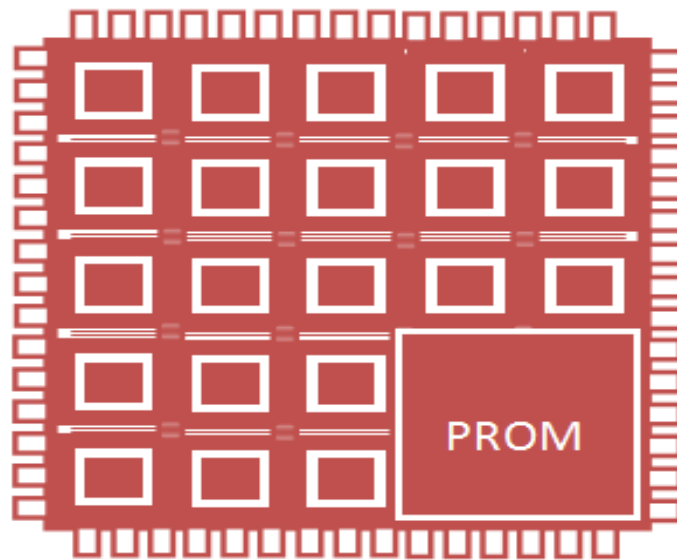


Fig 3.5 FPGA Configure through Internal FLASH

Slave Mode

In Slave Mode, The entire configuration Process is controlled by External device. Those External device may be of processor, Microcontroller, and so on. The configuration can perform serially or parallel method. The Clock input is supplied by the external device for Slave mode.

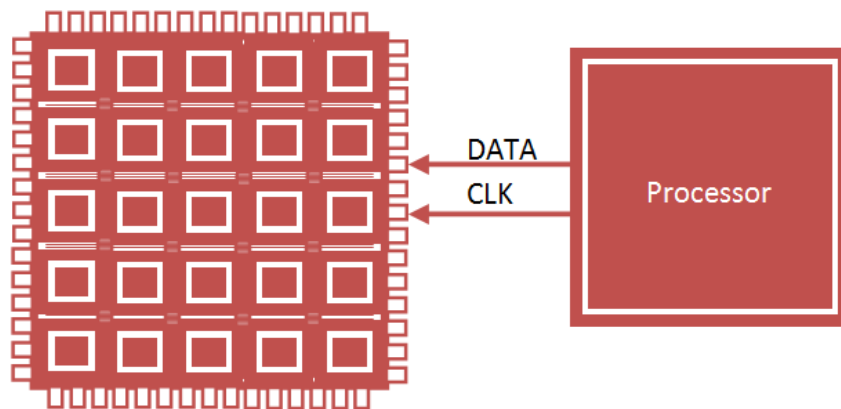


Fig 3.6: Slave Mode

JTAG Connection

The four-wire JTAG interface is common on board testers and debugging hardware. FPGA mainly uses JTAG interface for prototype download and debugging. JTAG consists of TCK, TMS, TDI and TDO lines for communication.

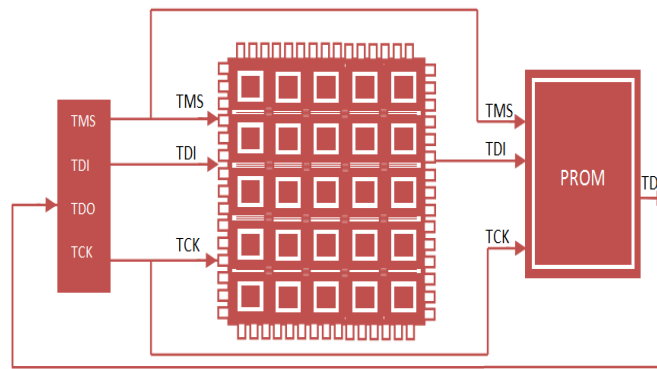


Fig 3.7: JTAG connection

3. Memory:

The Nexys 4 board contains two external memories: a 128Mbit Cellular RAM (pseudo-static DRAM) and a 128Mbit non-volatile serial Flash device. The Cellular RAM has an SRAM interface, and the serial Flash is on a dedicated quad-mode (x4) SPI bus. The connections and pin assignments between the FPGA and external memories

4. Ethernet PHY:

The Nexys 4 board includes an SMSC 10/100 Ethernet PHY (SMSC part number LAN8720A) paired with an RJ-45 Ethernet jack with integrated magnetic. The SMSC PHY uses the RMII interface and supports 10/100 Mb/s. At power-on reset, the PHY is set to the following defaults:

- RMII mode interface
- Auto-negotiation enabled, advertising all 10/100 mode capable
- PHY address=00001

3.3 Overview of FPGA

The Nexys4 DDR board is a complete, ready-to-use digital circuit development platform based on the latest Artix-7™ Field Programmable Gate Array (FPGA) from Xilinx®. With its large, high-capacity FPGA (Xilinx part number XC7A100T-1CSG324C), generous external memories, and collection of USB, Ethernet, and other ports, the Nexys4 DDR can host designs ranging from introductory combinational

circuits to powerful embedded processors. Several built-in peripherals, including an accelerometer, temperature sensor, MEMs digital microphone, a speaker amplifier, and several I/O devices allow the Nexys4 DDR to be used for a wide range of designs without needing any other components.

The Artix-7 FPGA is optimized for high performance logic, and offers more capacity, higher performance, and more resources than earlier designs. Artix-7 100T features include:

- 15,850 logic slices, each with four 6-input LUTs and 8 flip-flops
- 4,860 Kbits of fast block RAM
- Six clock management tiles, each with phase-locked loop (PLL)
- 240 DSP slices
- Internal clock speeds exceeding 450 MHz □ On-chip analog-to-digital converter (XADC)

The Nexys4 DDR also offers an improved collection of ports and peripherals, including:

- 16 user switches
- 16 user LEDs
- Two 4-digit 7-segment displays
- USB-UART Bridge
- Two tri-color LEDs
- Micro SD card connector
- 12-bit VGA output
- PWM audio output
- PDM microphone
- 3-axis accelerometer
- Temperature sensor
- 10/100 Ethernet PHY
- 128MiB DDR2
- Serial Flash
- Four Pmod ports
- Pmod for XADC signals
- Digilent USB-JTAG port for FPGA programming and communication
- USB HID Host for mice, keyboards and memory sticks

The Nexys4 DDR is compatible with Xilinx's new high-performance Vivado® Design Suite as well as the ISE® toolset, which includes ChipScope™ and EDK. Xilinx offers free WebPACK™ versions of these toolsets, so designs can be implemented at no additional cost. The Nexys4 DDR is not supported by the Digilent Adept Utility.

- Logic blocks
 - to implement combinational and sequential logic
- Interconnect
 - wires to connect inputs and outputs to logic blocks
- I/O blocks
 - special logic blocks at periphery of device for external connections
- Key questions:
 - How to make logic blocks programmable?
 - How to connect the wires?
 - After the chip has been fabricated?

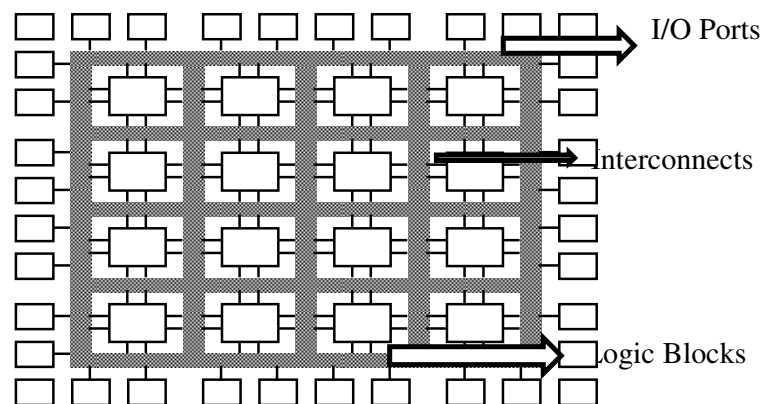


Fig3.8 Block diagram of FPGA

3.4 Programmability of FPGAs:

- User programmability of CPLDs and FPGAs is achieved via user-programmable switch technologies.
- For CPLDs, floating-gate transistors are used like EPROM or EEPROM. On the other hand, FPGAs normally use SRAM (static RAM) or anti fuse technology.
- Properties of the switches, such as, size, on-resistance, and capacitance dictate trade-offs in architecture.
- In SRAM based FPGAs, there is an SRAM bit corresponding to each of the programmable points within the device.

- When the device is powered-on or reset, it reads a configuration program from an off-chip memory and loads it into on-chip SRAM.
- The configuration program defines the logic function realized by individual logic blocks and interconnections.
- Devices using SRAM based switching can be reprogrammed easily by just changing the configuration program.
- FPGAs belonging to Xilinx, Plassey, Algotronix, Concurrent Logic, Toshiba, etc. are SRAM based.
- SRAM provides fast re-programmability at the cost of large area (at least five transistors for cell and one for switch).

Comparison between programming techniques

Name	Whether reprogrammable	Whether volatile	Technology
Fuse	No	No	Bi-polar
EPROM	Yes, out of circuit	No	UVC MOS
EEPROM	Yes, in circuit	No	EECMOS
SRAM	Yes, in circuit	Yes	CMOS
Anti FUSE	No	No	CMOS+

Table 3.3 Comparison between programming techniques

3.5 FPGA Logic Blocks

- There are wide variations in the logic block structure of FPGAs available from different vendors.
- They vary in number of inputs and outputs, amount of area consumed, complexity of logic functions that they can realize, total number of transistors needed, and so on.
- The logic blocks can broadly be classified into the following two categories – Fine Grain, Coarse Grain

Fine Grain Logic Block

- The block contains a few transistors that can be interconnected via programming.
- Cross point FPGA uses a single transistor pair for each Boolean variable in the logic block.

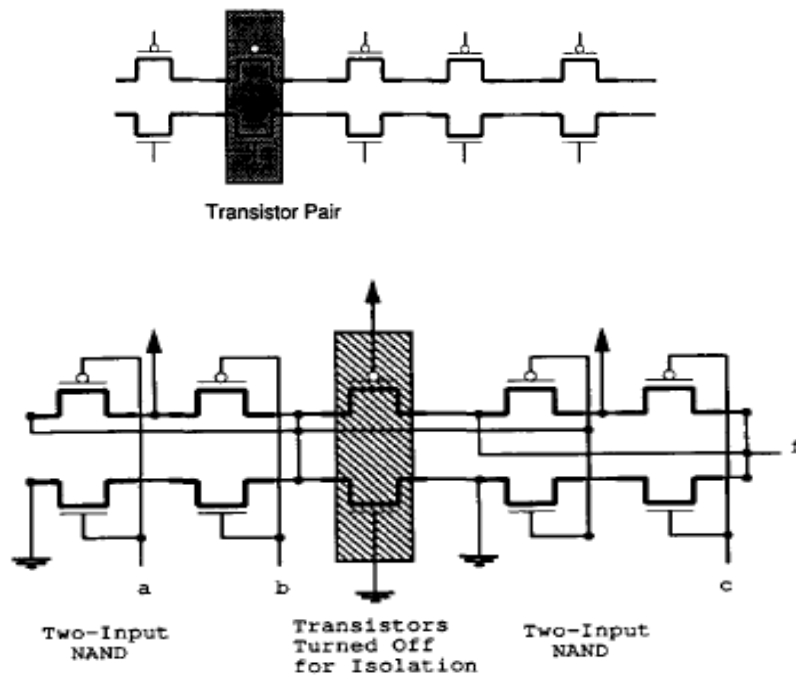


Fig3.9 Schematic diagram of FPGA Logic blocks

Coarse Grain Block – XC4000 from Xilinx

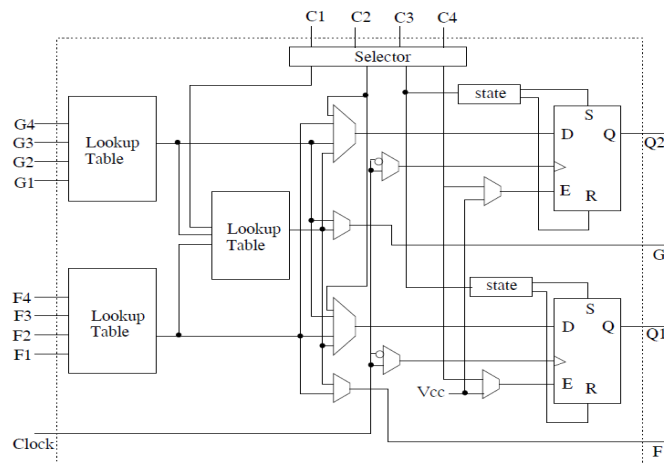


Fig3.10 Schematic diagram of coarse grain block

FPGA Design Flow

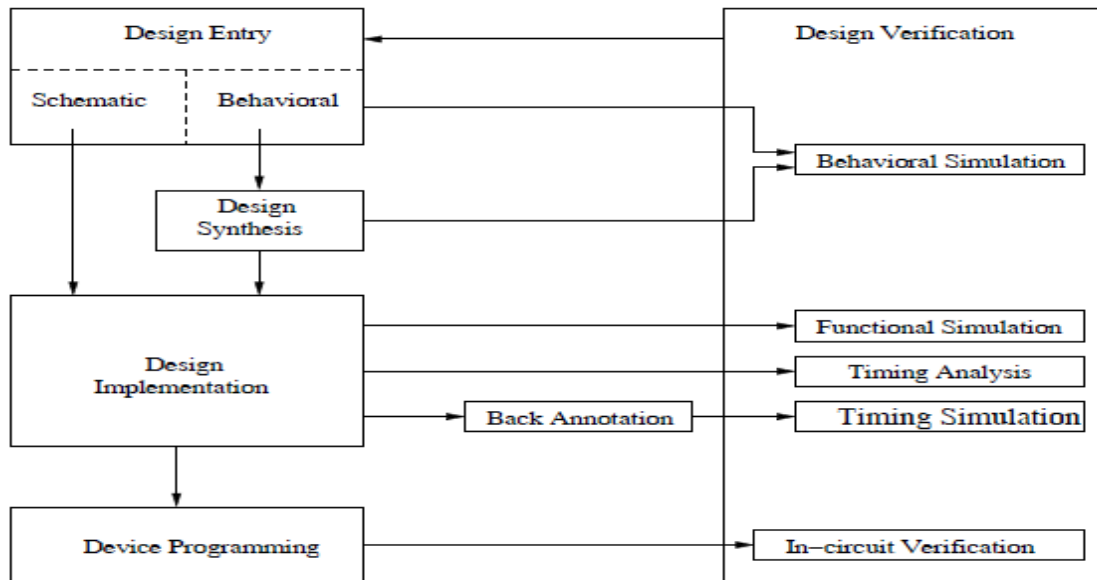


Fig3.11 Schematic diagram of FPGA Design flow

3.6 Modern FPGA's

- In addition to the basic blocks (such as, logic blocks, I/O blocks and interconnects), modern FPGAs have additional units that make the design process simpler and more efficient.
- The two major system components, difficult to implement in FPGAs are embedded memories and blocks for arithmetic calculations.
- Amongst the various calculations, multiplication is the most widely used one. Most of the modern FPGAs contain embedded logic blocks for multiplication and memories to hold data. DSP functionalities are highly facilitated by the availability of these.
- In many applications, FPGAs need to communicate with microprocessors. This has motivated many FPGA vendors to embed soft processor cores within FPGAs. This reduces the latency of communication between the microprocessor and the FPGA.

CHAPTER4

Verilog Hardware Description Language

4.1 What is HDL?

Hardware description language (HDL) is a specialized computer language used to program electronic and digital logic circuits. The structure, operation and design of the circuits are programmable using HDL. HDL includes a textual description consisting of operators, expressions, statements, inputs and outputs. Instead of generating a computer executable file, the HDL compilers provide a gate map. The gate map obtained is then downloaded to the programming device to check the operations of the desired circuit. The language helps to describe any digital circuit in the form of structural, behavioural and gate level and it is found to be an excellent programming language for FPGAs and CPLDs. The three common HDLs are Verilog, VHDL, and SystemC.

4.2 Importance of HDLs

HDLs have many advantages compared to traditional schematic-based design.

- Designs can be described at a very abstract level by use of HDLs. Designers can write their RTL description without choosing a specific fabrication technology. Logic synthesis tools can automatically convert the design to any fabrication technology. If a new technology emerges, designers do not need to redesign their circuit. They simply input the RTL description to the logic synthesis tool and create a new gate-level net-list, using the new fabrication technology. The logic synthesis tool will optimize the circuit in area and timing for the new technology.
- By describing designs in HDLs, functional verification of the design can be done early in the design cycle. Since designers work at the RTL level, they can optimize and modify the RTL description until it meets the desired functionality. Most design bugs are eliminated at this point. This cuts down design cycle time significantly because the probability of hitting a functional bug at a later time in the gate-level netlist or physical layout is minimized.
- Designing with HDLs is analogous to computer programming. A textual description with comments is an easier way to develop and debug circuits. This also provides a concise representation of the design, compared to gate-level schematics. Gate-level schematics are almost incomprehensible for very complex designs.

4.3 Introduction to Verilog HDL

Verilog HDL is one of the two most common Hardware Description Languages (HDL) used by integrated circuit (IC) designers. The other one is VHDL. HDL's allows the design to be simulated earlier in the design cycle in order to correct errors or experiment with different architectures. Designs described in HDL are technology-independent, easy to design and debug, and are usually more readable than schematics, particularly for large circuits.

Verilog can be used to describe designs at four levels of abstraction:

- (i) Algorithmic level (much like c code with if, case and loop statements).
- (ii) Register transfer level (RTL uses registers connected by Boolean equations).
- (iii) Gate level (interconnected AND, NOR etc.).

(iv) Switch level (the switches are MOS transistors inside gates).

The language also defines constructs that can be used to control the input and output of simulation.

Verilog has a variety of constructs as part of it. All are aimed at providing a functionally tested and a verified design description for the target FPGA or ASIC. The language has a dual function – one fulfilling the need for a design description and the other fulfilling the need for verifying the design for functionality and timing constraints like propagation delay, critical path delay, slack, setup, and hold times.

Verilog as an HDL has been introduced here and its overall structure explained. A widely used development tool for simulation and synthesis has been introduced; the brief procedural explanation provided suffices to try out the Examples and Exercises in the text.

4.4 Module

Any Verilog program begins with a keyword– called a “module.” A module is the name given to any system considering it as a black box with input and output terminals as shown in Figure 4.1. The terminals of the module are referred to as ‘ports’. The ports attached to a module can be of three types:

- **input ports** through which one gets entry into the module; they signify the input signal terminals of the module.
- **output ports** through which one exits the module; these signify the output signal terminals of the module.
- **inout ports:** These represent ports through which one gets entry into the module or exits the module; These are terminals through which signals are input to the module sometimes; at some other times signals are output from the module through these.

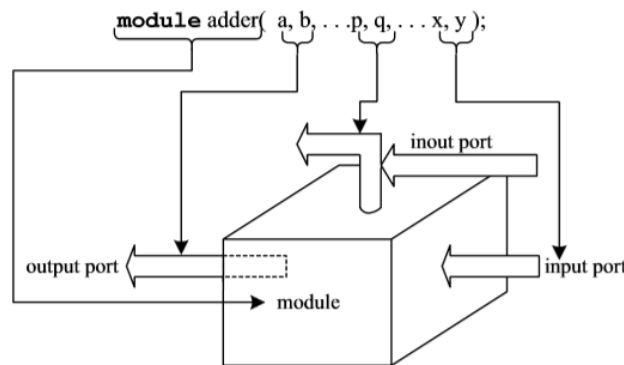


Fig 4.1: Representation of a module as black box with its ports.

4.5 Tokens of Verilog

The basic lexical conventions used by Verilog HDL are similar to those in the C programming language. Verilog contains a stream of tokens. Tokens can be comments, delimiters, numbers, strings, identifiers, and keywords.

4.5.1 Case Sensitivity

Verilog is a case-sensitive language like C. Thus sense, Sense, SENSE, sENse,... etc., are all treated as different entities / quantities in Verilog.

4.5.2 Keywords

The keywords define the language constructs. A keyword signifies an activity to be carried out, initiated, or terminated. As such, a programmer cannot use a keyword for any purpose other than that it is intended for. All keywords in Verilog are in small letters and require to be used as such (since Verilog is a case-sensitive language). All keywords appear in the text in New Courier Bold-type letters.

Examples

module <- signifies the beginning of a module definition.

endmodule <- signifies the end of a module definition.

begin<- signifies the beginning of a block of statements. **end**<- signifies the end of a block of statements.

if <- signifies a conditional activity to be checked

while<- signifies a conditional activity to be carried out.

4.5.3 Operators

Operators are of three types: unary, binary, and ternary. Unary operators precede the operand. Binary operators appear between two operands. Ternary operators have two separate operators that separate three operands.

Examples

a = ~ b; // ~ is a unary operator. b is the operand

a = b && c; // && is a binary operator. b and c are operands

a = b ? c : d; // ?: is a ternary operator. b, c and d are operands

4.5.4 Data Types

There are two groups of types, "**net data types**" and "**variable data types**."

An identifier of "**net data type**" means that it must be driven. The value changes when the driver changes value. These identifiers basically represent wires and are used to connect components.

"net data types" are: **wire**, **supply0**, **supply1**, **tri**, **triand**, **trior**, **tri0**, **tri1**, **wand**, **wor** "net data types" can have strength modifiers: **supply0**, **supply1**, **strong0**, **strong1**, **pull0**, **pull1**, **weak0**, **weak1**, **highz0**, **highz1**, **small**, **medium**, **large**.

Some "net data types" can take modifiers: **signed**, **vectored**, scalar.

An identifier of "**variable data type**" means that it changes value upon assignment and holds its value until another assignment. This is a traditional programming language variable and is used in sequential statements.

"Variable data types" are: **integer**, **real**, **realtime**, **reg**, **time**.

integer is typically a 32 bit twos complement integer.

real is typically a 64 bit IEEE floating point number.

real time is of type **real** used for storing time as a floating point value.

reg is by default a one bit unsigned value.

The **reg** variable data type may have a modifier **signed**, and may have many bits by using the vector modifier **msb:lsb**.

Scalars and Vectors

Entities representing single bits — whether the bit is stored, changed, or transferred — are called “scalars.” Often multiple lines carry signals in a cluster — like data bus, address bus, and so on. Similarly, a group of regs stores a value, which may be assigned, changed, and handled together. The collection here is treated as a “vector.” Figure 4.2 illustrates the difference between a scalar and a vector. **wr** and **rd** are two scalar nets connecting two circuit blocks **circuit1** and **circuit2**. **b** is a 4-bit-wide vector net connecting the same two blocks. **[b0]**, **[b1]**, **[b2]**, and **[b3]** are the individual bits of vector **b**. They are “part vectors.”

A vector **reg** or net is declared at the outset in a Verilog program and hence treated as such. The range of a vector is specified by a set of 2 digits (or expressions evaluating to a digit) with a colon in between the two. The combination is enclosed [within square brackets.

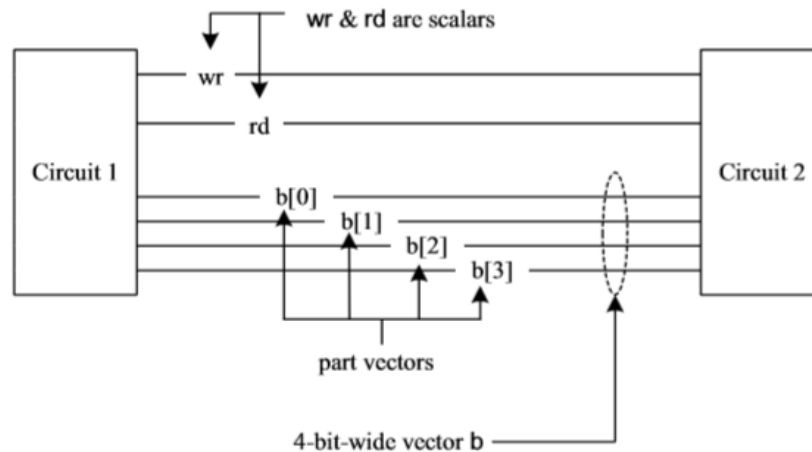


Fig 4.2: Illustration of Scalars and Vectors

Examples:

wire [3:0] a; /* a is a four bit vector of net type; the bits are designated as [a3], [a2], [a1] and [a0]. */

reg [2:0] b; /* b is a three bit vector of reg type; the bits are designated as [b2], [b1] and [b0]. */

reg [4:2] c; /* c is a three bit vector of reg type; the bits are designated as [c4], [c3] and [c2]. */

wire [-2:2] d ; */* d is a 5 bit vector with individual bits designated as [d-2], [d-1], [d0], [d1] and [d2]. */*

Whenever a range is not specified for a net or a reg, the same is treated as a scalar – a single bit quantity. In the range specification of a vector the most significant bit and the least significant bit can be assigned specific integer values. These can also be expressions evaluating to integer constants – positive or negative. Normally vectors – nets or regs – are treated as unsigned quantities. They have to be specifically declared as “signed” if so desired.

Examples

wire signed [4:0] num; *// num is a vector in the range -16 to +15.*

reg signed [3:0] num_1; *// num_1 is a vector in the range -8 to +7.*

4.5.5 Comments

Comments can be inserted in the code for readability and documentation. There are two ways to write comments. A one-line comment starts with `"/"`. Verilog skips from that point to the end of line. A multiple-line comment starts with `"/**"` and ends with `"/**"`. Multiple-line comments cannot be nested. However, one-line comments can be embedded in multiple-line comments.

`a = b && c;` *// This is a one-line comment*

/ This is a multiple line comment */*

/ This is /* an illegal */ comment */*

/ This is //a legal comment */*

4.5.6 Number Specification

There are two types of number specification in Verilog they are sized and unsized.

Sized numbers

Sized numbers are represented as `<size> '<base format> <number>`.

`<size>` is written only in decimal and specifies the number of bits in the number. Legal base formats are decimal ('d or 'D), hexadecimal ('h or 'H), binary ('b or 'B) and octal ('o or 'O). The number is specified as consecutive digits from 0, 1, 2, 3, 4, 5, 6, 7, 8, 9, a, b, c, d, e, f. Only a subset of these digits is legal for a particular base. Uppercase letters are legal for number specification.

`4'b1111` *// This is a 4-bit binary number*

`12'habc` *// This is a 12-bit hexadecimal number*

`16'd255` *// This is a 16-bit decimal number.*

Un-sized numbers

Numbers that are specified without a <base format> specification are decimal numbers by default. Numbers that are written without a <size> specification have a default number of bits that is simulator- and machine-specific (must be at least 32).

23456 // This is a 32-bit decimal number by default

'hc3 // This is a 32-bit hexadecimal number

'o21 // This is a 32-bit octal number

X or Z values

Verilog has two symbols for unknown and high impedance values. These values are very important for modeling real circuits. An unknown value is denoted by an x. A high impedance value is denoted by z.

12'h13x // This is a 12-bit hex number; 4 least significant bits unknown

6'hx // This is a 6-bit hex number

32'bz // This is a 32-bit high impedance number

An X or Z sets four bits for a number in the hexadecimal base, three bits for a number in the octal base, and one bit for a number in the binary base. If the most significant bit of a number is 0, X, or Z, the number is automatically extended to fill the most significant bits, respectively, with 0, X, or Z. This makes it easy to assign X or Z to whole vector. If the most significant digit is 1, then it is also zero extended.

Negative numbers

Negative numbers can be specified by putting a minus sign before the size for a constant number. Size constants are always positive. It is illegal to have a minus sign between <base format> and <number>. An optional signed specifier can be added for signed arithmetic.

-6'd3 // 8-bit negative number stored as 2's complement of 3

-6'sd3 // Used for performing signed integer math

4'd-2 // Illegal specification

4.6 Module Declaration:

Modules are the building blocks of Verilog designs. A module can be an element or a collection of lower-level design blocks. A module provides the necessary functionality to the higher-level block through its port interface (inputs and outputs), but hides the internal implementation. Module interface refers, how module communicates with external world. This communication is possible through different ports such as input, output and bi-directional (inout) ports. Design functionality is implemented inside module, after port declaration. In Verilog, a module is declared by the keyword module. A corresponding keyword endmodule must appear at the end of the module definition. Each module must have a module_name, which is the identifier for the module, and a port list, which describes the input and output terminals of the module. Design functionality is implemented inside module, after port declaration. The design functionality implementation part is represented as “body” here.

Syntax

```
module module_name(port_list);  
  
inputmsb:lsb] input_port_list;  
  
outputmsb:lsb] output_port_list;  
  
inoutmsb:lsb] inout_port_list;  
  
.....statements.....  
  
endmodule
```

NOTE: All module declarations must begin with the **module** (or **macro-module**) keyword and end with the **endmodule** keyword. After the module declaration, an identifier is required. A ports list is an option. After that, ports declaration is given with declarations of the direction of ports and the optionally type. The body of module can be any of the following:

- Any declaration including parameter, function, task, event or any variable declaration.
- Continuous assignment.
- Gate, UDP or module instantiation.
- Specify block.
- Initial block
- Always block.

If there is no instantiation inside the module, it will be treated as a top-level module.

Example

```
module module_1(a, b, c) ;  
  
parameter size = 3 ;  
  
input size : 0] a, b ;  
  
output size : 0] c;  
  
assign c = a & b;  
  
endmodule
```

4.6.1 Module Instantiation

Modules can be instantiated from within other modules. When a module is instantiated, connections to the ports of the module must be specified. There are two ways to make port connections. One is connection by name, in which variables connected to each of module inputs or outputs are specified in a set of parenthesis following the name of the ports. In this method order of connections is not significant (from *Example 1*).

The second method is called ordered connection. In this method the order of the ports must match the order appearing in the instantiated module (from *Example 2*).

When ports are connected by name it is illegal to leave any ports unconnected. This may occur when ports are connected by order (from *Example 3*).

What happens if you leave a port unconnected depends on the type of the port. If you are connecting net type ports, unconnected bits are driven with high impedance. In other cases, bits are driven with unknown values.

Module instantiations can create an array of instances. To create these instances, range specifications have to be declared after the module name. The array of instances can save you time in writing code and provide a way to enrich your readability (from *Example 4*).

Example 1

```
module dff (clk, d, q);  
input clk, d;  
output q;  
reg q;  
always @(posedge clk) q = d;  
endmodule  
  
module top;  
reg data, clock;  
wire q_out, net_1;  
  dff inst_1 (.d(data), .q(net_1), .clk(clock));  
  dff inst_2 (.clk(clock), .d(net_1), .q(q_out));  
endmodule
```

In the top module there are two instantiations of the 'dff' module. In both cases port connections are done by name, so the port order is insignificant. The first port is input port 'd', the second is output 'q' and the last is the clock in the 'inst_1'. In the dff module the order of ports is different than either of the two instantiations.

Example 2

```
module dff (clk, d, q);  
input clk, d;  
output q;  
reg q;  
always @(posedge clk) q = d;  
endmodule  
  
module top;  
reg data, clock;  
wire q_out, net_1;  
  dff inst_1 (clock, data, net_1);  
  dff inst_2 (clock, net_1, q_out);  
endmodule
```

Example 3

```
dff inst_1 (clock, , net_1);
```

Second port is unconnected and has the value Z because it is of the net type.

Example 4

```
module my_module (a, b, c);  
input a, b;  
output c;  
assign c = a & b ;  
endmodule
```

```
module top (a, b, c) ;  
input 3:0] a, b;  
output 3:0] c;  
    my_module inst 3:0] (a, b, c);  
endmodule
```

4.7 Flowchart of Verilog Code

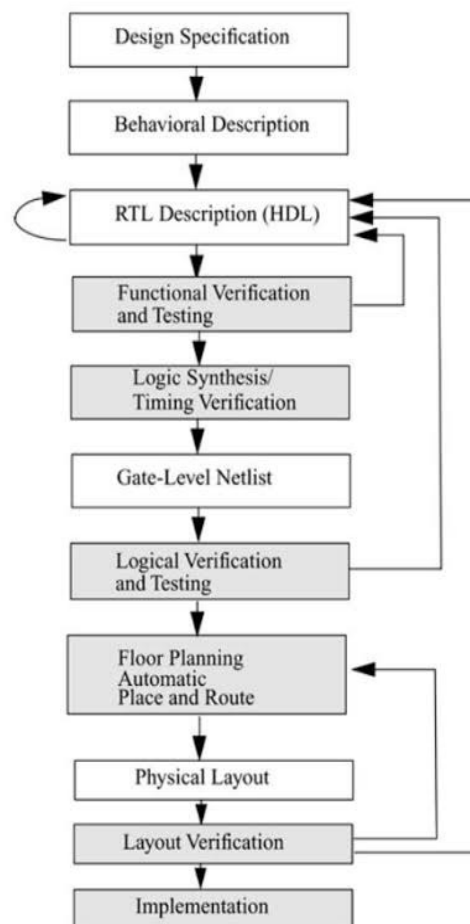


Fig 4.3: Flowchart representation of Verilog Code

Verilog has four levels of Modeling:

- 1) The switch level Modeling.
- 2) Gate-level Modeling.
- 3) The Data-Flow level.
- 4) The Behavioural Procedure

1) Switch level Modeling:

A circuit is defined by explicitly showing how to construct it using transistors like pmos and nmos, pre-defined modules.

Example:

```
module inverter (out, in);  
  
    output out;  
  
    input in;  
  
    supply0gnd;  
  
    supply1 vdd;  
  
    nmosx1 (out, in, gnd);  
  
    pmosx2(out, in, vdd);  
  
endmodule
```

2) Gate level modeling:

A circuit is defined by explicitly showing how to construct it using logic gates, predefined modules, and the connections between them. In this first we think of our circuit as a box or module which is encapsulated from its outer environment, in such a way that its only communication with the outer environment is through input and output ports. We then set out to describe the structure within the module by explicitly describing its gates and sub modules, and how they connect with one another as well as to the module ports. In other words, structural modeling is used to draw a schematic diagram for the circuit. As an example, consider the full-adder below.

Example:

```
module fulladder (a, b, sum, Cout);
```

```

Input a, b;

output sum, Cout;

xor x1(a, b, y);

xor x2(a, b, y);

endmodule

```

3) Data-flow modeling:

Dataflow modelling uses Boolean expressions and operators. In this we use assign statement.

Example :

```

module fulladder (a, b, sum, Cout);

input a, b;

output sum, Cout;

assign sum=a^b;

assign Cout =a^b;

endmodule

```

4) Behavioural modeling:

It is higher level of modeling where behaviour of logic is modelled. Verilog behavioural code is inside procedure blocks, but there is an exception: some behavioural code also exist outside procedure blocks.

There are two types of procedural blocks in Verilog :

Initial: initial blocks execute only once at time zero(start execution at time zero)

Always: always blocks loop to execute over and over again; in other words, as the name suggests, it executes always.

An always statement executes repeatedly, it starts and its execution at 0ns

Syntax:

```

always@ sensitivitylist)

```

```
begin

--Procedural statements--

end
```

Example:

```
module fulladder (a, b, clk, sum);

input a, b, clk;

output sum;

always@ (posedgeclk)

begin

sum =a+b;

endmodule
```

4.8 Software Tools Used

Xilinx Vivado 2019.2

Few important terminologies are tasks and functions. They are described below.

Tasks:

Tasks are used in all programming languages, generally known as procedures or subroutines. The lines of code are enclosed in task....end task brackets. Data is passed to the task, the processing done, and the result returned. They have to be specifically called, with data ins and outs, rather than just wired in to the general netlist. Included in the main body of code, they can be called many times, reducing code repetition.

- tasks are defined in the module in which they are used. It is possible to define a task in a separate file and use the compile directive 'include to include the task in the file which instantiates the task.
- tasks can include timing delays, like posedge, negedge, # delay and wait.
- tasks can have any number of inputs and outputs.
- The variables declared within the task are local to that task. The order of declaration within the task defines how the variables passed to the task by the caller are used.
- tasks can take, drive and source global variables, when no local variables are used. When local variables are used, basically output is assigned only at the end of task execution.
- tasks can call another task or function.
- tasks can be used for modeling both combinational and sequential logic.
- A task must be specifically called with a statement; it cannot be used within an expression as a function can.

Functions:

A Verilog HDL function is the same as a task, with very little differences, like function cannot drive more than one output, can not contain delays.

- functions are defined in the module in which they are used. It is possible to define functions in separate files and use compile directive 'include to include the function in the file which instantiates the task.
- functions can not include timing delays, like posedge, negedge, # delay, which means that functions should be executed in "zero" time delay.
- functions can have any number of inputs but only one output.
- The variables declared within the function are local to that function. The order of declaration within the function defines how the variables passed to the function by the caller are used.
- functions can take, drive, and source global variables, when no local variables are used. When local variables are used, basically output is assigned only at the end of function execution.
- functions can be used for modeling combinational logic.
- functions can call other functions, but cannot call tasks.

Steps to be followed to Create a Project in Xilinx Vivado 2019.2

- First open Xilinx Vivado 2019.2 then the Fig 4.4 appears on the screen.
- Click on **Create Project** to create a new project.

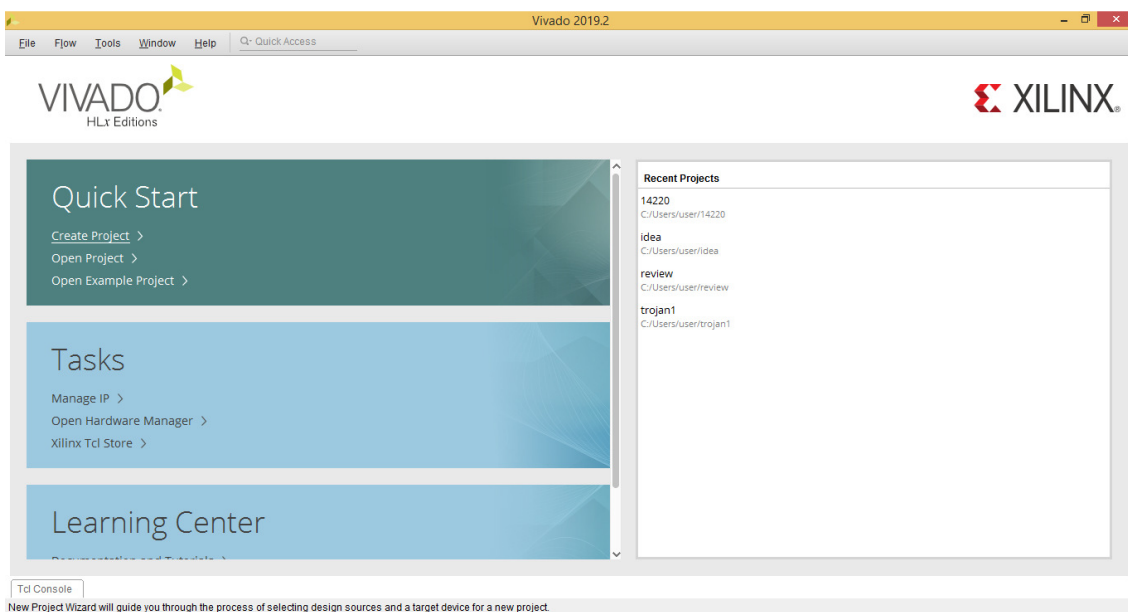


Fig 4.4 Opening window of Xilinx 2019.2

- Now a new window appears as shown in Fig 4.5

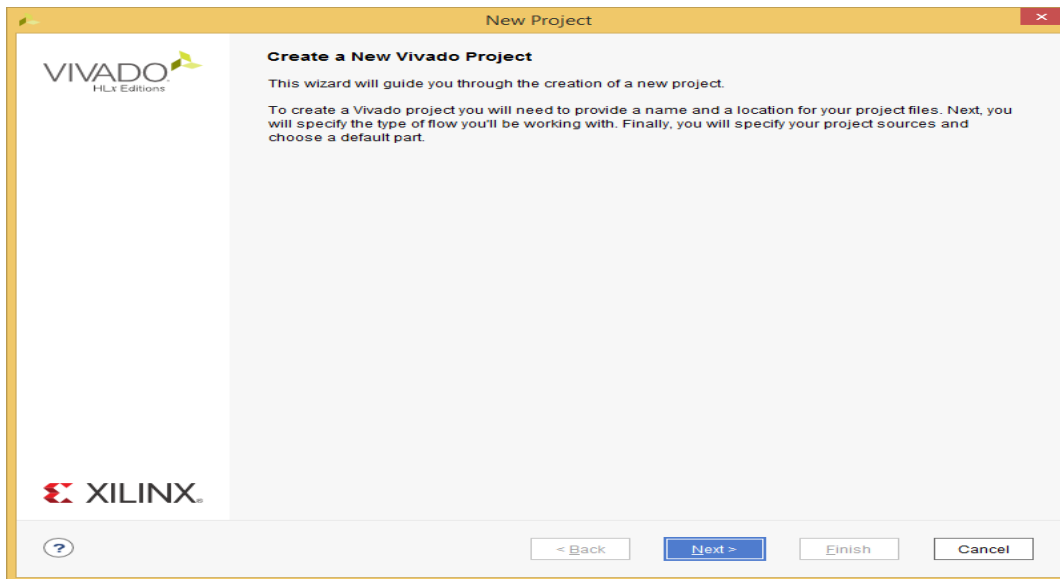


Fig:4.5: Create project window

- Click on **NEXT** to move further fig 4.6 shows the name and location of the project.

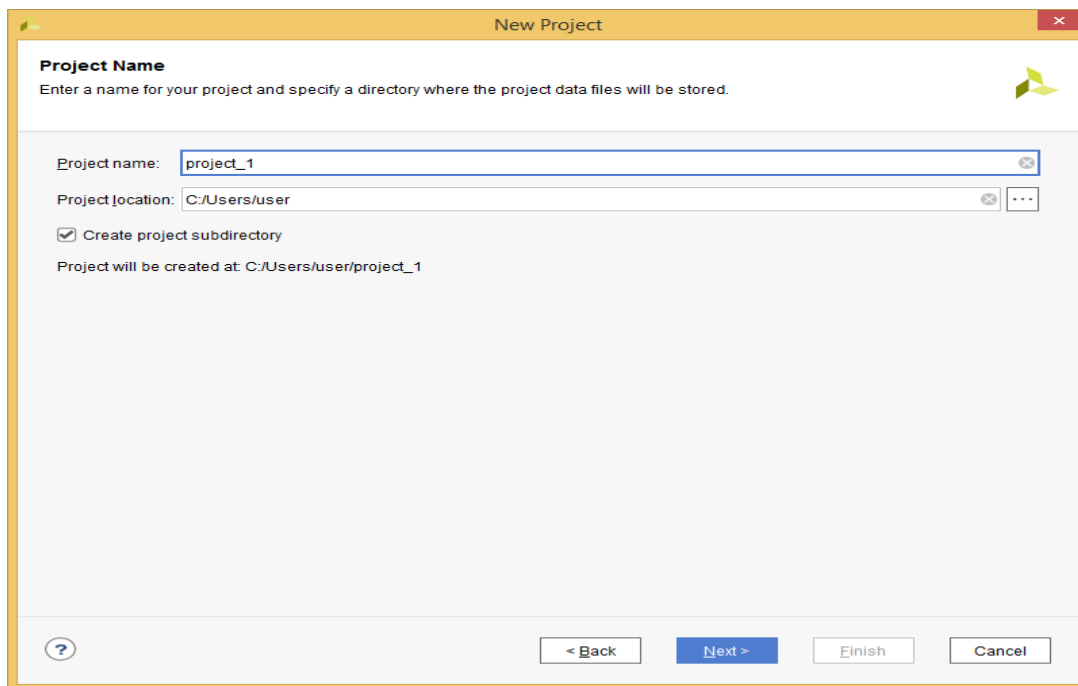


Fig 4.6: Name and Location entry for project

- Select Type of the project fig 4.7 shows type of project.

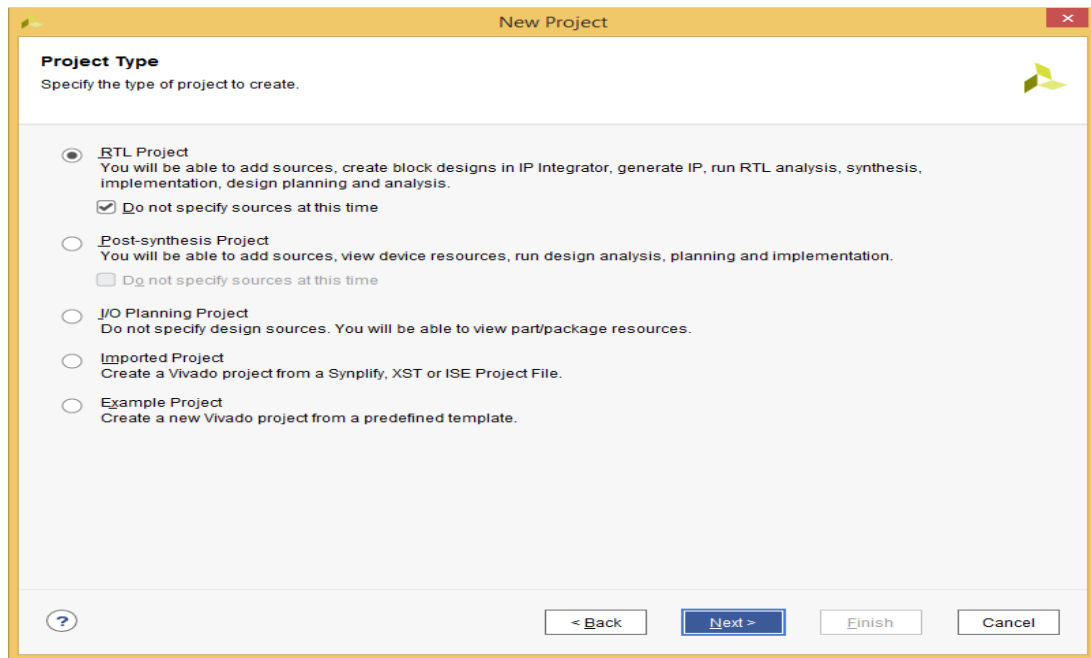


Fig 4.7 Selecting type of the project

- Now select specifications for the required project. Fig 4.8 shows Specifications of the project.

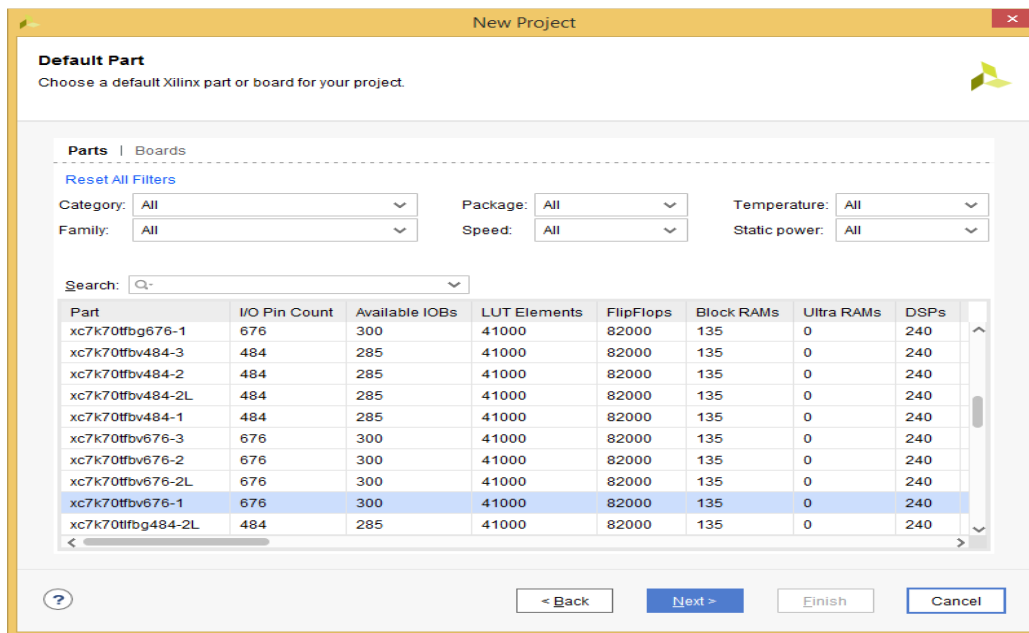


Fig 4.8.: Specifications window for the project

- Fig 4.9 shows summary of the project.

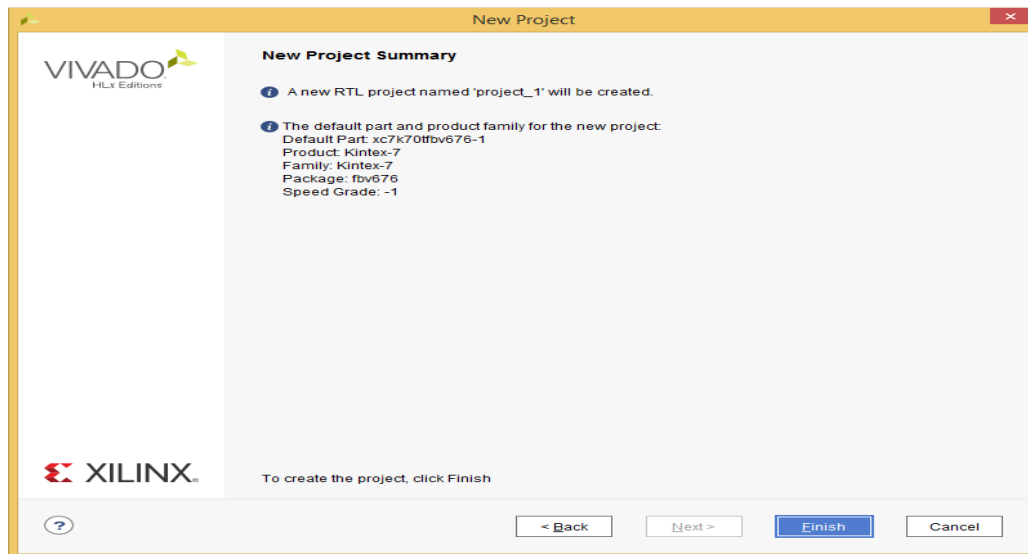


Fig 4.9: Summary window of the project

- Sources can be added upon clicking Add Sources. Fig 4.10 appears after following the above steps.

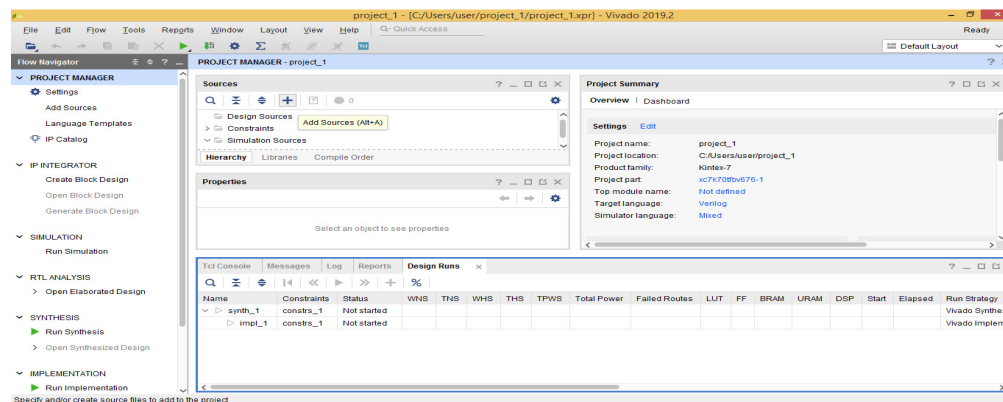


Fig 4.10: Add sources

- Sources are added to the project by clicking on add or create design and a Verilog file name.

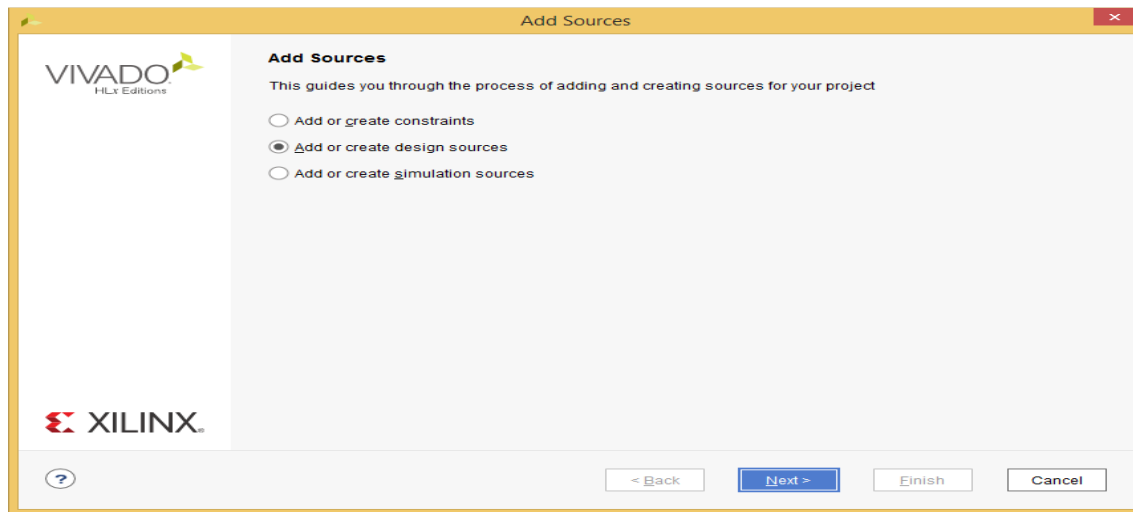


Fig4.11: Creating source

- Create Verilog files to write code. Fig 4.8.9 and 4.8.10 deals with creating a Verilog file for writing the code.

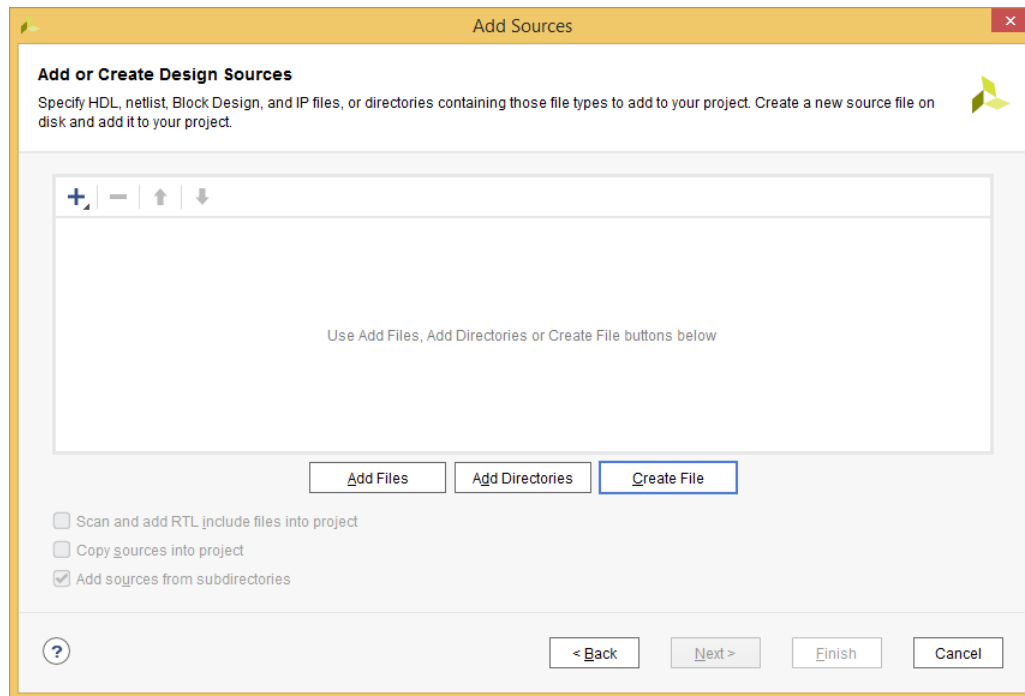


Fig 4.12: Creating file window

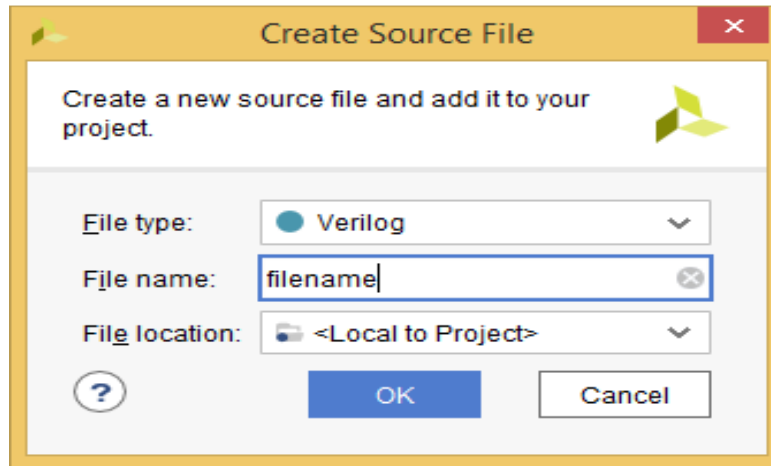


Fig 4.13: Filename creation window

- Sources will be shown in a new window as in fig 4.8.11

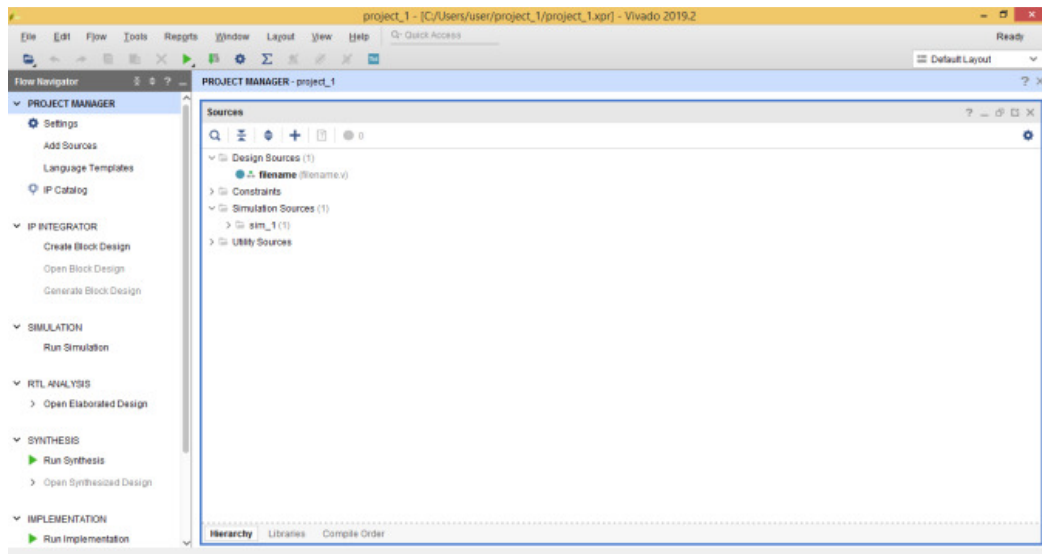


Fig 4.14: Sources window

- Simulation, Synthesis and implementation can be done by using the side bar options as shown in fig 4.8.12

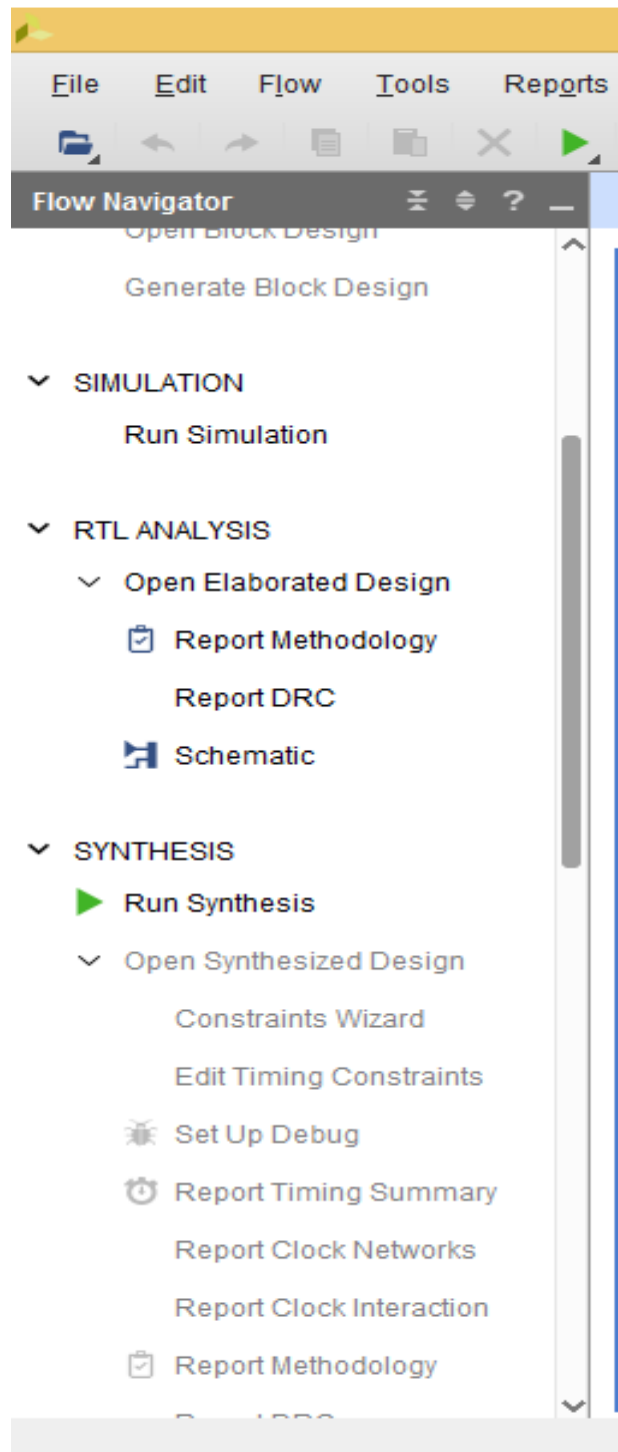


Fig 4.15 Sidebar for performing required process.

CHAPTER 5

Hardware Trojan Insertion and Detection

5.1 Hardware Trojan

A hardware trojan can be described as a malicious alteration or inclusion to an integrated circuit (IC) that will either alter its intended function or cause it to perform an additional malicious function. These malicious inclusions or alterations are generally programmed to activate only under a specific set of circumstances created by an attacker and are extremely hard to detect when in their dormant state. As technology advances, so does the demand for IC boards leaving many technology companies without the resources to produce secure enough ICs to meet current demands. This has pushed companies into the 'fabless' trend prevalent in today's semi-conductor industry, where companies are no longer attempting to produce the goods in their own factories, but instead are outsourcing the process to cheaper factories abroad. This growth brings with it a significant rise in the level of threat posed by hardware trojans, a threat that directly affects all companies concerned with products that utilise ICs. This encompasses many different industries, including the military and telecommunications companies, and can potentially affect billions of devices from mobile phones and computers to military grade aviation and detection devices, particularly at a time when wireless devices are being introduced as links in critical infrastructure, compounding trust and security issues even further.

5.2 Algorithm for Trojan Detection

The following steps are to be followed for detecting an hardware trojan.

1. Get the circuit under test
2. Retrieve the path delays and other parameters
3. $i=1$
4. Retrieve i th path
5. Give test vector to circuit under test
6. Measure the path delay and other parameters
7. If delay equal to golden circuit's delay then go to 'a' else go to 'b'.
 - a. $i=i+1$
if i =number of available paths then go to 'f1' else go to **step-4**.
f1. Circuit is trojan free.
 - b. *Circuit is affected.*

5.3 Hardware Trojan Detection Methodologies

In this section, we discuss some detection methods for detecting FPGA hardware Trojans. These methods must be used by the FPGA vendors when they receive the chips from the off-shore foundry. We assume that the testing facility used by a FPGA vendor is secure, eliminating the possibility that an

adversary in the testing facility could intentionally not detect malicious alterations. Detection methods can be classified into three categories: Visual detection techniques, logic testing, side-channel analysis and using ring oscillators.

5.3.1 Visual Detection Methods

This class of detection methods uses imaging to identify any malicious insertions in the chip. These techniques include using X-ray imaging, scanning optical microscopy (SOM), scanning electron microscopy (SEM) , and Pico-second imaging circuit analysis (PICA), among others. These methods, however, can be expensive in cost and analysis time. Moreover, these techniques suffer from lack of resolution to decipher logic/ transistor/interconnect level information, primarily due to the obstruction by the stack of metal layers in modern FPGAs. With increasing device density due to technology scaling, effectiveness of the imaging techniques is expected to reduce significantly. Partial de-layering of ICs appears more effective; however, it may in turn render an FPGA non-functional. Due to the above limitations, imaging analysis may not be a viable Trojan detection technique. The following shown fig is an example of differentiating active, sleep and trojan free circuit.

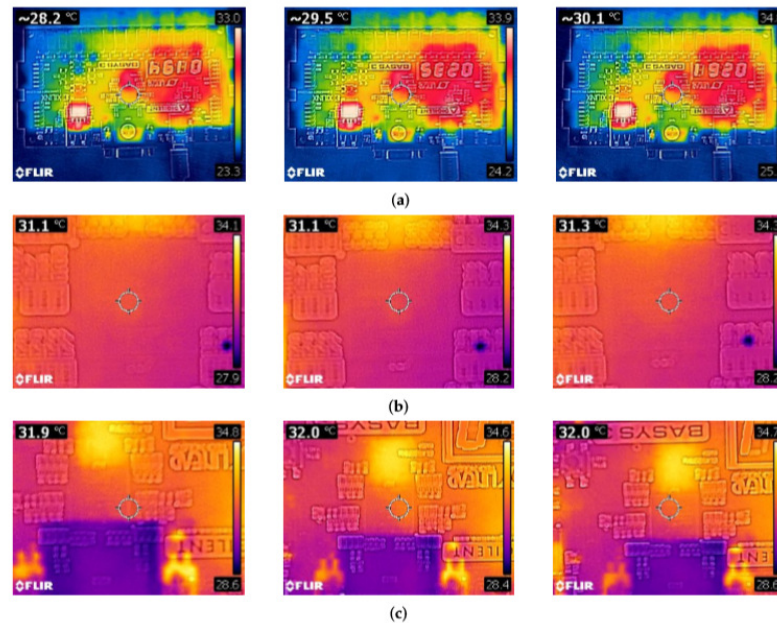


Fig5.1 The set of figure (a) from right to left represents the readings for the trojan free,sleeping and active respectively. The set of Figure (b) demonstrate a close up representation of the trojan free, sleeping and active on the board, while the set of Figure (c) provide close up temperature readings of the trojan active on the board over a period of 10 min.

5.3.2 Logic-Based Testing.

Standard logic testing of FPGAs by automatic test pattern generation (ATPG) tools is used for detecting faults. Using input vectors, all the programmable logic blocks can be tested to function correctly without faults. For example, a stuck-at-0 fault in the programmable logic blocks can be detected by mapping an AND function in the blocks and applying all-1 inputs. However, since Trojan models are very different from fault models, a better approach is required to detect Trojans. For example, an attacker can insert a Trojan which uses many values of the LUT SRAM cells or configuration cells as trigger nodes, and such a Trojan will not be detected using testing based on fault models. Due to the availability of a large number of

programmable blocks containing countless nodes, exhaustive testing of all combinations of nodes is infeasible. For a k -input LUT, having $L = 2^k$ cells in the logic block, $F = 2^{\text{pow}(2^k)}$ distinct functions are possible. For an n -input Trojan, the inputs can be chosen in LC_2 ways from the L cells. Since each combination can in turn be one out of 2^n values, the total number of functions that need to be mapped to exhaustively test a logic block becomes $F = 2^{\text{pow}(2^k)} \cdot 2^n$. For example, for a 2-input LUT having four cells, a 2-input Trojan can be chosen from the four cells in $4C_2=6$ ways. If the chosen two cells are designated a and b , then the trigger values for the Trojan can be $ab;(\bar{a}b);a(\bar{b});(\bar{a})(\bar{b})$, requiring 24 functions to be mapped. However, since entire functions are mapped onto the LUTs, mapping one function with values $a;b;c;d$ can detect several Trojans such as $ab;bc;cd$, etc., thus requiring fewer functions to be mapped. Still, if the trigger nodes are distributed among logic blocks, the sheer number of logic blocks, LUTs, and configuration cells makes it impossible for exhaustive testing to be used for Trojan detection. Due to this restriction, we propose a statistical approach of iterative self-checking based on the MERO test approach as shown in fig 5.1. Since the probability of activating a Trojan using the logic testing approach decreases with the number of trigger inputs, we assume that the number of trigger nodes n is small (two to four) and only distributed within nearby CLBs. Above algorithm shows the major steps in the proposed statistical test approach. We begin with a cluster of logic blocks of size S , the number of nodes B , random configuration set C , and input pattern set I . For each configuration mapped to the logic blocks, we activate the nodes to logic-0 and logic-1 several times using different input patterns. This procedure is done concurrently with the other remaining clusters of logic blocks, such that for each configuration all clusters are tested simultaneously. The outputs of the clusters can be tested by a few logic blocks configured as output response analyzers (ORA). These ORAs can be exhaustively tested at the beginning of the test procedure. Any Trojan that is activated will be observed at the primary outputs. Then, the cluster size is iteratively increased (e.g., to include two neighbouring logic blocks) and the process is repeated. Such a statistical approach of testing can be effective since an attacker does not know the exact test procedure to cleverly insert malicious circuits. Moreover, for larger combinational and sequential Trojans, this approach can be useful to cause partial activation of Trojans for detection using side-channel techniques.

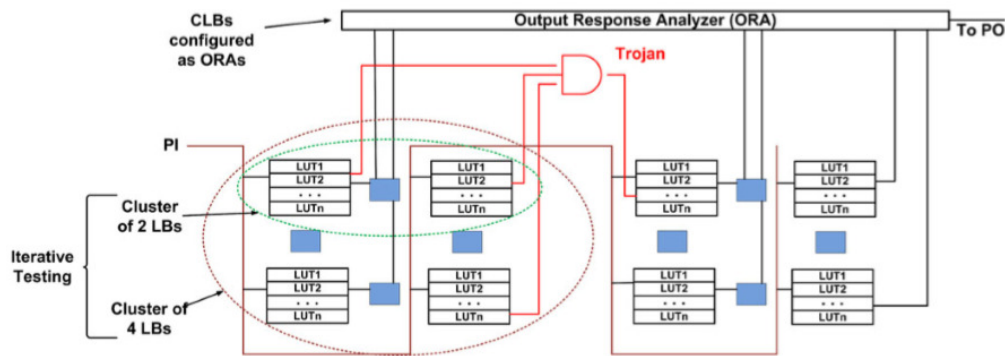


Fig5.2: Iterative logic-based self-checking test.

Redundancy and re-configurability are two key features of FPGA devices that can be that helpful for Trojan detection. Just as these features are used to counter run-time failures in FPGAs, so can they be used to counter against FPGA hardware and design Trojans? In the case of FPGA hardware, re-configurability allows the activation of several nodes in the logic blocks through different logic values. Redundancy can be used during testing, for example, by using N -modular redundancy to ensure that the trigger nodes present in the ORAs can also be detected by comparing the outputs of many ORAs. This is under the assumption that

Trojans in FPGA hardware (localized or distributed) do not affect all the resources in the same way. This can be coupled with dynamic run-time re-configurability to improve the level of security.

5.3.3 Side Channel Analysis

Logic-based testing may not be effective for activating large combinational or sequential Trojans due to the extremely large number of possible trigger nodes. Side-channel analysis involves the measurement and analysis of information obtained from an IC's side-channels. The information could be based on timing, power consumption, or electromagnetic radiation. Side-channel analysis has been proposed previously as a powerful technique to detect malicious insertions in an IC. In this section, we specifically concentrate on side-channel information obtained from power consumption in the device. Static power contains comprehensive information regarding all the gates in the chip (including malicious gates/ transistors). Trojans causing physical damage by creating electrical conflicts can also be detected using side-channel analysis since these Trojans result in a large current flow through the power supply. A simple design file can be loaded that configures I/O ports as inputs and then measures the supply current. If these Trojans simultaneously try to configure the port as an output, then a very large current can be detected by current sensors in the device, indicating a malicious modification. Since on-chip current sensors may also be tampered in the foundry during production, they must be tested thoroughly to identify any tampering. An alternative and secure strategy would be to use an on-board current sensor to detect short-circuit conditions. Trojans which do not cause physical damage and only cause logical malfunction may be extremely difficult to detect by analyzing static power. This is due to the difficulty in isolating the contribution of the malicious insertions from the measured power traces in ICs containing many millions of transistors. On the other hand, transient or dynamic power can be controlled by applying input vectors to reveal information about a few gates which are switching at any given time. The advantage of this type of analysis is that, unlike logic-based testing, a Trojan does not have to become active for detection; it merely needs to cause switching in the Trojan to consume dynamic power. For the IP-independent Trojans presented in Section 3, transient power analysis can be an effective detection method. For example, a counter-based Trojan inserted in the clock manager module can be detected by applying a clock signal to the FPGA and applying constant inputs to prevent logic blocks from switching. An extraneous counter or any sequential circuit will consume transient power as it transitions from one state to another. This contribution to dynamic power can be identified and associated with malicious insertions after accounting for process noise and clock coupling power.

The following circuit is taken and without changing the propagation delay the combinational Trojan was inserted in lowest path delay thereby we can detect the Trojan using path delays.

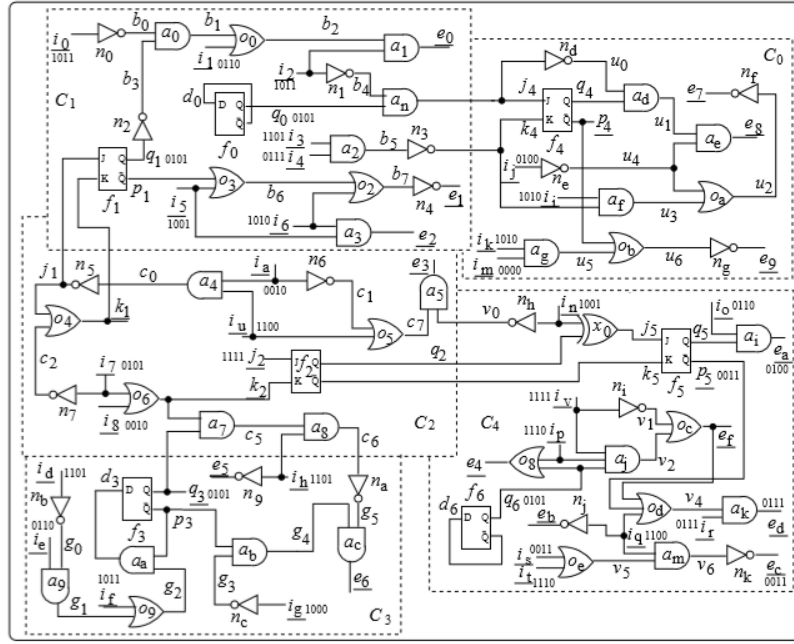


Fig5.3 Circuit under test

The algorithm stated earlier should be followed in achieving this.

Schematic of the circuit shown in fig 5.2 is obtained after synthesis is done. The LUTs and other components can be observed in the schematic shown in fig 5.3.

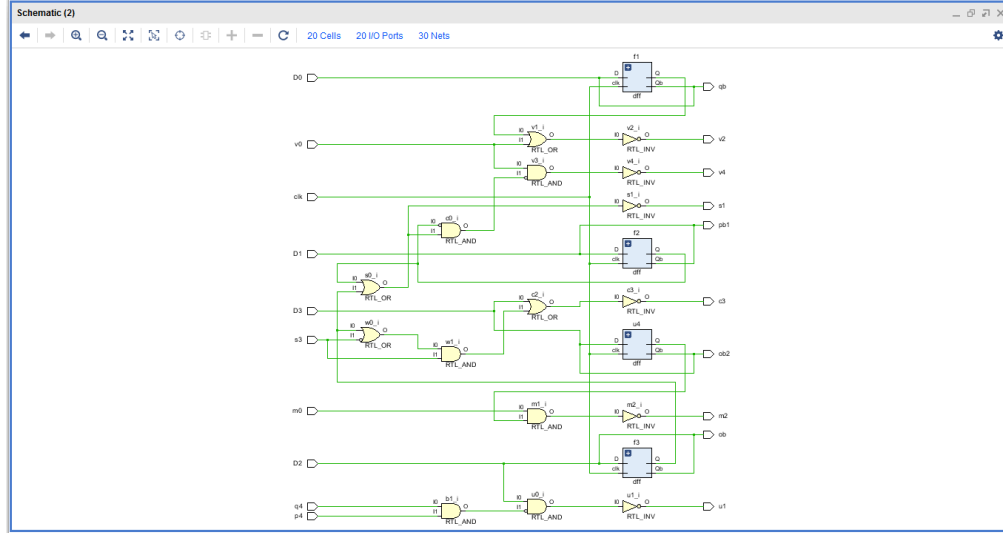


Fig5.4: Schematic diagram for the circuit in fig5.3

The floor planning for the trojan inserted and trojan free circuit is shown in fig 5.5

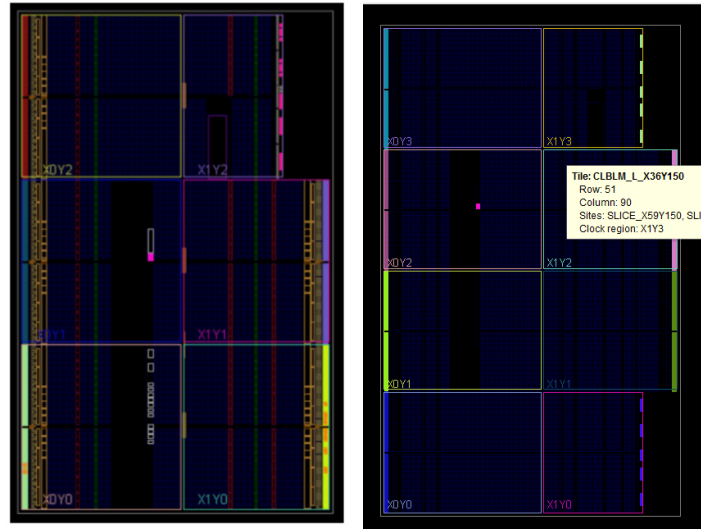


Fig 5.5: Floor planning for trojan free circuit and trojan inserted circuit

The simulation results for trojan effected circuit is shown in fig5.6

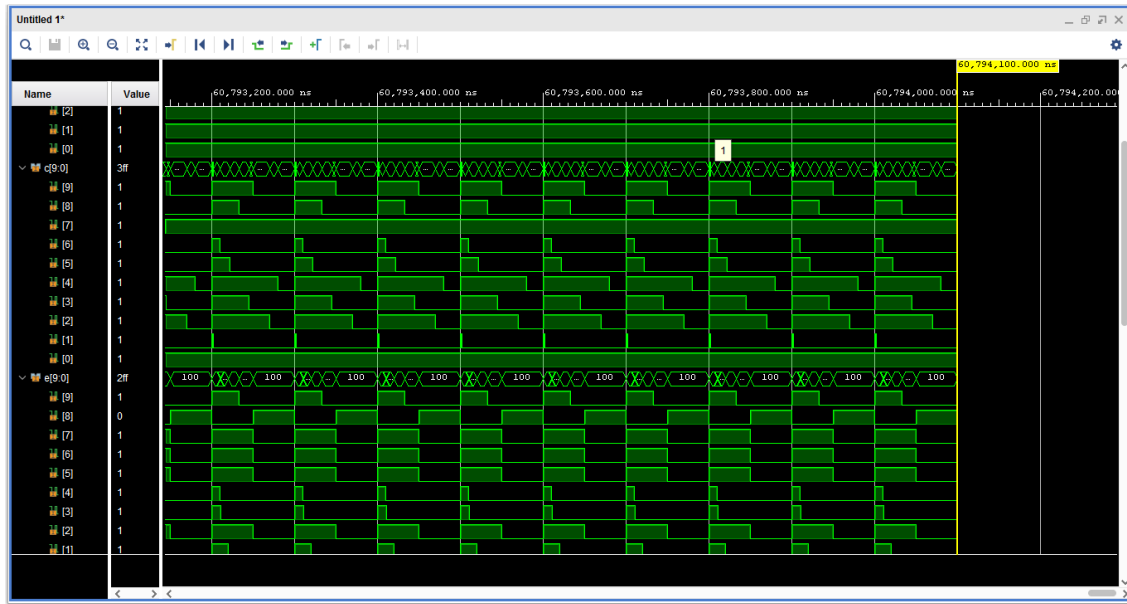


Fig 5.6 Simulation Results of trojan free circuit

The simulation results in fig 5.6 shows the normal functioning circuit which is trojan free circuit.

5.3.4 Using Ring Oscillators

The same circuit specified in fig 5.2 is again tested for trojan detection using ring oscillator. The ring oscillator is shown in fig 5.7

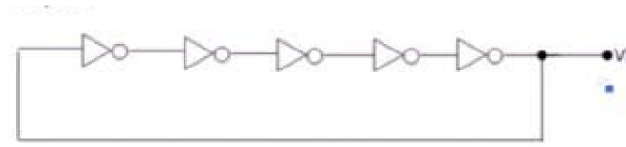


Fig 5.7: 5 Bit Ring Oscillator

The ring oscillator is mainly an important device in this hardware trojan insertion and detection as due to its buffering action through NOT gates lands up with an unpredictable output stage. This is the reason main hardware trojan insertion is done with ring oscillators.

The simulation results for ring oscillator trojan inserted circuit are shown in fig 5.8.

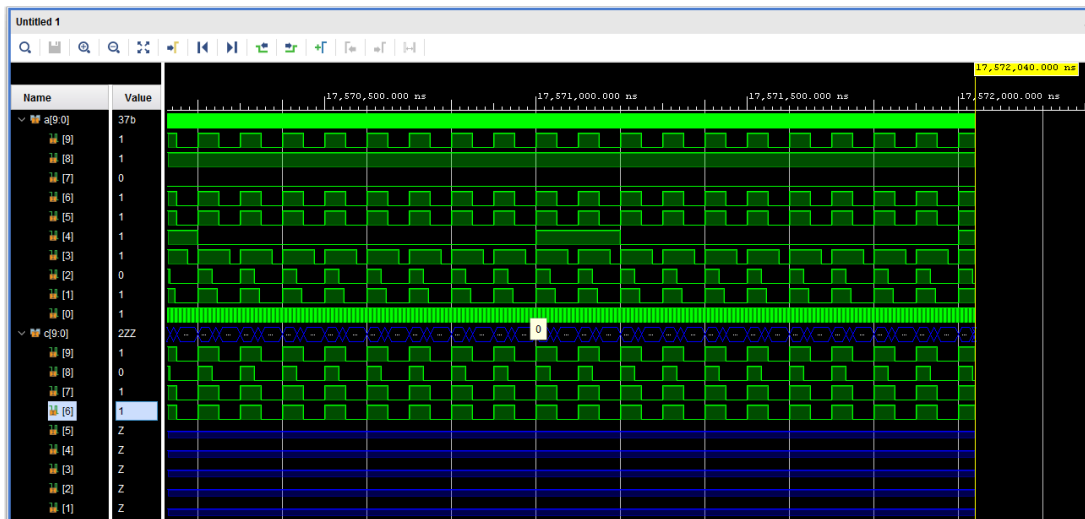


Fig5.8: Simulation Results for circuit in fig 5.2 using ring oscillator trojan

The simulation results for a ring oscillator trojan inserted circuit is shown in fig 5.8. Here few of the intermediate stages where the output is taken for analysing have been in high-impedance state. It shows the insertion of an unwanted component here called a trojan.

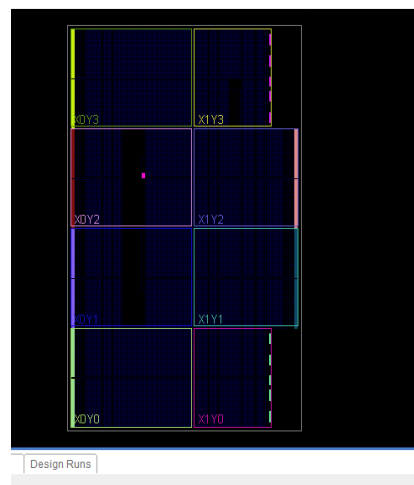


Fig 5.9The floor planning for trojan free and ring oscillator trojan inserted circuit

The floor planning for trojan free and ring oscillator trojan inserted circuit is shown in fig5.9

The schematic for ring oscillator trojan inserted circuit is shown in fig5.10

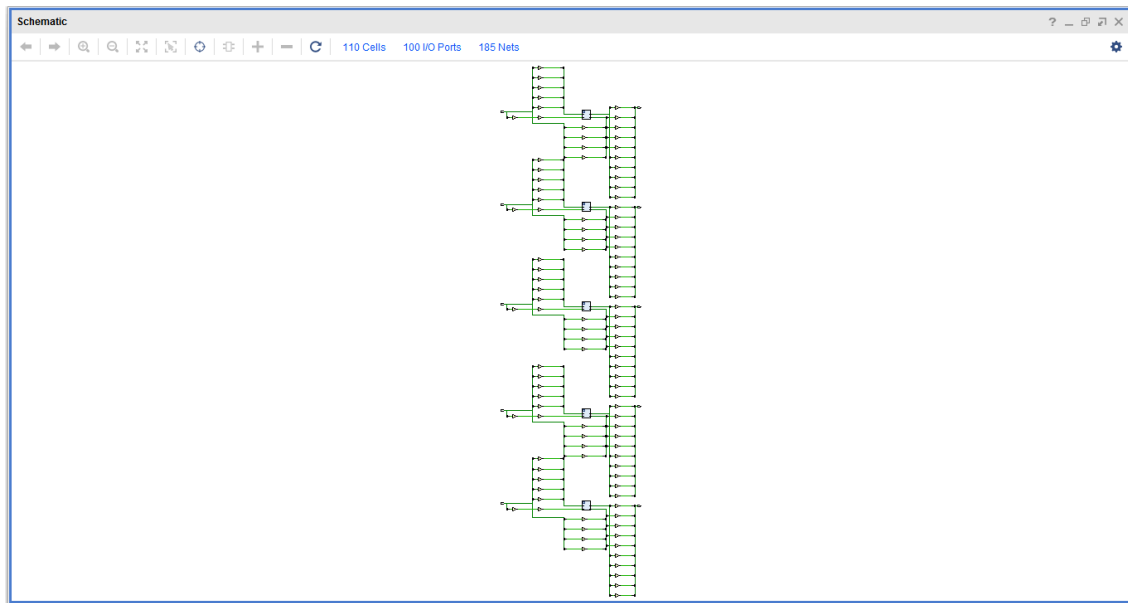


Fig 5.10: Schematic for ring oscillator trojan inserted circuit

Table 5.1: Parameters comparison of Trojan Free and Trojan Effectd Circuit

Trojan Free Circuit	Trojan Effectd Circuit
LUTs utilized are 60	LUTs utilized are 70
Slice Registers utilized are 40	Slice Registers utilized are 40
IOBs utilized are 100	IOBs utilized are 120
Registers as flip-flops are 40	Registers as flip-flops are 40
BUFGCTRL are 15	BUFGCTRL are 20

From the table 5.1 we can observe the performance of both the trojan affected and trojan free circuits in the parameters like LUTs, Slice registers, IOBs, Registers as flip-flops, BUFGCTRL.

The components like LUTs, IOBs and BUFGCTRLs are increased in trojan inserted circuit.

Few components like Slice Registers and Registers as flip flops remained same. From this observation we can extract that the number of memory elements utilized are same in both trojan free and trojan affected circuits.

CHAPTER 6

Conclusion and Future Scope

Formal methods are great tool in order to prove that the implementation of an electronic design behaves as specified. Hence, we identify equivalence checking as adequate measure in order to reveal manipulations of the bit-stream configuration, which is well-known state of the art. However, the big problem is that bit-stream verification is not a popular task in today's field-programmable gate array (FPGA) design and verification flows. One of the major reasons for this fact is that bit-stream formats are not publicly documented, which makes it hard for third-party verification tool vendors to offer solutions which prove that the bit-stream configuration is formally equivalent to the original HDL description. This project shows that it is easily possible to inject malicious behaviour into electronic designs using compromised design tools without being noticed by neither the designer nor the state-of-the-art tools targeted at Trojan detection. Therefore, we call for open and publicly documented bit-stream formats in order to increase trust in FPGA systems. Provided that bit-stream formats are publicly documented, it is easy for an equivalence check between the bit-stream configuration and the place list to reveal the attack we presented in this paper. However, once the design complexity increases of both the design and malicious functionality, more sophisticated approaches to equivalence checking are needed. First, equivalence between the bit-stream configuration and the place list must be ensured. In a second step, equivalence between the place list and the original HDL description can prove that the design can be trusted. We will address these issues in the future in order to demonstrate the feasibility of effective FPGA bit-stream verification.

This work provides sharper bounds for the case of detection of hardware trojans using off-the-shelf devices, allowing reducing the costs associated with trojan detection. The increasing number of devices being produced by un-trusted foundries puts critical infrastructure at the centre of attention. In this manuscript, we highlighted the dilemma of finding a one fits all solution to the problem finding hardware trojans fitting different taxonomies. To this end, we presented the corner stone for the detection of hardware trojans using off-the-shelf devices. We successfully demonstrated the ability of off-the-shelf devices to detect trojans in different settings, namely: sleeping and active. We believe that our practical work has the enormous potential in the successful detection of hardware trojans.

In the future we will aim at developing techniques to use thermal imaging for the detection of large scale hardware trojan infection and explore other trojan taxonomies in more intricate designs and with advanced malicious purposes. While we believe this technique is fully applicable to FPGAs, the technique might not be well suited for the denser ASICs and slight modifications might be required both in the methodology and in the off-the-shelf tool used. Moreover, we believe that this method could be used widely with the democratisation of specialised off-the-shelf hardware, following Moore's law with higher detection accuracy and better thermal imaging capabilities. Future work will compare the technique proposed against smaller known trojans and the process variation and manufacturing variation will be taken into account. Furthermore, the number of test vectors for Vivado power estimator will be increased in order to increase its accuracy.

REFERENCES

1. S. Mal-Sarkar, R. Karam, S. Narasimhan, A. Ghosh, A. Krishna and S. Bhunia, "Design and Validation for FPGA Trust under Hardware Trojan Attacks," in *IEEE Transactions on Multi-Scale Computing Systems*, vol. 2, no. 3, pp. 186-198, 1 July-Sept. 2016.
2. S. Moein, J. Subramnian, T. A. Gulliver, F. Gebali and M. W. El-Kharashi, "Classification of hardware Trojan detection techniques," *2015 Tenth International Conference on Computer Engineering & Systems (ICCES)*, Cairo, 2015, pp. 357-362.
3. P. Kitsos and A. G. Voyiatzis, "FPGA Trojan Detection Using Length-Optimized Ring Oscillators," *2014 17th Euromicro Conference on Digital System Design*, Verona, 2014, pp. 675-678.
4. A. P. Fournaris, L. Pyrgas and P. Kitsos, "An FPGA Hardware Trojan Detection Approach Based on Multiple Parameter Analysis," *2018 21st Euromicro Conference on Digital System Design (DSD)*, Prague, 2018, pp. 516-522.
5. A. Al-Anwar, Y. Alkabani, M. W. El-Kharashi and H. Bedour, "Hardware Trojan detection methodology for FPGA," *2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM)*, Victoria, BC, 2013, pp. 177-182.
6. D. M. Shila and V. Venugopal, "Design, implementation and security analysis of Hardware Trojan Threats in FPGA," *2014 IEEE International Conference on Communications (ICC)*, Sydney, NSW, 2014, pp. 719-724.
7. Y. Chen, W. Zhang and H. H. Li, "A hardware security scheme for RRAM-based FPGA," *2013 23rd International Conference on Field programmable Logic and Applications*, Porto, 2013, pp. 1-4. doi: 10.1109/FPL.2013.6645556
8. A. Amaricai, C. Gavriliu and O. Boncalo, "An FPGA sliding window-based architecture harris corner detector," *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*, Munich, 2014, pp. 1-4.
9. DARPA, TRUST in Integrated Circuits (TRUST), 2008. [Online]. Available: <http://www.darpa.mil/mto/programs/trust/index.html>
10. S. Trimberger, "Trusted design in FPGAs," in *Proc. 44th Des. Autom. Conf.*, 2007, pp. 5 8.
11. I. Hadzic, S. Udani, and J. Smith, "FPGA viruses," in *Proc. 9th Int. Workshop Field Programmable Logic Appl.*, 1999, pp. 291 300.
12. J. Rajendran, V. Jyothi, O. Sinanoglu, and R. Karri, "Design and analysis of ring oscillator based design for trust technique," in *Proc. 29th VLSI Test Symp.*, 2011, pp. 105 110.
13. H. Salmani, M. Tehranipoor, and J. Plusquellic, "A novel tech nique for improving hardware Trojandetection and reducing Trojanactivation time," *IEEE Trans. Very Large Scale Integration Syst.*, vol. 20, no. 1, pp. 112 125, Jan. 2011.
14. T. Ban and L. A. B. Naviner, "A simple fault tolerant digital voter circuit in TMR nanoarchitectures," in *Proc. 8th IEEE Int. NEWCAS Conf.*, 2010, pp. 269 272.

15. L. Lin, W. Burleson, C. Paar, "MOLES: Malicious off chip leakage enabled by side channels," in Proc. Intl. Conf. Comput. Aided Des., 2009, pp. 117–122.
16. N. Yoshimizu, "Hardware Trojan detection by symmetry breaking in path delays," in Proc. IEEE Int. Symp. Hardware Oriented Security Trust, 2014, pp. 107–111.
17. Bao, C.; Forte, D.; Srivastava, A. Temperature tracking: Toward robust run-time detection of hardware Trojans. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 2015, 34, 1577–1585.
18. Jin, Y.; Makris, Y. Hardware Trojan detection using path delay fingerprint. In Proceedings of the 2008 IEEE International Workshop on Hardware-Oriented Security and Trust (HOST), Anaheim, CA, USA, 9 June 2008; pp. 51–57.
19. Banga, M.; Hsiao, M.S. A region based approach for the identification of hardware Trojans. In Proceedings of the 2008 IEEE International Workshop on Hardware-Oriented Security and Trust (HOST), Anaheim, CA, USA, 9 June 2008; pp. 40–47.
20. Shende, R.; Ambawade, D.D. A side channel based power analysis technique for hardware trojan detection using statistical learning approach. In Proceedings of the 2016 Thirteenth International Conference on Wireless and Optical Communications Networks (WOCN), Hyderabad, India, 21–23 July 2016; pp. 1–4.
21. He, J.; Zhao, Y.; Guo, X.; Jin, Y. Hardware Trojan Detection Through Chip-Free Electromagnetic Side-Channel Statistical Analysis. *IEEE Trans. Very Large Scale Integr. (VLSI) Syst.* 2017, 25, 2939–2948.
22. Nowroz, A.N.; Hu, K.; Koushanfar, F.; Reda, S. Novel Techniques for High-Sensitivity Hardware Trojan Detection Using Thermal and Power Maps. *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.* 2014, 33, 1792–1805.
23. Narasimhan, S.; Du, D.; Chakraborty, R.S.; Paul, S.; Wolff, F.; Papachristou, C.; Roy, K.; Bhunia, S. Multiple-parameter side-channel analysis: A non-invasive hardware Trojan detection approach. In Proceedings of the 2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), Anaheim, CA, USA, 13–14 June 2010; pp. 13–18.
24. Department of Justice Press Release. VisionTech Administrator Sentenced to Prison for Role in Sales of Counterfeit Circuits Destined to US Military. Available online: <https://www.ice.gov/news/releases/viontech-administrator-sentenced-prison-role-sales-counterfeit-circuits-destined-us> (accessed on 17 September 2017).
25. Bellekens, X.; Seeam, A.; Nieradzinska, K.; Tachtatzis, C.; Cleary, A.; Atkinson, R.; Andonovic, I. Cyber-physical-security model for safety-critical iot infrastructures. In Proceedings of the Wireless World Research Forum Meeting, Santa Clara, CA, USA, 21–23 April 2015.
26. Omoogun, M.; Ramsurrun, V.; Guness, S.; Seeam, P.; Bellekens, X.; Seeam, A. Critical patient eHealth monitoring system using wearable sensors. In Proceedings of the 2017 1st International Conference on Next Generation Computing Applications (NextComp), Mauritius, 19–21 July 2017; pp. 169–174.

APPENDIX

Paper Published in **ALOCHANA CHAKRA JOURNAL** (UGC Approved)

Insertion and Detection of Hardware Trojans using Side Channel Analysis

P. Devi Pradeep^{#1}, P.Akhil^{*2}, Ch. Shalini^{#3}, G.S.P.M.Manohar Rao^{*4}, Ch. Viswa Yadeedya^{#5}

[#]ECE Department, ANITS

¹ devipradeep.ece@anits.edu.in

² akhil23.pilaka@gmail.com

³ shalini.chinta7@gmail.com

⁴ gspmmmanoharrao@gmail.com

⁵ viswayadeedya@gmail.com

Abstract— Security of any system has been related to the security of the information being processed. The hardware used for information processing has been considered trusted. This information should not get leaked to the third parties. A trojan horse or trojan is a type of malware that is often disguised as legitimate software. Trojans can be employed by cyber-thieves and hackers trying to gain access to user's systems and steal information. A Hardware Trojan (HT) is a malicious modification of the circuitry of an integrated circuit. A hardware trojan is completely characterized by its physical representation and its behaviour. Hardware Trojan Threats (HTTs) are stealthy components embedded inside integrated circuits (ICs) with an intention to attack and cripple the IC similar to viruses infecting the human body. Hardware trojans are designed by attackers to add unwanted functionality to the design. There is no a standard procedure to design hardware trojans, as its design is reliant upon the attacker's goals and available resources. This paper conveys a solution to prevent hardware trojans by detecting them and safeguarding the system with the help of various hardware Trojan detection methods.

Keywords— Hardware Trojan, FPGA, Ring Oscillators, IC, ASIC

I. INTRODUCTION

A hardware trojan can be described as a malicious alteration or inclusion to an integrated circuit (IC) that will either alter its intended function or cause it to perform an additional malicious function. These malicious inclusions or alterations are generally programmed to activate only under a specific set of circumstances created by an attacker and are extremely hard to detect when in their dormant state. As technology advances, so does the demand for IC boards leaving many technology companies without the resources to produce secure enough ICs to meet current demands. This has pushed companies into the 'fabless' trend prevalent in today's semi-conductor industry, where companies are no longer attempting to produce the goods in their own factories, but instead are outsourcing the process to cheaper factories abroad. This growth brings with it a significant rise in the level of threat posed by hardware trojans, a threat that directly affects all companies concerned with products that utilize ICs. This encompasses many different industries, including the military and telecommunications companies, and can potentially affect billions of devices from mobile phones and computers to military grade aviation and detection devices, particularly at a time when wireless devices are being introduced as links in critical infrastructure, compounding trust and security issues even further. It is also a direct threat to the already vulnerable Internet of Things, meaning that wireless-enabled household devices also become potential targets.

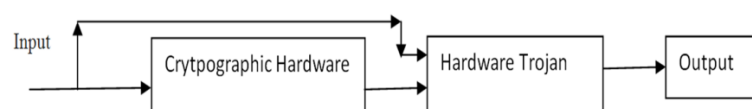


Fig 1: Block Diagram of Hardware Trojan Inserted Circuit

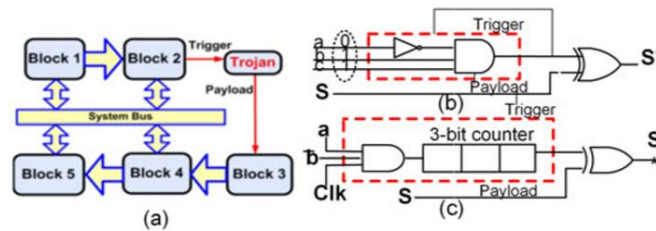


Fig 2. (a) General model of a hardware Trojan circuit realized through malicious modification of a hardware. (b) An example of combinational Trojan. (c) An example of sequential Trojan.

There are different types of hardware Trojans.

The condition of Trojan activation is referred to as the trigger, and the node affected by the Trojan is referred to as its payload. Trojans can be classified based on their triggering conditions or payload mechanisms. The trigger mechanism can be either digital or analog. Digitally triggered Trojans can be classified into combinational and sequential Trojans. Trojan can also be classified into digital and analog based on the payload mechanisms. Digital Trojans invert the logic values at internal nodes or modify the contents of memory locations. Analog payload Trojans may affect circuit parameters, such as performance, power, and noise margin.

Types of Hardware Trojan

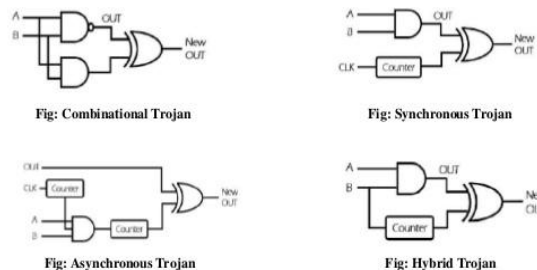


Fig3:Types of Hardware Trojans

II. THEORY

Hardware is a collection of physical elements that constitutes a computer system. Hardware is used by everyone even if they are not aware of it. Hardware in this context might be:

a. Computer Hardware: Some pieces of computer hardware are Processors, firmware, memory etc.

i. Processors: is the electronic circuitry within a computer that carries out the instructions of a computer program by performing the basic arithmetic, logical, control and input/output (I/O) operations specified by the instructions

ii. Firmware: is the combination of a hardware device, e.g. an integrated circuit, and computer instructions and data that reside as read only software on that device

iii. Memory: Memory refers to the device used to store information for use in a computer.

b. Mobile Hardware: SIM Card, RFID, Smart Card Chip and Pin

i. SIM Card: Subscriber Identity Module (SIM) card is used for securing the International Mobile Subscriber Identity (IMSI). It is an integrated circuit which is made available in different parameters like operating voltages, memory etc. It is used in mobile communications in a handy way.

ii. RFID: Radar Frequency Identification works on the radio waves. It can be used for tracking any objects like books, cars etc. It consists of RFID reader and RFID tags. It does not require line of sight for tracking the objects like barcode reader.

iii. Smart Card: Smart Card consists of a small chip which can be a microcontroller or a memory chip. These cards are designed in such a way to secure the information efficiently. Debit cards and credit cards are the examples of smart cards.

iv Chip & Pin: "Chip" refers to a computer chip embedded in the smartcard, and "PIN" refers to a personal identification number. PIN should be kept secured and the user should enter the PIN wherever required.

Using these PIN and dummy cards different hackers are hacking the accounts and stealing the information as well as money.

c. **Future Hardware:** Physically Un-clonable Functions (PUFs) have a unique fingerprint. It cannot accept the creation of a fingerprint which already holds the same fingerprint. These are used in high-end security of the information.

A) Attacks

The following are the different types of attacks in the hardware:

1. **Physical Attacks:** The major difference between a hardware and software Trojan is the software Trojan can be inserted using algorithms in anti-virus devices or IPs, this has nothing to do with the change or insertion of circuitry whereas the hardware Trojan is inserted using special circuitry in the ICs where the functionality of original circuit remains un-altered but the information can be accessed by the third parties due to the Trojan triggering conditions. Thus the hardware Trojan attacks are termed as the physical attacks as the physical components are introduced in the IC. The hardware Trojans can be detected using various methods specified in section III of this paper.

2. Generally, the hardware wants to protect secured information. Thus, the secured information will be embedded in a physical object. For example, the ATM card wants to protect your pin, the pin is encoded in the card. If the attacker probes the chip of the card and reads the pin then the card is useless and loses its importance. The secured information in the hardware should not be writable even though it provides the information on the card when placed on a terminal.

3. **Attack Vectors:** The big industries will depend on IC manufacturing industries for the manufacturing purpose. In this process the IC manufacturers may insert a Trojan inside the circuit without knowing to the other people so that the circuit is Trojan inserted circuit. This will be given to the industry and the information will be accessible to the IC manufacturer which will be a fraud. Generally these types of attacks will focus on military and safety of the nation. The probability of occurrence of these types of attacks will be low but if they occur they will cause a huge harm to the safety of the nation. For example, subway cards need to be rewritten to every month so in that case, the terminals have write access to the card but the user doesn't have write access so it can't be overwritten by the user. To fabricate the hardware, the laboratory/factory needs to be trusted. And to prove they are trustworthy, the laboratory gets certified. The certification body sends people in, to audit them, check out the employees and their procedures and conclude whether the laboratory is trusted or not.

4. **Supply Chain:** Generally after the manufacturing of the device it will be shipped to the customer. In this process the attackers make advantage in the shipping process and they might explore the information by introducing a Trojan without disturbing the circuit functionality and re-packs the IC as it is and then ships to the customers. This is a serious type of attack and it should be taken care by the customer alone to avoid these sorts of attacks.

5. **Accidents:** There will be lot of memory based devices like USB Keys, which may contain the malware in them accidentally this could affect the user's system. The company that created this hardware may not be aware of the malware in the device. Examples of companies that had this kind of accidents are IBM, Dell, Samsung, HP, Apple etc. All the attacks stated above are the main reasons for why the user might get an affected hardware.

RFID can't perform any computations based on the signal sent to it whereas smart card performs computations based on what was sent to it. Micro-controller is like an ID with no chip. ASICS are fabricated circuits that are custom made to do implementations so all the processors and memories are ASICS. ASICS are expensive because they are custom made, and before the design is committed to ASICS, a lot of testing will be done with FPGAs to make sure that circuit actually works.

FPGAs are programmable chips. When the chip is bought, it will be blank and it can be programmed with software to do anything. They are not as fast as ASICS because they are general purpose devices. FPGAs are faster than software but slower than ASICS.

B) Why circuit attacks are done?

The main reason for attacking circuits is to recover a secret that had been encoded in a piece of hardware or for the attacker to program a certain value to the circuit. The secret could be the actual algorithm itself. Some attackers reverse engineer the algorithm of the RFID or Smart Card to find flaws in the algorithm itself. This is because some developer wants to keep the algorithm used to protect the circuit a secret due to

the fact that the developers are not using a standard algorithm. The algorithm used should be a standard algorithm so as to know how to better protect it.

The Circuit attacks are described below:

1. **Black Box Testing:** In this attack, the attacker creates a circuit which sends an input to the circuit and receives an output. The attacker may use Bluetooth or GPS or any IP based Trojan insertion and performs this action. Based on the input and output behaviour, the attacker will decide the kind of the algorithm implemented. An example is Speed Gas RFID which is a proprietary stream cipher. The attackers found the documentation and modified it to discover the cipher used and break the circuit. This type of attack is non-invasive, meaning that the IC will not be destroyed when probed so it can be used another time.

2. **Physical Probing:** Physical probing is an attack where the attacker inserts a trojan which may not steal the information directly but tracks the data flow by the movement of data in the bus. This Trojan does not impact on the functioning of the circuit. Once the data path is known to the attacker then by performing the special techniques the algorithm can be broken and information gets stolen.

3. **Reverse Engineering:** This is a new style of attacking on ICs. To perform this, the attacker must get the smart card and make the circuit exposed. The smart card consists of different layers and each layer is peeled off until the schematic circuit is exposed. Once the circuit gets known to the attacker then the attacker takes high resolution photos and using machine learning the possible security information like PIN gets known to the attacker where the algorithm followed gets exposed to the attacker. An example of a smart card where the reverse engineering attack was performed on is the Mi-fare (Subway card).

4. **Fault Generation:** Few technologies are having more loop holes where the control of the system can be done by the undesired third parties. These technologies are controlled to the large extent these days. For example, if a system needs an input x for stopping the access to the user and if the user does not respond i.e. giving no input which means system is not getting input x . Here the user is supposed to be rejected for access by the system but instead of input x there is no input given to the system which makes the system still grant the access which is undesired. These are seen in olden days by changing the date in the computer and extending software license.

5. **Side Channel Analysis:** In side channel analysis the attackers makes use of some parameters like delay, power etc. Without changing the functioning of the circuit the Trojan will be inserted and upon the conditions specified by the attackers the Trojan triggers and thereby gets the information from the user. These are called side channel because they are outside the normal channels. They are non-invasive. It is slower than the normal attacks.

III. IMPLEMENTATION

The hardware Trojan detection can be done by using different methods. For any method the Algorithm shown below is followed.

A) Algorithm for Trojan Detection

The following steps are to be followed for detecting an hardware trojan.

1. Get the circuit under test
2. Retrieve the path delays and other parameters
3. $i=1$
4. Retrieve i th path
5. Give test vector to circuit under test
6. Measure the required parameters like power, delay, current etc
7. If delay equal to golden circuit's delay then go to 'a' else go to 'b'.
 - a. $i=i+1$
if i =number of available paths then go to 'f1' else go to step-4.
 - f1. Circuit is trojan free.
 - b. Circuit is affected.

B) Detection Methods

Hardware Trojan Detection methods can be classified into four categories.

1. Visual detection techniques
2. Logic testing
3. Side-channel analysis.
4. Ring Oscillator Based Approach

Visual detection techniques

This class of detection methods uses imaging to identify any malicious insertions in the chip. These techniques include using X-ray imaging, scanning optical microscopy (SOM), scanning electron microscopy (SEM), and picoseconds imaging circuit analysis (PICA). These methods, however, can be expensive in cost. These techniques suffer from lack of resolution to decipher logic/transistor/interconnect level information, primarily due to the obstruction by the stack of metal layers in modern FPGAs.

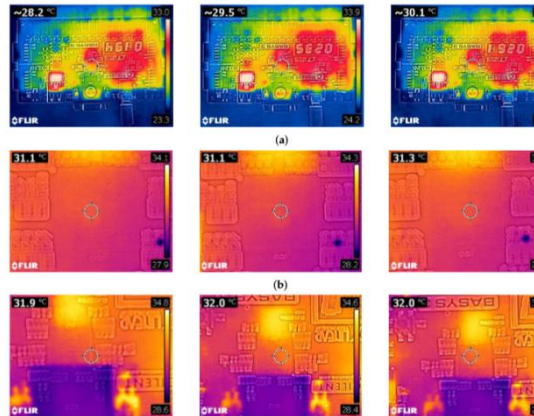


Fig4 The set of Figure(a) from right to left represents the readings for the trojan free, sleeping and active respectively. The set of Figure (b) demonstrate a close up representation of the trojan free, sleeping and active on the board, while the set of Figure (c) provide close up temperature readings of the trojan active on the board over a period of 10 min.

Logic testing

Standard logic testing of FPGAs by automatic test pattern generation (ATPG) tools is used for detecting faults. Using input vectors, all the programmable logic blocks can be tested to function correctly without faults. The leakage current, transient supply current and the maximum operating frequency were used in the trojan detection process. A disadvantage of this approach is that it is not effective for logic testing for small trojans. The trojan causes changes in circuit function and is abstraction level. This trojan can be implemented using either combinational or sequential logic and is internally triggered by a specific sequence or a counter. Gate Level Characterization (GLC) considers the timing delay, switching power and leakage power of each gate in the circuit. Experiments were performed using several benchmarks to obtain the GLC of simple gates such as AND, OR, NOT, NOR and NAND. The disadvantage of GLC is that the characterizing all gates in a circuit is a very time consuming process and the complexity can make it intractable to include all gates in the circuit.

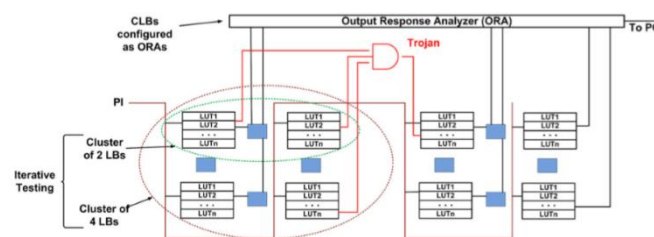


Fig5: Iterative logic-based self-checking test.

3. Side-Channel Analysis

A variety of parameters are analysed in Side-Channel Analysis. They are:

i) Delay:

Path Delay Sensors are used to measure the delay across the components. This sensor predicts the delay characteristics of specific sequences of operations in the circuit and compares these with reference values. A significant difference indicates that a trojan may be present.

ii) Current:

a) Self-referencing:

A self-referencing approach was employed to detect hardware trojans. A test vector is used to create a leakage current for a trojan free circuit. The leakage current for an IC under test is compared with the trojan free circuit to determine if a trojan may exist.

b) Current Integration:

Typically, the leakage current and current consumed by a chip is measured and compared with that of a trojan free circuit. Instead of using the current directly, it is integrated to convert current into charge which magnifies any differences in the measured current. Therefore, this approach can be useful in identifying small trojans where the effect on the current is masked by noise.

iii) Electromagnetic (EM) Radiation:

With the help of variation of electromagnetic radiation along the circuit we can detect the trojans. To detect this radiation an electromagnetic probe can be used. It was shown that significant differences can exist in the emissions at different chip locations due to the presence of hardware trojans. In particular, it is easier to detect a trojan at the corners of chips because of the proximity to the power lines. In EM radiation method the EM probe should be placed correctly across the circuit otherwise due to multiple reflections of the signal the probe may detect it wrongly. Also this method is not efficient in detecting trojan at the middle of the chip.

iv) Power Consumption:

Power measurements for random test patterns can be used to generate power characteristics using a trojan free circuit. These characteristics can be based on the power consumption as well as the noise generated from surrounded circuits. By applying the same random test patterns to all chips, they can be divided into groups according to their power characteristics. Reverse engineering can then be applied to a chip from each set to determine which groups are trojan free.

4. Ring Oscillator Based Approach

In ring oscillator approach the following circuit is considered and placed in any path such that the propagation delay should not change.

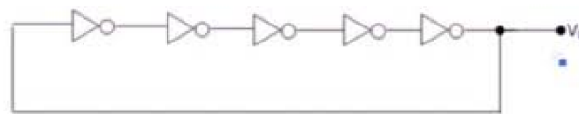


Fig 6 5 Bit Ring Oscillator

Generally the ring oscillator is placed in feedback path for continuous change in input which is unpredictable. This is based on differences in the RO frequency due to the presence of a trojan. Trojans were introduced without changing the layout of the circuit. RO of length 7 provides adequate detection performance.

In this project a circuit under test is considered and two types of Trojans namely combinational and ring oscillator were introduced in between the paths without changing the propagation delay and are simulated individually and the observations were extracted as in section IV. The entire simulation and synthesis was done using the software tool Xilinx Vivado2019.2.

IV. RESULTS

The following circuit shown in fig 7 is taken and without changing the propagation delay the combinational Trojan was inserted in lowest path delay thereby we can detect the Trojan using path delays.

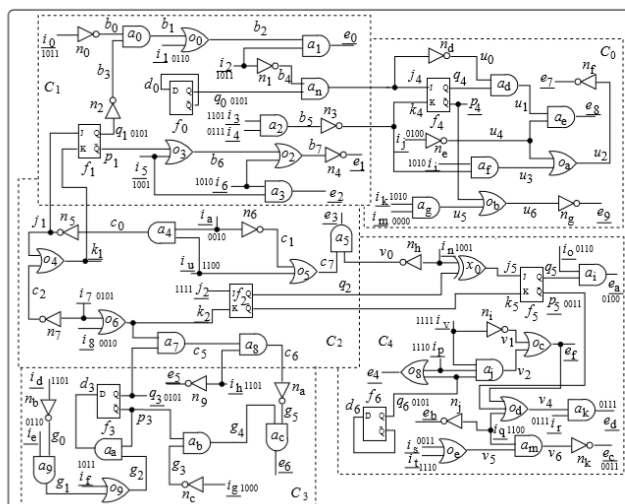


Fig7: Circuit under test

The floor planning for the trojan inserted circuit is shown in fig 8

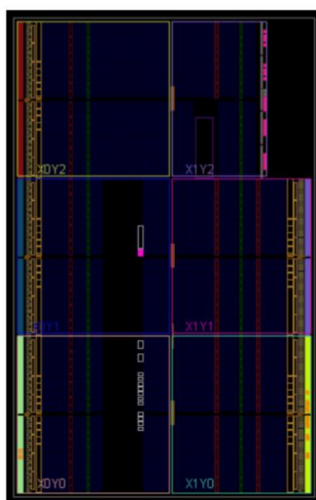


Fig8:Floor planning for the trojan inserted circuit

The floor planning for the trojan free circuit is shown in fig 9

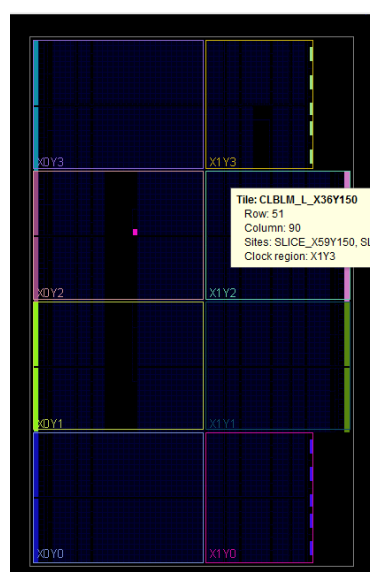


Fig9: Floor planning for trojan free circuit

The simulation results for trojan free circuit is shown in fig 10

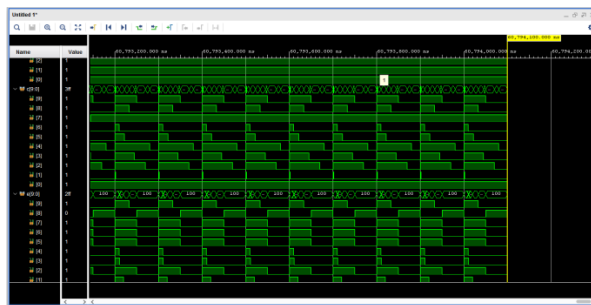


Fig10: Simulation results for trojan free circuit

The same circuit specified in fig 7 is again tested for trojan detection using ring oscillator. The ring oscillator is shown in fig 6.

The simulation results for combinational Trojan inserted circuit are shown in fig 11.

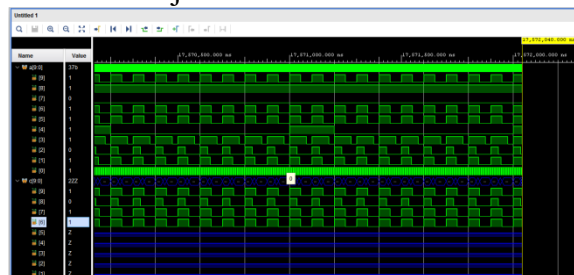


Fig 11: Simulation results for combinational trojan effected circuit

The simulation results for ring oscillator trojan inserted circuit is shown in fig 12.



Fig 12: Simulation results for ring oscillator trojan inserted circuit

The floor planning for ring oscillator trojan inserted circuit is shown in fig 13

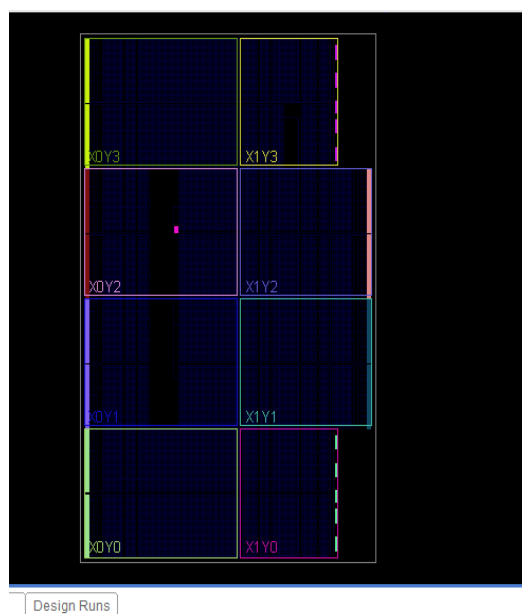


Fig 13: Floor planning for ring oscillator trojan inserted circuit

The following table shows the typical observations from the reports obtained after synthesis and simulation.

Table 1: Parameters comparison of Trojan Free and Trojan Effectuated Circuit

Trojan Free Circuit	Trojan Effectuated Circuit
LUTs utilized are 60	LUTs utilized are 70
Slice Registers utilized are 40	Slice Registers utilized are 40
IOBs utilized are 100	IOBs utilized are 120
Registers as flip-flops are 40	Registers as flip-flops are 40
BUFGCTRL are 15	BUFGCTRL are 20

V.CONCLUSION

The insertion of hardware trojans is not limited to just the fabrication stage. The trojan can be inserted at any stage of IC design cycle. Different trojans can be detected with different methodologies. A single methodology can't be used for all the trojans detection. In industry point of view the primary test should be conducted using PICA a Visual Detection Method. Finding a trojan in a complex circuit takes time so different test vectors along with different triggering mechanisms are to be implemented on the circuit under test. The trojan inserted through ring oscillator based approach is more complex than others to detect because the Trojan inserted through this approach will be generally activated at only some triggering conditions. Research is going on this insertion detection approach. In addition to this the upcoming engineers should be encouraged to work on cyber security and prevent the attacks well in advance as it was stated in section II the attacks might affect the security and safety of the nation so as an engineer it is the primary responsibility of an engineer to prevent this attacks. The scope in this field is vast.

REFERENCES

- [1] 1. S. Mal-Sarkar, R. Karam, S. Narasimhan, A. Ghosh, A. Krishna and S. Bhunia, "Design and Validation for FPGA Trust under Hardware Trojan Attacks," in IEEE Transactions on Multi-Scale Computing Systems, vol. 2, no. 3, pp. 186-198, 1 July-Sept. 2016.
- [2] 2. S. Moein, J. Subramanian, T. A. Gulliver, F. Gebali and M. W. El-Kharashi, "Classification of hardware Trojan detection techniques," 2015 Tenth International Conference on Computer Engineering & Systems (ICCES), Cairo, 2015, pp. 357-362.
- [3] 3. P. Kitsos and A. G. Voyiatzis, "FPGA Trojan Detection Using Length-Optimized Ring Oscillators," 2014 17th Euromicro Conference on Digital System Design, Verona, 2014, pp. 675-678.
- [4] 4. A. P. Fournaris, L. Pyrgas and P. Kitsos, "An FPGA Hardware Trojan Detection Approach Based on Multiple Parameter Analysis," 2018 21st Euromicro Conference on Digital System Design (DSD), Prague, 2018, pp. 516-522.
- [5] 5. A. Al-Anwar, Y. Alkabani, M. W. El-Kharashi and H. Bedour, "Hardware Trojan detection methodology for FPGA," 2013 IEEE Pacific Rim Conference on Communications, Computers and Signal Processing (PACRIM), Victoria, BC, 2013, pp. 177-182.
- [6] 6. D. M. Shila and V. Venugopal, "Design, implementation and security analysis of Hardware Trojan Threats in FPGA," 2014 IEEE International Conference on Communications (ICC), Sydney, NSW, 2014, pp. 719-724.
- [7] 7. Y. Chen, W. Zhang and H. H. Li, "A hardware security scheme for RRAM-based FPGA," 2013 23rd International Conference on Field programmable Logic and Applications, Porto, 2013, pp. 1-4.
- [8] doi: 10.1109/FPL.2013.6645556
- [9] 8. A. Amaricai, C. Gavrilu and O. Boncalo, "An FPGA sliding window-based architecture harris corner detector," 2014 24th International Conference on Field Programmable Logic and Applications (FPL), Munich, 2014, pp. 1-4.
- [10] 9. DARPA, TRUST in Integrated Circuits (TRUST), 2008. [Online]. Available: <http://www.darpa.mil/mto/programs/trust/index.html>
- [11] 10. S. Trimberger, "Trusted design in FPGAs," in Proc. 44th Des. Autom. Conf., 2007, pp. 5-8.
- [12] 11. I. Hadzic, S. Udani, and J. Smith, "FPGA viruses," in Proc. 9th Int. Workshop Field Programmable Logic Appl., 1999, pp. 291-300.
- [13] 12. J. Rajendran, V. Jyothi, O. Sinanoglu, and R. Karri, "Design and analysis of ring oscillator based design for trust technique," in Proc. 29th VLSI Test Symp., 2011, pp. 105-110.

- [14] 13. H. Salmani, M. Tehranipoor, and J. Plusquellic, "A novel technique for improving hardware Trojandetection and reducing Trojanactivation time," IEEE Trans. Very Large Scale Integration Syst., vol. 20, no. 1, pp. 112–125, Jan. 2011.
- [15] 14. T. Ban and L. A. B. Naviner, "A simple fault tolerant digital voter circuit in TMR nanoarchitectures," in Proc. 8th IEEE Int. NEWCAS Conf., 2010, pp. 269–272.
- [16] 15. L. Lin, W. Burleson, C. Paar, "MOLES: Malicious off chip leakage enabled by side channels," in Proc. Intl. Conf. Comput. Aided Des., 2009, pp. 117–122.
- [17] 16. N. Yoshimizu, "Hardware Trojandetection by symmetry breaking in path delays," in Proc. IEEE Int. Symp. Hardware Oriented Security Trust, 2014, pp. 107–111.
- [18] 17. Bao, C.; Forte, D.; Srivastava, A. Temperature tracking: Toward robust run-time detection of hardware Trojans. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 2015, 34, 1577–1585.
- [19] 18. Jin, Y.; Makris, Y. Hardware Trojan detection using path delay fingerprint. In Proceedings of the 2008 IEEE International Workshop on Hardware-Oriented Security and Trust (HOST), Anaheim, CA, USA, 9 June 2008; pp. 51–57.
- [20] 19. Banga, M.; Hsiao, M.S. A region based approach for the identification of hardware Trojans. In Proceedings of the 2008 IEEE International Workshop on Hardware-Oriented Security and Trust (HOST), Anaheim, CA, USA, 9 June 2008; pp. 40–47.
- [21] 20. Shende, R.; Ambawade, D.D. A side channel based power analysis technique for hardware trojan detection using statistical learning approach. In Proceedings of the 2016 Thirteenth International Conference on Wireless and Optical Communications Networks (WOCN), Hyderabad, India, 21–23 July 2016; pp. 1–4.
- [22] 21. He,J.;Zhao,Y.;Guo,X.;Jin,Y. HardwareTrojanDetectionThroughChip-FreeElectromagneticSide-Channel Statistical Analysis. IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 2017, 25, 2939–2948.
- [23] 22. Nowroz, A.N.; Hu, K.; Koushanfar, F.; Reda, S. Novel Techniques for High-Sensitivity Hardware Trojan Detection Using Thermal and Power Maps. IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst. 2014, 33, 1792–1805.
- [24] 23. Narasimhan, S.; Du, D.; Chakraborty, R.S.; Paul, S.; Wolff, F.; Papachristou, C.; Roy, K.; Bhunia, S. Multiple-parameter side-channel analysis: A non-invasive hardware Trojan detection approach. In Proceedings of the 2010 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST), Anaheim, CA, USA, 13–14 June 2010; pp. 13–18.
- [25] 24. Department of Justice Press Release. VisionTech Administrator Sentenced to Prison for Role in Sales of Counterfeit Circuits Destined to US Military. Available online: <https://www.ice.gov/news/releases/visiontech-administrator-sentenced-prison-role-sales-counterfeit-circuits-destined-us> (accessed on 17 September 2017).
- [26] 25. Bellekens, X.; Seeam, A.; Nieradzinska, K.; Tachtatzis, C.; Cleary, A.; Atkinson, R.; Andonovic, I. Cyber-physical-security model for safety-critical iot infrastructures. In Proceedings of the Wireless World Research Forum Meeting, Santa Clara, CA, USA, 21–23 April 2015.
- 26. Omoogun, M.; Ramsurrun, V.; Guiness, S.; Seeam, P.; Bellekens, X.; Seeam, A. Critical patient eHealth monitoring system using wearable sensors. In Proceedings of the 2017 1st International Conference on Next Generation Computing Applications (NextComp), Mauritius, 19–21 July 2017; pp. 169–174.