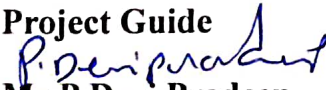


**DEPARTMENT OF ELECTRONICS AND COMMUNICATION ENGINEERING**  
**ANIL NEERUKONDA INSTITUTE OF TECHNOLOGY AND SCIENCES**  
*(Permanently Affiliated to AU, Approved by AICTE and Accredited by NBA & NAAC*  
*with 'A' Grade)*  
**Sangivalasa, Bheemili Mandal, Visakhapatnam dist.(A.P)-531162**




**CERTIFICATE**

*This is to certify that the project report entitled “TROJAN INSERTION AT ALGORITHM LEVEL TO ENHANCE HARDWARE SECURITY” submitted by Sk. Abdul Kareem (319126512181) D. Lakshmi Prasanna (319126512141) R. Harish (319126512174) A.N.V. Saimanikanth (320126512L23) in partial fulfillment of the requirements for the award of the degree of Bachelor of Technology in Electronics & Communication Engineering of Anil Neerukonda Institute of technology and Sciences (ANITS), 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  
ANITS

**Assistant Professor**  
**Department of E.C.E.**  
**Anil Neerukonda**  
**Institute of Technology & Sciences**  
**Sangivalasa, Visakhapatnam-531 162**

  
**Head of the Department**  
**Dr. B.Jagadeesh**  
Department of E.C.E  
ANITS

**Head of the Department**  
**Department of E C E**  
**Anil Neerukonda Institute of Technology & Sciences**  
**Sangivalasa - 531 162**

**TROJAN INSERTION AT ALGORITHM LEVEL  
TO ENHANCE HARDWARE SECURITY**

*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*

**Sk. Abdul Kareem (319126512181)**

**D. Lakshmi Prasanna (319126512141)**

**R. Harish (319126512174)**

**A.N.V. Saimanikanth (320126512L23)**

**Under the guidance of**

**Mr.P.Devi Pradeep**

**M.Tech (Ph.D)**

**Assistant Professor**



**ANITS**

**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**

**2022-2023**

## **ACKNOWLEDGEMENT**

We would like to express our deep gratitude to our project guide **Mr.P.Devi Pradeep Assistant Professor**, Department of Electronics and Communication Engineering, ANITS, for his guidance with unsurpassed knowledge and immense encouragement. We are grateful to **Dr. B.Jagadeesh**, 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**

**Sk. Abdul Kareem (319126512181)**

**D. Lakshmi Prasanna (319126512141)**

**R. Harish (319126512174)**

**A.N.V. Saimanikanth (320126512L23)**

# CONTENTS

<b>ABSTRACT</b>	<b>ix</b>
<b>LIST OF FIGURES</b>	<b>x</b>
<b>LIST OF ABBREVIATIONS</b>	<b>xii</b>
<b>CHAPTER 1 INTRODUCTION</b>	<b>01</b>
1.1 Project Objective	01
1.2 Background	02
1.3 Motivation	02
1.4 Project Outline	03
<b>CHAPTER 2 INTRODUCTION TO FPGA</b>	<b>05</b>
2.1 FPGA Architecture	07
2.2 Types of FPGA	08
2.2.1 SRAM FPGA	08
2.2.2 Anti fuse FPGA	09
2.2.3 Flash FPGA	10
2.3 FPGA Families	11
2.3.1 SRAM FPGA Families	12
2.3.2 Antifuse FPGA Families	12
2.3.3 Flash FPGA Families	12
2.3.4 Hybrid flash/SRAM FPGA Families	12
2.4 ARTIX-7 FPGA Trainer Kit	12
2.5 SLICEM	14
2.5.1 Look-up tables	14
2.5.2 Flip Flops	15
2.5.3 Fast Carry Logic	15
2.6 Interconnection	16
2.7 DSP Slice	17
2.8 Applications of FPGA	18
2.9 History of FPGA	19
2.10 Modern usage of FPGA	20

2.10.1 Particle physics experiments	20
2.10.2 Astronomy	20
2.10.3 Genomics	20
2.10.4 Machine learning	20
<b>CHAPTER 3 HARDWARE SECURITY</b>	<b>21</b>
3.1 Introduction	21
3.2 Different types of Attacks on Hardware	22
3.2.1 Side-channel attacks	22
3.2.2 Fault injection attacks	22
3.2.3 Reverse engineering attacks	22
3.2.4 Tampering attacks	22
3.3 Hardware Trojan Attacks	23
3.3.1 Design-time Trojans	23
3.3.2 Post-manufacturing Trojans	23
3.4 Types of Trojans	23
3.4.1 Logic manipulation Trojans	23
3.4.2 Data alteration Trojans	23
3.4.3 Clock manipulation Trojans	23
3.4.4 Power manipulation Trojans	23
3.4.5 Stealthy Trojans	24
3.4.6 False functionality Trojans	24
3.4.7 Activation-based Trojans	24
3.5 Counter measures against Trojan Attacks	24
3.5.1 Design-time countermeasures	24
3.5.2 Manufacturing-time countermeasures	24
3.5.3 Post-manufacturing countermeasures	24
3.6 Active and passive techniques in Hardware security	25
3.6.1 Active Hardware Security Techniques	25
3.6.1.1 Encryption	25
3.6.1.2 Obfuscation	25

3.6.1.3	Hardware-Based Security Mechanisms	25
3.6.2	Passive Hardware Security Techniques	26
3.6.2.1	Hardware-Based Monitoring and Debugging	26
3.6.2.2	Side-Channel Analysis	26
3.6.2.3	Fault Injection Analysis	26
3.7	Use of PUFs and TRNGs in Hardware Security	26
3.8	Randomization	28
3.8.1	Circuit-level randomization	28
3.8.2	Gate-level randomization	28
3.8.3	Layout-level randomization	28
3.8.4	Gate Substitution	29
3.8.5	Gate Duplication	29
3.8.6	Logic Restructuring	29
3.8.7	Polymorphism	29
3.9	Diversity	29
3.9.1	Randomization	30
3.9.2	Heterogeneity	30
3.9.3	Redundancy	30
3.9.4	Obfuscation	30
<b>CHAPTER 4</b>	<b>VERILOG HDL</b>	<b>31</b>
4.1	What is Verilog HDL	31
4.1.1	Bottom-up Design	31
4.1.2	Top-down Design	32
4.2	History of Verilog	33
4.3	Use of Verilog	34
4.3.1	ASIC Design	34
4.3.2	FPGA Design	34
4.3.3	SoC Design	34
4.3.4	Verification	34
4.3.5	High-Level Synthesis (HLS)	34

4.3.6	Machine Learning (ML)	35
4.4	Levels Of Abstraction	35
4.4.1	Behavioural level	35
4.4.2	RTL (Register Transfer Level) level	35
4.4.3	Gate level	35
4.5	Lexical Tokens of Verilog	36
4.5.1	Identifiers	36
4.5.2	Keywords	36
4.5.3	Operators	36
4.5.4	Literals	36
4.5.5	Delimiters	36
4.5.6	Comments	36
4.5.7	Function Calls	40
4.6	Modules	40
4.7	Module declaration	41
4.8	Module Installation	43
<b>CHAPTER 5 TROJAN INSERTION IN IP's AND SIMULATION RESULTS</b>		<b>46</b>
5.1	Introduction to IP	46
5.1	IP Encryption	46
5.1.1	Symmetric-key encryption	47
5.1.2	Asymmetric-key encryption	47
5.1.3	Hardware encryption	47
5.1.4	Obfuscation	47
5.2	Working of IP security	47
5.3	Components of IP Security	48
5.3.1	Encapsulating Security Payload (ESP)	48
5.3.2	Authentication Header	48
5.3.3	Internet Key Exchange(IKE)	48
5.4	Uses of IP Security	49
5.5	Adder or subtractor IP from IP catalog in vivado	50

5.5.1	Generating and Customizing IP from IP Core	50
5.5.2	Core Parameters	51
5.5.3	Modes to customize adder or subtractor IP	53
5.5.4	Latency Configuration	55
5.6	Trojan insertion at algorithm level	58
<b>CHAPTER 6 CONCLUSION AND FUTURE WORK</b>		<b>60</b>
<b>REFERENCES</b>		<b>62</b>



## **ABSTRACT**

A Trojan is defined as “malicious software program that is often spread through social engineering tactics, such as email phishing, and can be used to gain unauthorized access to a computer system or steal sensitive information.” (OR) A Trojan is a form of malicious code that can be hidden within seemingly harmless files, programs, or attachments, and can be used to compromise the security of a computer system or network and non-infected circuits. Trojan horses have been a persistent security threat for many years, and unfortunately, they are likely to remain a threat in the future. As technology advances, there will be new opportunities for trojan implementation, and as such, there will always be a need for effective detection methods.

There are several ways a Trojan can be inserted at the algorithm level. One way is to modify the source code of the algorithm directly. Another way is to introduce a Trojan during the compilation or build process. This can be done by infecting the compiler or build tools with malware, or by replacing legitimate files with Trojan-infected ones. Once the Trojan is inserted, it can be used to carry out a variety of malicious actions, such as stealing sensitive data, altering the behaviour of the algorithm, or disrupting the operation of the system.

## LIST OF FIGURES

Fig 1: FPGA Architecture	06
Fig 2: The fundamental FPGA architecture (Image Source: National Instruments)	08
Fig 3: ARTIX-7 FPGA TRAINER KIT	12
Fig 4: SLICEM	14
Fig 5: Interconnection of CLB's on ARTIX-7	16
Fig 6: DSP48E1 Slice	17
Fig 7: Classification of hardware attacks	22
Fig 8: functional diagram of ASIC/SOC	27
Fig 9: TRNG Module	28
Fig 10: Hardware Security for FPGA based systems	30
Fig 11: Bottom-Up Design	32
Fig 12: Top-Down Design	33
Fig 13: IP Encryption	47
Fig 14: Internet Key Exchange	49
Fig 15: Core design for adder/subtractor IP	51
Fig 16: Adder/Subtractor IP from IP catalogue	56

Fig 17: Designing of core IP and Customization	56
Fig 18: Generation Of Output Products	57
Fig 19: Preparing to launch the adder/subtractor IP	57
Fig 20: Simulation results before inserting the trojan for adder/subtractor IP	58
Fig 21: Simulation results after insertion of trojan for adder/subtractor IP	59

## LIST OF ABBREVIATIONS

IP-Intellectual Property	01
DDoS- Distributed Denial-Of-Service	03
FPGA- Field Programmable Gate Array	05
ASICs-Application-Specific Integrated Circuits	05
HDL-Hardware Description Languages	05
LUT-Look Up Table	07
I/O-Input Output	07
DSP-Digital Signal Processing	07
SRAM-Static Random Access Memory	08
EEPROM-Electrically Erasable Programmable Read Only Memory	08
DDR-Double Data Rate	13
USB-Universal Serial Bus	14
UART-Universal Asynchronous Receiver-Transmitter	14
JTAG-Joint Test Action Group	14

SLICE-Selective Line Insertion Communication Equipment	14
CLB-Configurable Logic Block	15
MRI (Magnetic Resonance Imaging)	18
CT (Computed Tomography)	18
AMD-Advanced Micro Devices	19
NRE-Non-recurring Engineering Costs	19
MIMO-Multiple-Input Multiple Output	19
DNA-Deoxyribo Nucleic Acid	20
VLSI-Very Large Scale Integration	21
IC-Integrated Circuit	21
TPM-Trusted Platform Modules	21
PUF-Physical Unclonable Function	21
TRNG-True Random Number Generator	21
SoC-System on Chip	27
UVM-Universal Verification Methods	33
HLS- High-Level Synthesis	33

ML-Machine Learning	34
RTL- Register Transfer Level	35
AES- Advanced Encryption Standard	48
DES-Data Encryption Standard	48
IKE -Internet Key Exchange	48
ESP-Encapsulating Security Payload	48
AH-Authentication Header	48
ISA-Internet Security Association	49
SA-Security Associations	49
VPN-Virtual Private Networks	49
VoIP- Voice over IP	50
SCCE-Synchronous Controls and Clock Enable	54
GUI-Graphical User Interface	55
AI-Artificial Intelligence	60
IoT-Internet Of Things	60

# **CHAPTER-1**

## **INTRODUCTION**

A Trojan attack is a type of malicious software, or malware, that is designed to trick users into downloading and installing it on their device. Once the Trojan is installed, it can perform various malicious activities, such as stealing sensitive information, controlling the device, or damaging the system.

The name "Trojan" is derived from the Trojan horse of Greek mythology, which was a wooden horse used by the Greeks to sneak into the city of Troy and win the war. Similarly, a Trojan attack tricks users into installing the malicious software by disguising it as legitimate software, such as a game or a security update. Trojans can be spread through a variety of methods, including email attachments, downloads from malicious websites, and software bundles. They often rely on social engineering tactics to convince users to download and install them, such as promising free software or warning of a security threat that requires immediate action. To protect against Trojan attacks, it is important to be cautious when downloading software or clicking on links, especially from unfamiliar sources. It is also essential to keep software and security systems up to date to ensure that any known vulnerabilities are patched. Finally, a reputable antivirus program can help detect and remove Trojan malware from a device.

### **PROJECT OBJECTIVE:**

The objective of Trojan detection in IP (Intellectual Property) is to identify and mitigate any potential security threats that may exist within third-party or externally sourced IP cores that are integrated into an integrated circuit design.

**Identification of IP cores:** This involves identifying all third-party or externally sourced IP cores that are used within an integrated circuit design.

**Verification of IP cores:** This step involves verifying the functionality and integrity of the IP cores, including ensuring that they do not contain any intentional or unintentional vulnerabilities or malicious circuitry.

**Trojan detection:** This step involves using various detection techniques such as static analysis, functional verification, and hardware testing to detect any potential Trojans or malicious circuitry within the IP cores.

## **BACKGROUND:**

A Trojan horse, or simply Trojan, is a type of malware that gain access to a computer system or network. Once installed, a Trojan has the ability to do different malicious activities, like stealing sensitive information, modifying files, and installing additional malware.

Trojan detection is the process of identifying and removing Trojans from a computer system. This is typically done using antivirus software, which scans the system for known Trojan signatures and behavioral patterns.

The development of Trojan detection has been ongoing since the early days of computing. As the threat landscape has evolved, so too have the methods and technologies used to detect and combat Trojans. Modern antivirus software employs advanced heuristics, behavioral analysis, and machine learning algorithms to identify and prevent Trojan infections.

Despite these advances, Trojans remain a persistent threat to computer security. Attackers continually develop new techniques for disguising and delivering Trojans, making it difficult for antivirus software to keep up.

## **MOTIVATION:**

The motivation behind hardware Trojans can vary depending on the attacker's goals. Here are a few potential motivations:

**Data theft:** Trojans can steal data in several ways. Some Trojans are designed to log keystrokes, capturing everything a user types, including login credentials and other sensitive information. Others can take screenshots or record video and audio, giving attackers a window into the user's device and activities.

**Remote control:** Remote control is another common feature of Trojan attacks. Once a Trojan infects a device, it can allow the attacker to remotely access and control the device. This can give the attacker complete access to the device's files, applications, and settings, as well as the ability to execute commands, install additional malware, or manipulate the device in other ways.

**Financial gain:** Some Trojans can be used to generate revenue for attackers through



various means, such as cryptocurrency mining or using the victim's computer resources to Launch Attacks On Other Targets.

**Espionage:** Hardware Trojans can be used to steal sensitive information from a targeted system, such as encryption keys, intellectual property, or classified data.

**Sabotage:** Attackers can use hardware Trojans to cause malfunctions or disrupt the normal operation of a system. This can be done for various reasons, such as to damage a competitor's products or to create chaos in critical infrastructure

**Financial gain:** Hardware Trojans can be used to generate revenue through various means, such as mining cryptocurrency, stealing financial information, or hijacking computing resources for Distributed Denial-Of-Service (DDoS) attacks

**Political or ideological motives:** State-sponsored attackers may use hardware Trojans to achieve political or military objectives, such as disrupting the operations of a rival state's military or critical infrastructure.

**Revenge:** A disgruntled employee or contractor may plant a hardware Trojan as a form of retaliation against their employer or client.

Hardware Trojans can be difficult to detect and remove, making them an attractive tool for attackers with these various motivations. Therefore, it is important to employ effective security measures and regularly audit and test hardware components to identify potential Trojans.

### **Project Outline:**

At a high level, Trojan insertion involves modifying the design or implementation of an algorithm in a way that allows for the insertion of malicious code. This can be done by an attacker with access to the design or implementation process, such as a rogue employee or a third-party vendor.

The insertion of Trojans at the algorithm level can be difficult to detect, as the Trojans are embedded in the algorithm itself and may not be apparent through traditional security measures. Once the Trojan is inserted, it can be activated remotely by the attacker, allowing them to carry out a variety of malicious activities, such as stealing sensitive information, manipulating data, or causing system failures.

To protect against Trojan insertion at the algorithm level, it is important to implement a strong security protocol throughout the design and implementation process, including background checks for employees and vendors, multi-layered verification and testing, and ongoing monitoring for unusual activity. It is also essential to implement strong access controls and encryption to limit the impact of any successful Trojan attacks. Finally, it is important to have a comprehensive incident response plan in place to quickly detect and respond to any suspected Trojan attacks.

## **CHAPTER-2**

### **INTRODUCTION TO FPGA**

FPGA stands for Field Programmable Gate Array. It is a type of integrated circuit that can be programmed to perform specific functions after manufacturing. Unlike traditional application-specific integrated circuits (ASICs), which are designed for specific tasks and cannot be reprogrammed, FPGAs can be reprogrammed and customized for different tasks.

FPGAs consist of an array of logic blocks and interconnects that can be configured to perform a wide range of functions, from simple logic gates to complex digital signal processing tasks. They are often used in applications where fast processing and low latency are critical, such as in high-performance computing, telecommunications, and digital signal processing.

FPGAs can be programmed using hardware description languages (HDLs), such as VHDL or Verilog, which describe the desired behavior of the circuit. Once the HDL code is compiled and synthesized, it can be programmed onto the FPGA using specialized programming tools.

One of the key advantages of FPGAs is their flexibility and reprogrammability, which allows for rapid prototyping and customization. They can also be used to implement complex algorithms and hardware accelerators that may be difficult or impossible to implement on a traditional processor.

FPGAs can have a significant impact on hardware security, both positively and negatively. On the positive side, FPGAs can be used to implement custom security features and hardware accelerators, such as encryption/decryption modules, secure boot managers, and key generation modules. These features can be customized and optimized for specific applications, providing a high level of security and performance.

Additionally, FPGAs can be used to implement security through obscurity, by encoding the hardware design in a way that is difficult to reverse engineer or modify. This can make it more difficult for attackers to discover and exploit vulnerabilities in the design.

On the negative side, FPGAs can be vulnerable to a variety of security threats, such as side-channel attacks, Trojan insertions, and reverse engineering. Side-channel attacks involve

exploiting unintended leaks of information, such as power consumption or electromagnetic radiation, to extract secret information. Trojan insertions involve the insertion of malicious code into the FPGA design, which can be used to steal sensitive information or manipulate the system. Reverse engineering involves disassembling the FPGA design to discover the underlying logic and functionality, which can be used to create a replica of the system or exploit vulnerabilities.

To mitigate these risks, it is important to implement strong security measures throughout the design and implementation process, such as strong access controls, encryption, and verification/testing. Additionally, it is important to monitor the FPGA for unusual activity, such as unexpected power consumption or unauthorized access, and to have a comprehensive incident response plan in place to quickly detect and respond to any suspected security breaches.

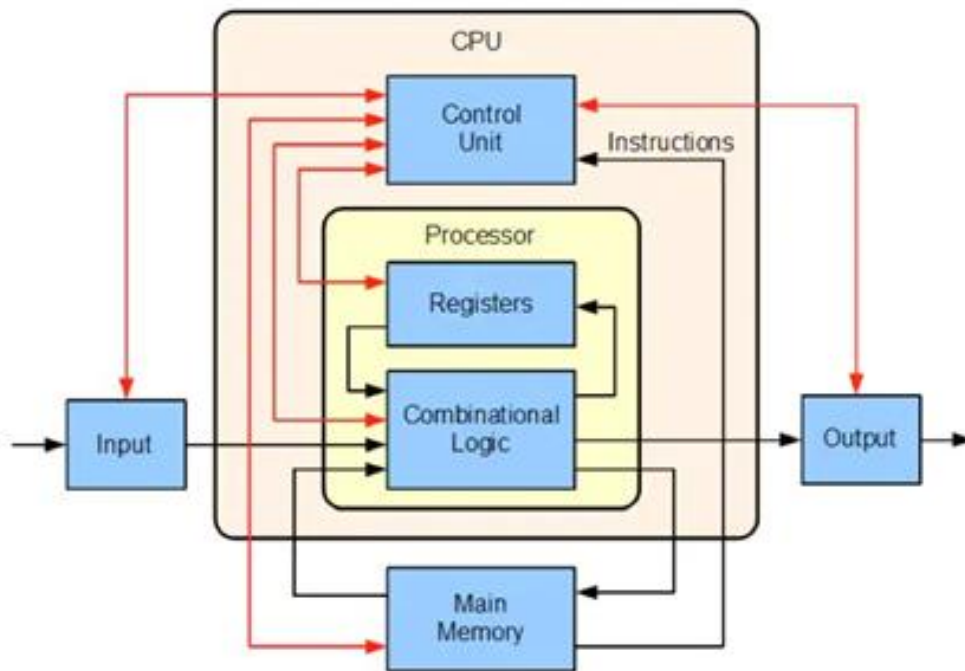


Fig 1: FPGA Architecture

## **FPGA ARCHITECTURE:**

- The architecture of an FPGA typically consists of three main components: programmable logic blocks, input/output blocks, and interconnect resources.
- Programmable logic blocks are the building blocks of an FPGA and are used to implement digital circuits.
- These blocks consist of lookup tables (LUTs) and flip-flops, which can be configured to implement different types of logic functions.
- The LUTs can also be programmed to implement memory elements, such as registers and counters.
- Input/output (I/O) blocks provide the interface between the FPGA and the outside world. These blocks typically include input buffers, output drivers, and configurable I/O standards, which can be used to interface with a wide range of different external devices.
- Interconnect resources provide the means for connecting the programmable logic blocks and I/O blocks together.
- These resources typically consist of a network of programmable routing channels and switch boxes, which can be configured to provide the desired connectivity between different blocks.
- In addition to these main components, modern FPGAs may also include additional resources, such as digital signal processing (DSP) blocks, memory blocks, and embedded processors.
- These resources can be used to implement more complex functionality and to offload processing from the main processor.
- Overall, the architecture of an FPGA is highly flexible and configurable, allowing for a wide range of different designs and applications.
- However, designing and programming FPGAs can also be more complex and time-consuming than traditional processors, and may require specialized skills and knowledge.

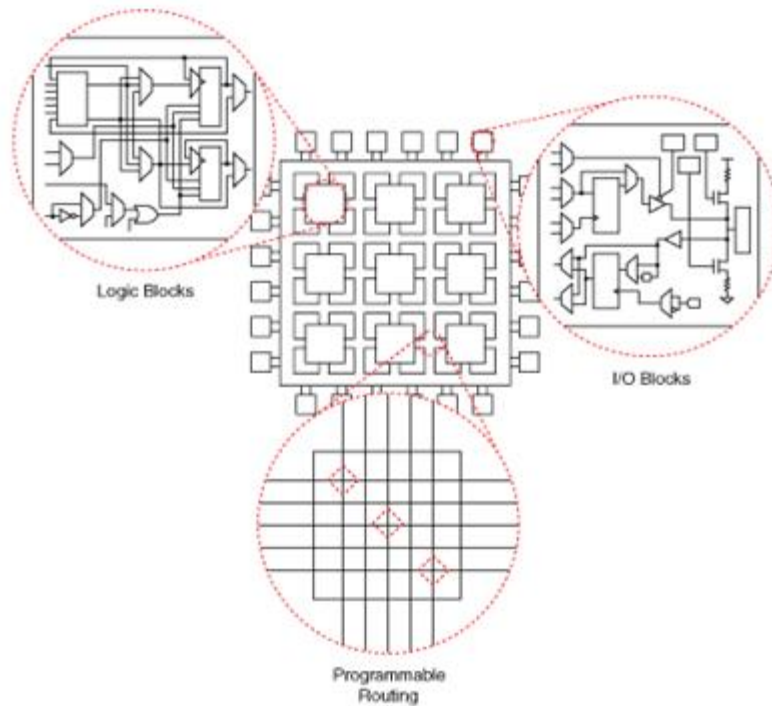


Fig 2: The fundamental FPGA architecture (Image Source: National Instruments)

**TPYES OF FPGA:**

The three types of FPGAs are static RAM (SRAM), anti-fuses, and flash EPROM.

**SRAM FPGA:**

In an SRAM-based FPGA, the logic cells and routing resources are implemented using a network of configurable SRAM cells. The configuration data that defines the logic cells and routing is stored in an external memory chip, typically a Flash memory or EEPROM.

When an SRAM-based FPGA is powered up, the configured data will be uploaded into the SRAM cells from the external memory chip, and the FPGA becomes operational. The logic cells and routing can be reconfigured by loading a different configuration bitstream into the SRAM cells.

### **ADVANTAGES:**

- High flexibility: SRAM-based FPGAs can be reprogrammed as needed, making them ideal for prototyping, development, and low-volume production.
- High performance: SRAM-based FPGAs can support high clock frequencies and low-latency operation, making them suitable for high-speed applications such as signal processing and digital communications.
- Easy testing and debugging: SRAM-based FPGAs can be easily reprogrammed to enable testing and debugging of different logic and routing configurations.

### **DISADVANTAGES:**

- Configuration volatility: SRAM-based FPGAs require configuration data to be loaded on power-up or after any configuration change. This can make them more susceptible to configuration errors or unintended changes.
- Configuration security: The configuration data for an SRAM-based FPGA can be read out and potentially reverse-engineered, making them less secure for certain applications.
- Power consumption: SRAM-based FPGAs require a continuous power supply to maintain their configuration, which can lead to higher power consumption compared to other types of FPGAs such as Flash-based or Antifuse-based FPGAs.

### **Anti fuse FPGA:**

An antifuse FPGA is a type of FPGA, that uses antifuses as the configuration storage element. Antifuses are non-conductive structures that can be programmed to become conductive, typically by applying a high voltage. In an antifuse FPGA, the logic cells and routing resources are implemented using a network of antifuses.

When an antifuse FPGA is manufactured, the antifuses are programmed to form the desired logic cells and routing. Once programmed, the antifuses are permanently set and cannot be reprogrammed. This means that antifuse FPGAs are not reconfigurable, but they offer several advantages over other types of FPGAs

### **ADVANTAGES:**

- High reliability: Antifuse FPGAs are less susceptible to configuration errors or unintended changes since the programming is permanent.
- Low power consumption: Antifuse FPGAs require very low power to maintain their configuration, since the programming is non-volatile.
- High security: Since the programming is permanent, antifuse FPGAs offer a high level of security and protection against reverse engineering.
- High performance: Antifuse FPGAs can support high clock frequencies and low-latency operation, making them suitable for high-speed applications.

### **DISADVANTAGES:**

- High cost: Antifuse FPGAs are more expensive to manufacture than other types of FPGAs.
- Limited flexibility: Antifuse FPGAs cannot be reprogrammed, which limits their flexibility and adaptability to changing requirements.
- Limited testing and debugging: Antifuse FPGAs cannot be easily reprogrammed for testing and debugging, which can make development and prototyping more challenging.

Overall, antifuse FPGAs are well-suited for applications that require high reliability, low power consumption, and high security, such as aerospace and defense systems, but may not be the best choice for applications that require flexibility and adaptability to changing requirements.

### **Flash FPGA:**

Flash-based FPGAs (Field Programmable Gate Arrays) use flash memory as the configuration storage element. In a flash-based FPGA, the logic cells and routing resources are implemented using a network of configurable memory cells, such as SRAM or antifuse cells. The configuration data that defines the logic cells and routing is stored in a flash memory chip, which is typically located on the same chip as the FPGA.

When a flash-based FPGA is powered up, the configured data is uploaded from the flash memory chip into the memory cells, and the FPGA becomes operational. The logic cells and



routing can be reconfigured by loading a different configuration bitstream into the flash memory chip.

#### **ADVANTAGES:**

- High flexibility: Flash-based FPGAs can be reprogrammed as needed, making them ideal for prototyping, development, and low-volume production.
- Low power consumption: Flash-based FPGAs require very low power to maintain their configuration, since the configuration data is stored in non-volatile flash memory.
- High reliability: Flash-based FPGAs offer high reliability since the configuration data is stored in non-volatile flash memory, which is less susceptible to configuration errors or unintended changes.
- High security: Flash-based FPGAs can offer a high level of security and protection against reverse engineering since the configuration data is stored in non-volatile flash memory.
- Easy testing and debugging: Flash-based FPGAs can be easily reprogrammed to enable testing and debugging of different logic and routing configurations.

#### **DISADVANTAGES:**

- Lower performance: Flash-based FPGAs typically have lower performance compared to other types of FPGAs, such as SRAM-based FPGAs, due to the overhead of accessing the flash memory.
- Limited endurance: Flash memory has limited write endurance, which can be a concern for applications that require frequent reconfiguration.
- Limited scalability: Flash-based FPGAs have limited scalability due to the size and cost of the flash memory required to store the configuration data.

Overall, flash-based FPGAs are well-suited for applications that require flexibility, low power consumption, and high reliability, but may not be the best choice for applications that require high performance or frequent reconfiguration.

#### **FPGA FAMILIES:**

##### **SRAM FPGA Families**

- Altera Stratix II and Cyclone II families
- Atmel AT6000 and AT40K families
- Lattice LatticeEC and LatticeECP families
- Xilinx Spartan-3 and Virtex-4 families

### **Antifuse FPGA Families**

- Actel SX and Axcelerator families
- Quicklogic Eclipse II family

### **Flash FPGA Families**

- Actel ProASIC family

### **Hybrid flash/SRAM FPGA Families**

- Lattice LatticeXP family

### **ARTIX-7 FPGA TRAINER KIT:**

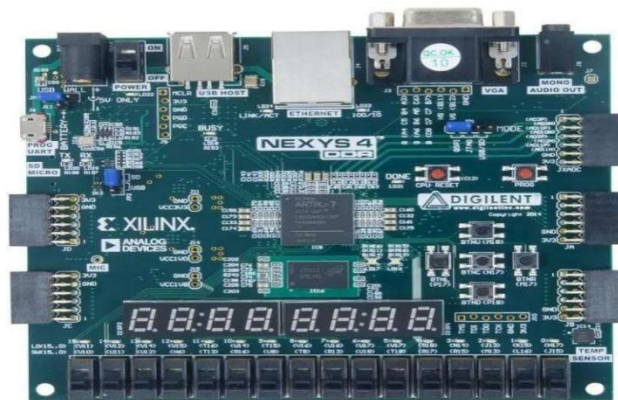


Fig 3: ARTIX-7 FPGA TRAINER KIT

An Artix-7 FPGA trainer is a development board designed to help students and engineers learn and experiment with FPGA technology. The Artix-7 FPGA is a popular FPGA chip from Xilinx, which offers a balance of performance, power consumption, and cost.

The Artix-7 FPGA trainer typically includes the following features:

- Artix-7 FPGA chip: The board includes an Artix-7 FPGA chip, which provides a configurable logic fabric that can be used to implement digital circuits.

- IO interfaces: The board includes a variety of IO interfaces, such as switches, buttons, LEDs, and displays, which can be used to interact with the FPGA and test different digital circuits.
- Memory interfaces: The board may include memory interfaces, such as DDR3 or DDR4 memory, which can be used to store and retrieve data from the FPGA.
- Communication interfaces: The board may include communication interfaces, such as USB, Ethernet, or UART, which can be used to communicate with other devices or computers.
- Power supply: The board includes a power supply circuit, which provides the necessary voltage levels to power the FPGA and other components on the board.
- Programming interface: The board includes a programming interface, such as JTAG or USB, which can be used to program the FPGA with a bitstream generated by a software tool, such as Xilinx Vivado.
- Users can use the kit to learn about FPGA architecture, HDL programming (Verilog or VHDL), digital circuits design, FPGA-based project development, and more.
- This also supports Xilinx Vivado Design Suite, which is a comprehensive development environment for FPGAs.
- Overall, the ARTIX-7 FPGA trainer kit is a useful tool for anyone interested in learning about FPGAs and developing FPGA-based projects. Some of the major applications of ARTIX-7 FPGA trainer kit include:
  - Digital Signal Processing (DSP): The Artix-7 FPGA trainer kit can be used to implement various DSP algorithms such as filtering, Fourier transforms, and digital modulation/demodulation. This makes it a valuable tool for students and researchers studying signal processing and communications.
  - Robotics: The Artix-7 FPGA trainer kit can be used to design and control robotic systems. FPGAs can be used to implement real-time control systems with high accuracy and low latency, which is critical for robotic applications.
  - Image and Video Processing: FPGAs are ideal for image and video processing applications due to their high parallelism and ability to process data in real-time. The Artix-7 FPGA trainer kit can be used to implement various image and video processing algorithms.

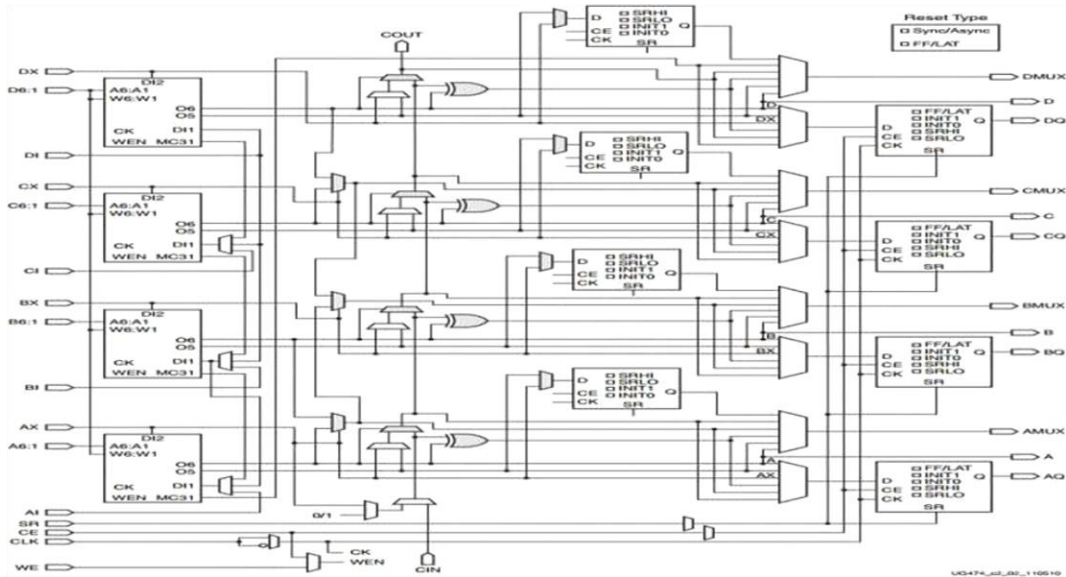


Fig 4: SLICEM

In the diagram above, three major SLICEM sub-systems can be seen:

1. The four 6-input LUTs,
2. The eight flip-flops, and
3. The fast carry logic.

These subsystems will be discussed in the subsequent sections.

### 1. Look-up tables

- The SLICEMs are organized into columns and rows, and are interconnected using configurable routing resources, such as multiplexers, buffers, and interconnect wires. The routing resources allow the SLICEMs to be connected in various configurations, enabling the implementation of complex digital circuits.
- The SLICEMs in Xilinx FPGAs also include additional features, such as carry logic for implementing adders and subtractors, and dedicated multiplexers for implementing arithmetic functions.
- The LUTs in a SLICEM are the fundamental building blocks of Xilinx FPGAs and are mostly used in digital circuits, from basic combinational logic to complex state machines and processors.
- The SLICEM architecture allows designers to optimize the performance, power consumption, and area utilization of their designs, and provides a high degree of flexibility and configurability.

## 2. Flip Flops

- Flip-flops are fundamental building blocks of FPGA (Field Programmable Gate Array) designs. They are used to implement sequential logic circuits that store and transmit binary data.
- FPGA flip-flops also have additional features, such as asynchronous and synchronous reset, preset, enable, and clock gating. These features can be used to implement more complex logic circuits and optimize the performance and power consumption of the design.
- The number of flip-flops available in an FPGA depends on the specific FPGA architecture and the size of the design. Modern FPGAs contains tens or thousands or even millions flip-flops, which can be interconnected using configurable routing resources to implement complex digital circuits.
- The number and type of ff's used in an FPGA design depend on the design requirements, such as speed, power consumption, and area utilization.

## 3. Fast Carry Logic

- This is a technique used in digital circuits to implement fast and efficient adders and subtractors. It is commonly used in FPGA designs to implement arithmetic circuits that perform addition and subtraction operations.
- In FPGA designs, Fast Carry Logic is implemented using dedicated carry chains that can be configured to implement carry propagation for addition and borrow propagation for subtraction.
- The carry chains are implemented using configurable logic blocks that are configured to verify different types of carry chains depending on the specific FPGA architecture and design requirements.
- One of the advantage of Fast Carry Logic is it is used for fast and efficient addition and subtraction operations with a small area footprint.
- This is achieved by minimizing the number of logic levels and interconnect delays in the carry chain, which reduces the overall delay and power consumption of the circuit.
- In addition, Fast Carry Logic can be optimized for different design requirements, such as speed, power consumption, and area utilization, by adjusting the configuration of the carry chain. This makes it a flexible and versatile technique for FPGA designs.

## Interconnect:

- Interconnect in configurable logic blocks (CLBs) is a critical component of FPGA designs. The interconnect allows for the routing of signals between different CLBs and remaining components of the FPGA, such as I/O and memory blocks.
- In an FPGA, the interconnect is implemented using programmable routing resources, such as wires, multiplexers, buffers, and programmable switches. These resources can be configured to implement various types of routing topologies, such as point-to-point, bus-based, and tree-based routing.
- The interconnect resources in CLBs are used to link the I/O's of the CLBs with other components of the FPGA. For example, the inputs and outputs of the LUTs (Lookup Tables) in a CLB are connected to the interconnect resources, which can be used to route signals to other CLBs or I/O blocks.
- The interconnect in CLBs can also be used to implement other types of logic functions, such as carry logic for adders and subtractors, and multiplexers for implementing arithmetic functions. This allows for the implementation of complex digital circuits using a combination of CLBs and interconnect resources.
- The interconnect in CLBs is a key factor in determining the consumption, and area utilized FPGA designs. It is important to optimize the interconnect for the specific design requirements to achieve the best performance and efficiency. This can be achieved by carefully configuring the

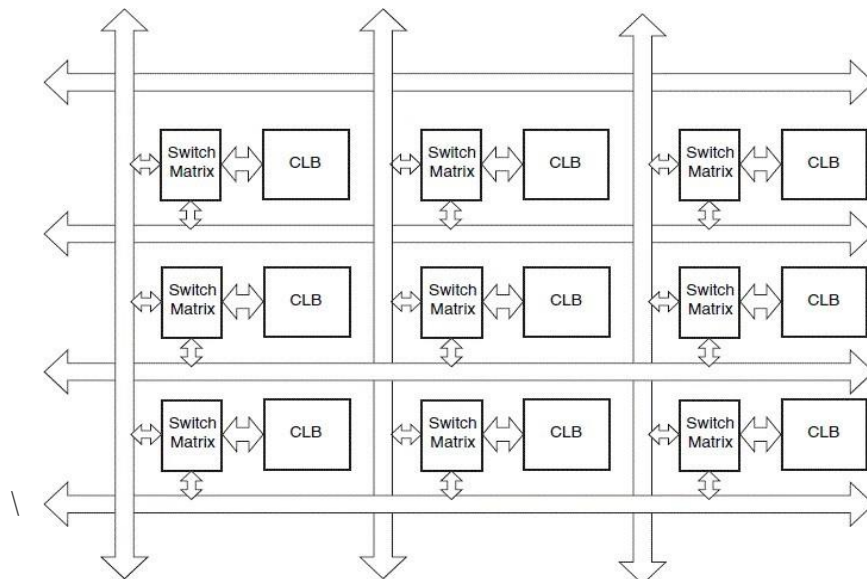
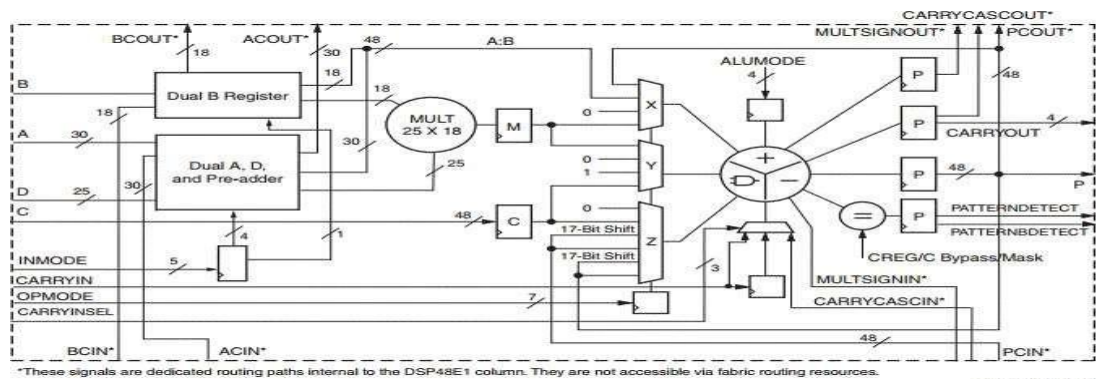


Fig 5: Interconnection of CLB's on ARTIX-7

## DSP Slice

- DSP (Digital Signal Processing) slices are specialized blocks in FPGA (Field Programmable Gate Array) designs that are used to implement complex arithmetic functions commonly used in signal processing applications.
- A DSP slice typically consists of dedicated multiplier and adder circuits that are optimized for high-speed multiplication and accumulation operations. The multiplier and adder circuits are often pipelined to improve the performance and reduces the delay.
- In addition to the multiplier and adder circuits, DSP slices can also include dedicated registers, pre-adders, post-adders, and other circuit elements that are optimized for signal processing applications. These circuit elements can be configured to implement various types of arithmetic functions.
- DSP slices are commonly used in FPGA designs for a wide range of applications, including audio and video processing, wireless communication, radar and sonar systems, and other applications that require high-speed and efficient signal processing.
- The number of DSP slices available in an FPGA depends on the specific FPGA architecture and the size of the design. Modern FPGAs can contain hundreds or even thousands of DSP slices, which can be interconnected using configurable routing resources to implement complex signal processing circuits.
- Overall, DSP slices are a key component of many FPGA designs and play a critical role in implementing high-performance and efficient signal processing systems.

Fig 6: DSP48E1 Slice



## **APPLICATIONS OF FPGA:**

- **ASIC Prototype-** To create an ASIC prototype in an FPGA, you first need to synthesize and simulate the ASIC design using a hardware description language (HDL) such as Verilog or VHDL. Then, you need to convert the design to a format that can be programmed onto the FPGA, such as a bitstream file.
- **Medical-** FPGAs are used in medical imaging applications such as MRI (Magnetic Resonance Imaging), CT (Computed Tomography) scanning, and ultrasound. FPGAs can be programmed to handle the high-speed data transfer and processing required for medical image acquisition and processing.
- **Digital signal processing (DSP):** FPGAs can perform complex DSP algorithms at high speeds and making them ideal for applications such as audio and video processing, wireless communication, and radar signal processing.
- **High-performance computing (HPC):** FPGAs can be used to accelerate computations in HPC like scientific simulations.
- **Embedded systems:** FPGAs can be used to implement custom logic in embedded systems, such as digital cameras, automotive electronics, and medical devices.
- **Network processing:** FPGAs can be used in routers, switches, and other network equipment to perform packet processing and protocol handling at high speeds.
- **Aerospace and defense:** FPGAs can be used in avionics systems, including flight control systems, navigation systems, and mission computers. They can provide high-speed processing capabilities and can be programmed to handle different tasks simultaneously.
- **Video and image processing:** FPGAs can be used in video compression and decompression applications. They can perform real-time encoding and decoding of video streams, making them ideal for applications such as video surveillance, video conferencing, and video streaming.
- **Cryptography and security:** FPGAs can be used to implement hardware-based cryptography and security functions, such as encryption, decryption, and authentication



- Internet of Things (IoT): FPGAs can be used in IoT applications to implement custom logic and to process data at the edge of the network, reducing the need for data transmission and improving overall system performance.
- These are just a few examples of the many applications of FPGAs. With their flexibility, high performance, and low power consumption, FPGAs are an important technology for many different industries and applications.
- Video and Image Processing - AMD FPGAs and targeted design platforms enable higher degrees of flexibility, faster time-to-market, and lower overall non-recurring engineering costs (NRE) for a wide range of video and imaging applications.
- Wired Communications -FPGAs can be used in Ethernet applications to perform real-time data processing and packet handling. They can be programmed to support different Ethernet standards, including 10/100/1000 Mbps and 10Gbps.
- Wireless communications - FPGAs can be used in 5G networks to process high-speed data in real-time. They can be programmed to handle advanced features such as beamforming, massive MIMO, and MicroWave communications.

### **HISTORY OF FPGA:**

- They have a relatively short history compared to other electronic components, such as transistors and integrated circuits.
- The XC2064 was quickly followed by the XC2018, which had 18 inputs and 4 outputs per logic block, and the XC3090, which had 90 logic blocks.
- In the 1990s, FPGAs became more popular and widely used in the electronics industry. Xilinx introduced the Virtex series of FPGAs in 1998, which were the first to use a segmented routing architecture and multiple voltage domains.
- Other FPGA manufacturers, such as Altera and Actel, also entered the market and introduced their own families of FPGAs.
- In the 2000s, FPGAs continued to evolve and become more powerful and flexible. The introduction of the Spartan and Virtex-II series of FPGAs by Xilinx in 2000 and 2002, respectively, marked a significant advancement in the density and performance of FPGAs.

- They are also used in research and development for prototyping and testing of new hardware designs, and in high-performance computing for applications such as datacenter acceleration and machine learning.
- FPGA technology continues to evolve and improve, with the latest FPGAs featuring multi-gigabit transceivers, high-speed serial interfaces, and sophisticated routing and placement algorithms.

### **Modern Usage of FPGA:**

FPGAs have been increasingly used in modern science to accelerate computationally intensive tasks in fields such as physics, astronomy, biology, and machine learning. Some recent examples of FPGA usage in modern science include:

**Particle physics experiments:** Large-scale experiments like the Large Hadron Collider at CERN require massive amounts of data processing in real-time. FPGAs have been used to accelerate the processing of data from particle detectors, improving the speed and efficiency of the experiments.

**Astronomy:** FPGAs have been used in radio telescopes to accelerate signal processing and reduce data bandwidth. For example, the Square Kilometer Array (SKA), a next-generation radio telescope, is planning to use FPGAs for real-time signal processing of astronomical data.

**Genomics:** FPGAs have been used in DNA sequencing to accelerate the alignment and analysis of genomic data. This has enabled faster and more accurate sequencing of DNA, leading to advances in personalized medicine and disease diagnosis.

**Machine learning:** FPGAs have become increasingly popular for accelerating machine learning algorithms, particularly for inference tasks such as object recognition and speech processing. FPGAs can provide high-performance, low-latency processing for these tasks, making them suitable for real-time applications such as autonomous vehicles and robotics.

Overall, FPGAs offer a flexible and scalable platform for accelerating computationally intensive tasks in modern science, and are expected to play an increasingly important role in scientific research and development in the coming years.

## CHAPTER 3

### HARDWARE SECURITY

#### INTRODUCTION:

- Hardware security in Very Large Scale Integration (VLSI) is an important aspect in ensuring the protection of electronic devices against different types of attacks. Hardware security refers to the techniques and methods used to protect integrated circuits (ICs) and electronic systems from malicious attacks, which can include intellectual property theft, reverse engineering, and tampering.
- There are different techniques used to achieve hardware security in VLSI. They are active, passive techniques.
- Active hardware security techniques include measures that are designed to prevent attackers from exploiting vulnerabilities in the system. These techniques include encryption, obfuscation, and hardware-based security mechanisms like Trusted Platform Modules (TPMs), secure boot, and secure enclaves.
- Passive hardware security techniques include measures that are designed to detect and respond to attacks. These techniques include hardware-based monitoring and debugging, side-channel analysis, and fault injection analysis.
- One important aspect of hardware security in VLSI is the random number generators for generating unique identifiers or keys for each device. PUFs use the inherent variability in the manufacturing process to generate unique identifiers, while TRNGs generate random numbers using a physical process that is inherently unpredictable.
- In addition, VLSI designers can use other techniques such as layout randomization, gate-level obfuscation, and diversity to enhance the security of their designs.

## Different types of Attacks on Hardware:

There are several types of attacks that can be used to compromise the security of hardware, including:

**Side-channel attacks:** By this, an attacker might analyze the power consumption, electromagnetic radiation, or timing behavior of a device to extract information about its secrets or cryptographic keys.

**Fault injection attacks:** Fault injection attacks involve inducing errors in a device's operation in order to cause it to behave in unexpected ways. For example, an attacker might use voltage or clock glitches to cause a device to produce incorrect results or to bypass security measures.

**Reverse engineering attacks:** Reverse engineering attacks involve analyzing the physical structure of a device in order to extract information about its secrets or functionality. For example, an attacker might use microscopy, x-ray analysis, or chemical etching to inspect the physical structure of a device and identify its components and interconnections.

**Tampering attacks:** Tampering attacks involve physically modifying a device in order to bypass its security measures or extract its secrets. For example, an attacker might use acid or laser cutting to remove the protective coatings on a device and access its internal components. These are just a few examples of the types of attacks that can be used to compromise the security of hardware. Protecting hardware from such attacks requires a combination of design techniques, testing, and countermeasures, and can involve these of hardware security features such as secure boot, secure key storage, and secure communication protocols.

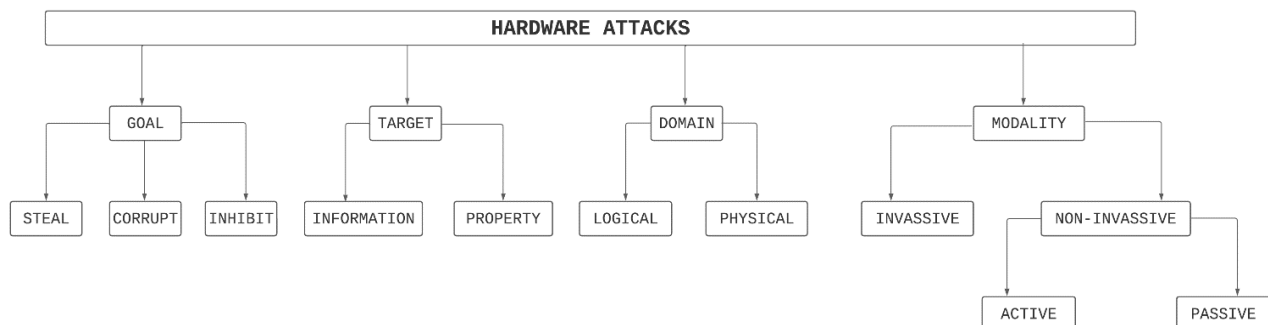


Fig 7: Classification of hardware attacks

## Hardware Trojan Attacks:

A hardware Trojan attack is a type of attack in which a malicious modification is inserted into the design of a hardware device during the design or manufacturing process. Hardware Trojan attacks can be difficult to identify because they are often inserted at the gate-level and can be designed to be triggered only under certain conditions. Furthermore, hardware Trojans can be designed to be dormant for long periods of time, making it difficult to identify them using traditional testing techniques.

Hardware Trojan attacks are categorised as: design-time Trojans and post-manufacturing Trojans.

**Design-time Trojans** are inserted during the design phase of a device, and may be inserted by an insider or by a third party who has access to the design files.

**Post-manufacturing Trojans** are inserted during the manufacturing process, and may be inserted by a malicious manufacturer or by a third party that has access to the device before it is deployed in the field.

## Types of Trojans:

Here are some common types of Hardware Trojans:

**Logic manipulation Trojans:** They can Trojans modify the design of a device by inserting additional gates or modifying the functionality of existing gates.

**Data alteration Trojans:** These Trojans modify the data stored in a device's memory, registers, or other storage locations. They can be used to corrupt or destroy data, to modify the behavior of a device, or to leak sensitive information.

**Clock manipulation Trojans:** These Trojans modify the timing or frequency of a device's clock signal. They can be used to disrupt the operation of a device, to cause the device to fail, or to perform some other unauthorized activity.

**Power manipulation Trojans:** These Trojans modify the power consumption of a device by introducing additional power-consuming circuitry or by modifying the behavior of existing circuitry.

**Stealthy Trojans:** These Trojans are designed to be difficult to detect and can be activated by a specific trigger, such as a particular sequence of inputs or a certain environmental condition.

**False functionality Trojans:** These Trojans are designed to add a new function to a device that is not disclosed to the user or the system designer. They can be used to perform unauthorized activities or to gain access to sensitive information.

**Activation-based Trojans:** These Trojans are dormant until a specific event or condition occurs, such as a particular sequence of inputs or the passage of time. They can be used to perform unauthorized activities, to leak sensitive information, or to disable the device.

- These are just a few examples of the many types of trojans that can be designed. As the field of hardware security continues to evolve, new types of Hardware Trojans are likely to emerge, and new countermeasures will need to be developed to address these threats.

**Counter measures against Trojan Attacks:**

- There are several countermeasures that can be used to reduce the risk of hardware Trojan attacks:

**Design-time countermeasures:** These countermeasures are aimed at preventing hardware Trojans from being inserted into a device during the design phase. Some examples of design-time countermeasures include:

- Using secure design practices to minimize the risk of insider attacks and to ensure the integrity of the design files.
- Using formal verification techniques to detect hardware Trojans during the design phase.
- Using hardware diversity to make it more difficult for attackers to insert hardware Trojans that are common across multiple devices.

**Manufacturing-time countermeasures:** These countermeasures are aimed at preventing hardware Trojans from being inserted into a device during the manufacturing process. Some examples of manufacturing-time countermeasures include:

- Using trusted foundries and supply chains to minimize the risk of hardware Trojans being introduced during the manufacturing process.

- Using physical inspection techniques, such as X-ray analysis, to inspect the physical structure of a device and identify any signs of tampering or malicious modifications.

**Post-manufacturing countermeasures:** These countermeasures are aimed at detecting and mitigating hardware Trojans that are introduced after a device has been deployed in the field. Some examples of post-manufacturing countermeasures include:

- Performing regular security audits to detect any signs of tampering or malicious modifications.
- Using side-channel analysis to detect any unusual behavior or information leaks.
- Using software-based countermeasures, such as intrusion detection systems or security software, to detect any unusual behavior or to protect against data leaks.
- Overall, the most effective approach to countering hardware Trojan attacks involves a combination of these countermeasures. By incorporating these countermeasures into the design, manufacturing, and deployment of electronic systems and devices, it is possible to reduce the risk of trojan attacks and to protect from other attacks.

#### **Active and passive techniques in Hardware security :**

- Active and passive techniques are two broad categories of hardware security measures that can be employed to protect integrated circuits (ICs) and electronic systems against different types of attacks.

#### **Active Hardware Security Techniques:**

- Active hardware security techniques are designed to prevent attackers from exploiting vulnerabilities in the system. These techniques include:

**Encryption:** This technique involves encoding data in such a way that it can only be accessed by authorized parties who have the necessary decryption keys. Encryption can be applied to both data at rest and in transit.

**Obfuscation:** This technique involves intentionally making the design of a system or component more difficult to understand. This can include things like adding extra logic gates, randomizing the layout of the circuit, and making the control flow more complex.

**Hardware-Based Security Mechanisms:** These are hardware-based security features that can provide enhanced security for the system. Examples include Trusted Platform Modules (TPMs), secure boot, and secure enclaves. These mechanisms can provide hardware-based authentication, encryption, and access control.

## Passive Hardware Security Techniques:

- Passive hardware security techniques are designed to detect and respond to attacks. These techniques include:

**Hardware-Based Monitoring and Debugging:** This technique involves using hardware-based monitoring and debugging tools to detect and analyze system activity. These tools can help identify abnormal behavior that may be indicative of an attack.

**Side-Channel Analysis:** This technique involves analyzing the electromagnetic radiation, or other side-channel signals generated by a system.

**Fault Injection Analysis:** This technique involves intentionally injecting faults or errors into a system to test its security or to determine how it will behave in the event of an attack.

- In practice, a combination of active and passive techniques is often used to provide a comprehensive hardware security solution. By employing a variety of techniques, designers can build more secure and trustworthy electronic systems that are better able to withstand different types of attacks.

## Use of PUFs and TRNGs in Hardware Security:

- Physically Unclonable Functions and True Random Number Generators are two important techniques used in hardware security to enhance the protection of electronic devices against various types of attacks.
- PUFs are hardware circuits that exploit the unique manufacturing variations that occur in electronic devices to create a uniqueness for each device. PUFs generate a unique response to a given challenge by utilizing the natural randomness and variation in the manufacturing process of ICs. These unique identifiers can be used for security purposes, and device authentication, secure key storage, and secure communication.
- PUFs have several advantages over traditional security techniques such as software-based security and traditional hardware-based authentication mechanisms. They are difficult to clone or replicate, as the unique identifier is based on the manufacturing variations of the device.



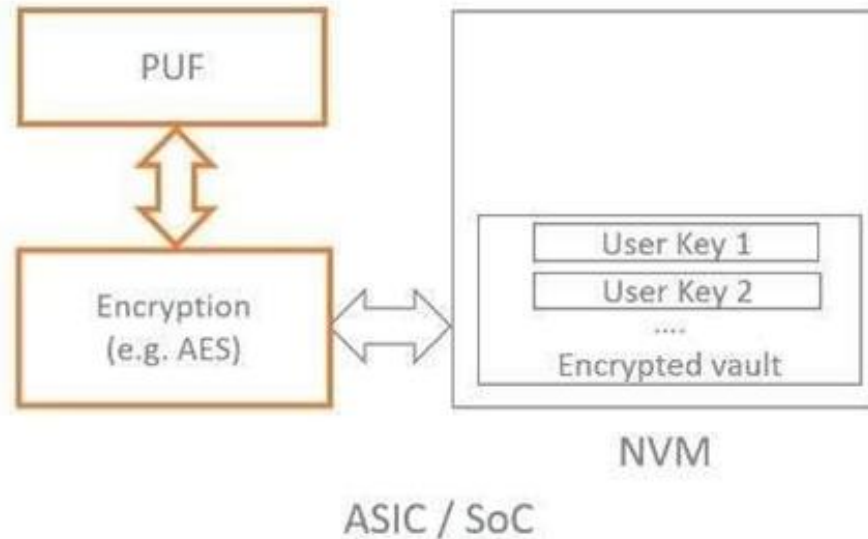


Fig 8: functional diagram of ASIC/SoC

- TRNGs, on the other hand, are hardware circuits that generate random numbers using a physical process that is inherently unpredictable. TRNGs exploit the natural randomness found in physical systems to generate truly random numbers that cannot be predicted or reproduced.
- TRNGs are an important building block in many security protocols, as they are used to generate random numbers for cryptographic purposes such as key generation and initialization vectors.
- A TRNG generates truly random numbers that are not influenced by any external factors or biases, and are thus more secure than pseudo-random numbers generated by software-based methods.
- In summary, PUFs and TRNGs are important hardware security techniques that can be used to enhance the security of electronic devices. By exploiting the inherent randomness and variation in ICs, PUFs and TRNGs provide a more secure and trustworthy foundation for electronic systems.

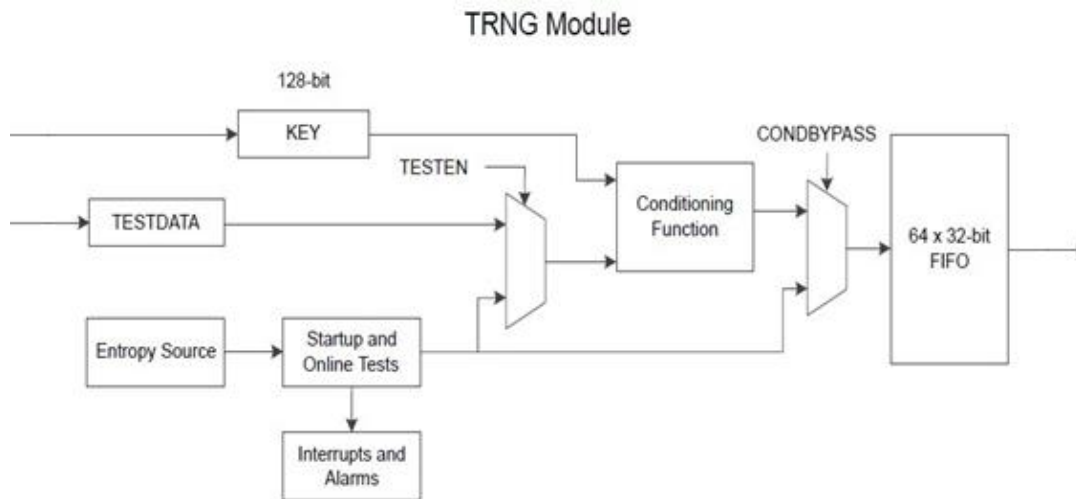


Fig 9: TRNG Module

### Randomization:

- Randomization is a hardware security technique that involves introducing randomness into the design of electronic circuits to make them more difficult to analyze or attack.

**Circuit-level randomization** involves adding additional circuitry to a design to make it more complex and difficult to analyze. For example, a designer may add additional gates to a circuit that are not strictly necessary for its function, but that make it more difficult to analyze the circuit and extract its secrets. This technique is often referred to as "logic locking."

**Gate-level randomization** involves changing the logic of individual gates in a design to make them more difficult to analyze. For example, a designer may swap the input and output signals of a gate, or change the timing of the inputs to make it more difficult to reverse-engineer the circuit.

**Layout-level randomization** involves changing the physical layout of a circuit to make it more difficult to analyze. For example, a designer may randomly place gates in a circuit layout to make it more difficult to identify the function of individual gates. This technique is often referred to as "layout obfuscation."

- Randomization can also be applied to the operation of electronic devices themselves, for example, by varying the timing of certain operations or by introducing random

delays into the processing of data, which involves analyzing the timing or power consumption of a device to extract sensitive information.

- Overall, randomization is a powerful hardware security technique that can make it more difficult for attackers to analyze and attack electronic devices. By introducing randomness into the design and operation of circuits, designers can create more secure and trustworthy electronic systems that are more resilient to attacks.

**Gate Substitution:** This technique involves replacing standard gates (such as AND, OR, and NOT gates) with equivalent, but more complex, gate structures. For example, a designer might replace an AND gate with a two-input XOR gate followed by a two-input NAND gate. This makes it more difficult for an attacker to identify the function of individual gates.

**Gate Duplication:** This technique involves adding redundant gates to a circuit, which can make it more difficult to identify the important gates that are essential for the circuit's function.

**Logic Restructuring:** This technique involves modifying the logic of a circuit to make it more complex and difficult to understand. For example, a designer might add feedback loops, introduce random delays, or introduce noise to the signal to make it more difficult for an attacker to reverse-engineer the circuit.

**Polymorphism:** This technique involves randomly varying the structure of the circuit, so that different versions of the same circuit have different gate-level structures. This makes it more difficult for an attacker to identify patterns in the circuit and reverse-engineer its function.

- Gate-level obfuscation can be most useful technique for enhancing the safety of electronic systems, as it makes it more difficult for attackers to understand the underlying logic of a circuit.
- However, gate-level obfuscation can also introduce additional complexity and potential performance overhead, so it should be used judiciously and with a thorough understanding of its potential tradeoffs.

#### **Diversity:**

- Diversity is a hardware security technique that involves introducing variations in the design of electronic circuits to make them more difficult to attack. The goal of

diversity is to prevent attackers from exploiting vulnerabilities that are common to many different devices or systems, and to make it more difficult to develop attacks that can be easily applied to a large number of targets.

- There are several ways in which diversity can be introduced into hardware designs, including:

**Randomization:** As described above, randomization can be used to introduce variations into the design of electronic circuits. By introducing randomness at the gate or layout level, designers can create diverse circuits that are less susceptible to attack.

**Heterogeneity:** By using different types of components, designers can create circuits that are diverse in terms of their hardware implementation. For example, a designer might use a mix of FPGAs, ASICs, and microcontrollers in a system, rather than relying solely on one type of component.

**Redundancy:** By adding redundant components or subsystems, designers can create systems that are more resilient to attacks. For example, a designer might add redundant processors or memory banks to a system, so that if one component fails or is attacked, the system can still function.

**Obfuscation:** By using techniques such as gate-level obfuscation (described above), designers can create circuits that are difficult to reverse-engineer or analyze. This can help to prevent attackers from identifying common vulnerabilities or developing attacks that can be easily applied to many targets.

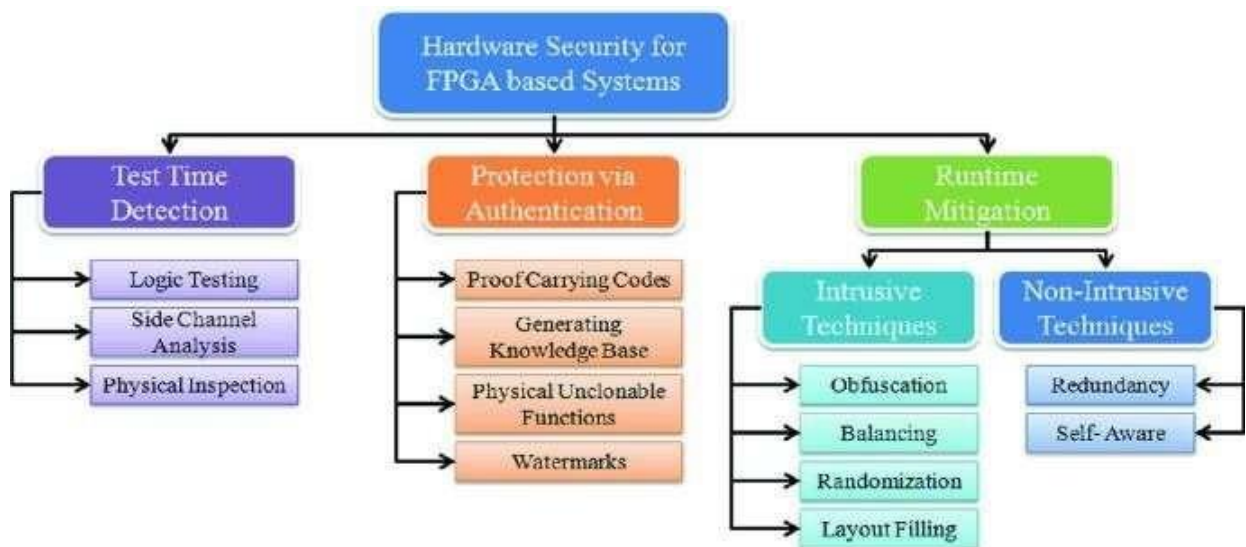


Fig 10: Hardware Security for FPGA based systems

## CHAPTER 4

### Verilog Hardware Description Language

#### **What is Verilog HDL?**

Verilog HDL is used to describe and design circuits and systems at a high level of abstraction. It is commonly used in the design and verification of digital circuits and systems. Verilog HDL is often used in conjunction with a hardware design tool such as Quartus, Vivado, or ModelSim.

Verilog HDL is a strongly typed language, meaning that every variable and expression must have a defined type.

Verilog HDL was developed in the mid-1980s by Phil Moorby at Gateway Design Automation, and later acquired by Cadence Design Systems. The language has gone through several revisions, with the latest being Verilog-2001.

The language provides constructs for describing the structure and behavior of digital components, such as gates, flip-flops, and memory elements. It also supports features for simulation and verification, such as timing constraints and assertion statements.

Verilog HDL is widely used in the semiconductor industry for designing and verifying digital circuits and systems. It is also used in education and research for teaching and studying digital design.

**Bottom-Up Design:** Bottom-up design in Verilog refers to the approach of designing digital circuits by starting from the lowest level of abstraction and building up to the highest level. In other words, the design starts with individual building blocks, such as basic logic gates or flip-flops, and these building blocks are then combined to create more complex components, such as registers or adders. These components are then further combined to create even more complex modules, until the entire system is built.

The bottom-up design approach is often used in Verilog because it allows designers to focus on the details of individual components and ensure that they are functioning correctly before integrating them into larger systems. This approach also promotes modularity and reusability, as individual components can be reused in different designs.

To implement a bottom-up design approach in Verilog, a designer typically starts by defining the basic building blocks of the system, such as logic gates or flip-flops. These building blocks are then tested and verified using simulation tools. Once the building blocks are functioning correctly, they are combined to create more complex components, such as registers or adders. These components are again tested and verified using simulation tools.

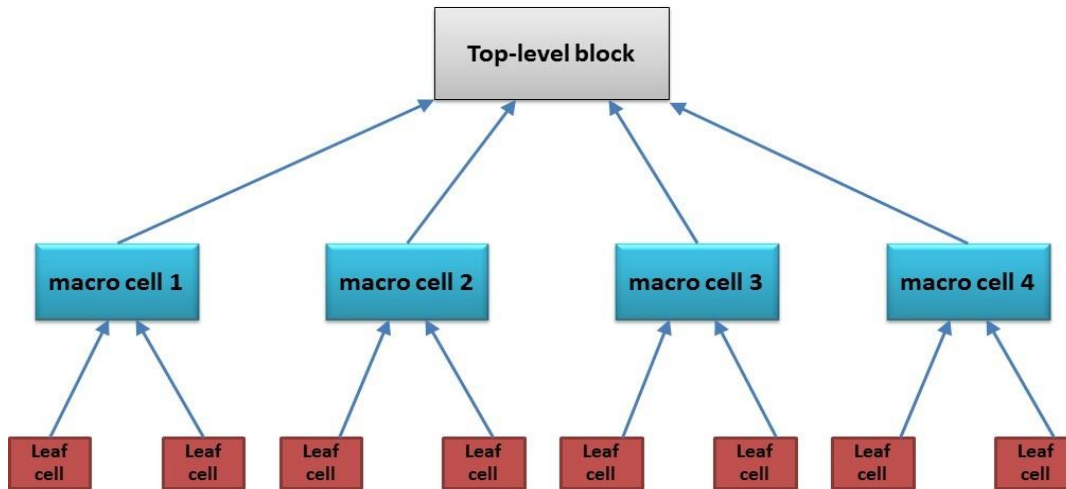


Fig 11: Bottom-Up Design

**Top-Down Design :** Top-down design in Verilog is a design approach that starts with a high-level view of the system and works down to the lowest level of abstraction. In other words, the design starts by defining the overall system architecture and functionality, and then breaks it down into smaller sub-components, each of which is designed and implemented separately.

The top-down design approach in Verilog is often used when designing complex systems, as it allows designers to focus on the overall functionality of the system and ensure that it meets the desired requirements before getting into the details of individual components. This approach also promotes modularity and reusability, as sub-components can be designed and tested independently and then reused in different designs.

To implement a top-down design approach in Verilog, a designer typically starts by defining the system architecture and functionality, and then breaks it down into smaller sub-components. Each sub-component is then designed and implemented separately, starting with the highest level of abstraction and working down to the lowest level.

Once the sub-components are designed and implemented, they are integrated into larger components, and then into the overall system. This integration is typically done using

simulation and/or hardware testing to ensure that each component is functioning correctly before being integrated into larger systems.

The top-down design approach in Verilog can be highly effective for designing complex systems, as it allows designers to focus on the overall functionality of the system and ensure that it meets the desired requirements before getting into the details of individual components. It also promotes modularity and reusability, which can save time and effort in future designs.

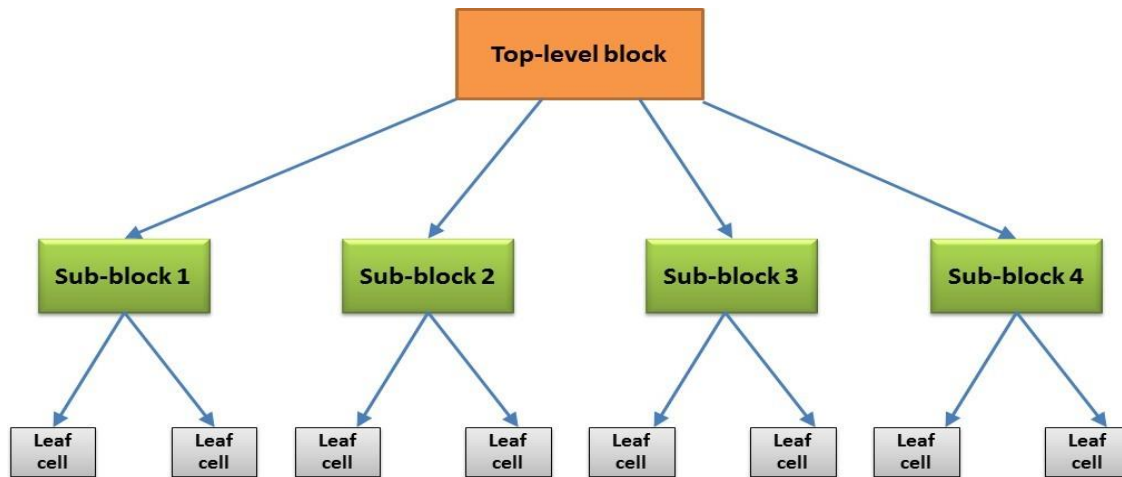


Fig 12: Top-Down Design

### History of Verilog:

Verilog HDL has been around for over three decades, and during this time it has seen many developments and advancements. Some of the notable developments in Verilog are:

**The development of SystemVerilog:** SystemVerilog is an extension of Verilog that adds many new features, such as object-oriented programming, randomization, and assertions. SystemVerilog has become the de facto standard for designing complex digital systems.

**The development of Verification Methodology:** Verification Methodology, or UVM, is a widely used verification framework that provides a standard methodology for creating testbenches in Verilog and SystemVerilog.

**The development of High-Level Synthesis (HLS):** HLS is a tool that allows designers to write Verilog code at a higher level of abstraction, such as C or C++, and then automatically generate Verilog code from it. This approach can save time and effort in designing complex digital systems.

**The development of Open Source Tools:** There are now many open-source tools available for designing and simulating digital systems in Verilog. These tools have made it easier for hobbyists and small companies to design and test their own digital systems.

**The development of FPGA devices:** FPGA (Field Programmable Gate Array) devices have become more powerful and versatile over the years, providing designers with more options for implementing their Verilog designs in hardware.

**The development of Machine Learning (ML) in Verilog:** Machine learning has been incorporated into Verilog to provide designers with a more automated way of generating digital circuits, based on a set of input/output data.

### **Use of Verilog:**

**ASIC Design:** Verilog is widely used for designing Application-Specific Integrated Circuits (ASICs), which are specialized integrated circuits that are designed for a specific application or function. ASICs are used in a variety of applications, including automotive, aerospace, and consumer electronics.

**FPGA Design:** Verilog is also used for designing Field-Programmable Gate Array (FPGA) circuits, which are integrated circuits that can be programmed or reprogrammed after manufacturing. FPGAs are used in a variety of applications, including data centers, telecommunications, and industrial automation.

**SoC Design:** Verilog is also used for designing System-on-Chip (SoC) devices, which are integrated circuits that integrate multiple functions onto a single chip. SoCs are used in a variety of applications, including smartphones, tablets, and embedded systems.

**Verification:** Verilog is widely used for verifying digital circuits and systems. Verification involves testing the digital system to ensure that it meets the required specifications and standards.

**High-Level Synthesis (HLS):** Verilog is also used in High-Level Synthesis (HLS), which is a process of generating hardware descriptions from a high-level software description. HLS can increase productivity and reduce design time for complex systems.



## **LEVELS OF ABSTRACTION:**

VHDL (Verilog Hardware Description Language) provides several levels of abstraction for describing and designing digital circuits and systems. These levels of abstraction allow the designer to understand the functionality and behaviour of a system at all possible levels of detail, based on the difficulty of the design.

The different levels of abstraction in Verilog HDL are:

**Behavioural level:** The first/primary level of abstraction, where the behaviour and functionality of the system are described using high-level constructs. The focus is on what the system does, rather than how it does it. In this phase, the design is typically explained using procedural constructs, such as if-else statements and loops.

**RTL (Register Transfer Level) level:** The next lower level of abstraction, here the functionality of the system is described in terms of registers and combinational logic. The designer specifies the data flow between registers and the logic that operates on that data. At this level, the design is typically described using Verilog HDL constructs, such as always blocks and assign statements.

**Gate level:** This is the final level of abstraction, where the system is described based on individual logic gates, like AND, OR, and NOT gates. At this level, the designer specifies the exact implementation of the logic in the system. The design is typically described using primitive gate-level constructs.

The level of abstraction used in a Verilog HDL design based on the complexity of the system being designed, the design goals, and also the preferences of the designers. Higher levels of abstraction are generally easier to understand and modify, but may not be as efficient as lower levels of abstraction. Lower levels of abstraction are generally more efficient, but can be more difficult to understand and modify.

**Lexical tokens in Verilog HDL:**

In Verilog HDL, a lexical token is a sequence of characters that forms a single unit of meaning for the language. These tokens are used to create the syntax of a Verilog program, and they are categorized into several different types:

**Identifiers:** These are used to name variables, modules, and other program entities. Identifiers must start with a letter or underscore, followed by any combination of letters, numbers, and underscores.

**Keywords:** These are reserved words in Verilog that have a specific meaning and cannot be used as identifiers. Examples include "always", "assign", "begin", "end", "module", and "wire".

**Operators:** Helps to perform mathematical or logical operations on variables. Examples include "+", "-", "\*", "/", "&&", "||", and "~".

**Literals:** These are constant values that are used in a program. Verilog supports several different types of literals, including decimal, binary, octal, and hexadecimal numbers, as well as character and string literals.

**Delimiters:** These are used to separate different elements of a Verilog program. Examples include commas, semicolons, parentheses, and braces.

**Comments:** These are used to add explanatory text to a Verilog program. Comments can be either single-line (starting with "//") or multi-line (starting with "/\*" and ending with "/\*").

In summary, lexical tokens in Verilog HDL are the basic building blocks of the language's syntax, and they include identifiers, keywords, operators, literals, delimiters, and comments.

**GATE LEVEL MODELLING:**

Gate-level modelling is a type of digital circuit modelling that describes logic gates and interconnections among them. In this type of modelling, the circuit is represented using a set of Boolean functions that define the output of each gate based on its inputs. The logic gates used in gate-level modelling are usually simple, such as AND, OR, and NOT gates.

The gate-level modelling process involves the following steps:

- Inputs and outputs of the design should be defined.

- Design the logic gates and their interconnections based on the desired functionality of the circuit.
- Use Boolean algebra to derive the Boolean equations that describe the behaviour of each gate.
- Implement the circuit using physical gate-level components such as transistors, resistors, and capacitors.
- Simulate the circuit to verify its behaviour and correct any errors.
- Gate-level modelling is often used in digital design and verification, where it is important to ensure that the circuit behaves correctly under all possible input conditions. It is also used in the design of electronic systems, such as microprocessors, where the circuit complexity requires efficient and accurate modelling.
- One advantage of gate-level modelling is that it is relatively easy to understand and implement. However, it can be difficult to modify and maintain as the circuit complexity increases. Other modelling techniques, such as register-transfer level (RTL) modelling, may be used in conjunction with gate-level modelling to improve the design and verification process.

### **DATA TYPES:**

In Verilog HDL, there are several different data types used to represent different kinds of information. These data types include:

**Wire:** A wire is used to represent a single bit of information. Wires are used to connect the inputs and outputs of different modules or to connect different components within a single module.

**Reg:** A register is used to represent a single bit of information that can be stored and updated over time. Registers help to store the output of a module or the state of a sequential circuit.

**Integer:** An integer is a data type used to represent a signed or unsigned integer value. Integers can be used to represent numerical values or as loop counters.

**Real:** A real is a data type used to represent a floating-point value. Reals can be used to perform mathematical calculations with decimal values.

**Time:** A time is a data type used to represent a time value. Time can be used to delay the execution of a module or to represent the time elapsed since the simulation began.

**Parameter:** A parameter is a data type used to represent a constant value that is set at compile time. Parameters are used to set the size of data types, configure the behaviour of a module, or specify constants for calculations.

**Array:** An array is a data type used to represent a group of related values. Arrays can be used to store sets of values or to represent multidimensional data.

These data types can be used to define variables and signals in Verilog HDL. The appropriate data type to use depends on the kind of information being represented and the purpose of the variable or signal. Verilog HDL also supports type casting and other operations for manipulating data types.

### **Transmission Gate Primitives:**

- Transmission gate is a type of logic gate that can be used as a primitive in digital circuits. It is also known as a bilateral switch or analog switch. A transmission gate can be used to pass or block signals based on the state of its control input.
- The transmission gate primitive can be represented using a schematic symbol with two inputs, one input for the signal to be transmitted and the other input for the control signal that determines whether the signal is passed or blocked. The output of the transmission gate is the transmitted signal.
- Both 'tranif1' and 'tranif0' primitives can be used to implement bidirectional buses by connecting the output of one transmission gate to the input of another transmission gate. Additionally, both primitives can be used in conjunction with other Verilog logic primitives to implement complex digital circuits.
- In digital circuits, transmission gates are often used for level-shifting and signal isolation. They are also used in analog circuits for signal routing and multiplexing.

### **OPERATORS:**

- Verilog HDL (Hardware Description Language) provides a variety of operators that allow designers to perform operations on digital signals and data types. These operators can be used in expressions to perform arithmetic, logical, relational, and bitwise operations. Here are some of the most common operators in Verilog HDL:

**Arithmetic operators:** Arithmetic operators are used to perform arithmetic operations on signals and data types. The most common arithmetic operators are: Addition (+)

Subtraction (-)

Multiplication (\*)

Division (/)

Modulus (%)

**Logical operators:** Logical operators are used to perform logical operations on signals and data types. The most common logical operators are:

Logical AND (&&)

Logical OR (||)

Logical NOT (!)

**Relational operators:** Relational operators are used to compare two values and return a Boolean value. The most common relational operators are:

Equal to (==)

Not equal to (!=)

Greater than (>)

Less than (<)

Greater than or equal to (>=)

Less than or equal to (<=)

**Bitwise operators:** Bitwise operators are used to perform bitwise operations on signals and data types. The most common bitwise operators are:

Bitwise AND (&)

Bitwise OR (|)

Bitwise XOR (^)

Bitwise NOT (~)

Left shift (<<)

Right shift (>>)

These operators can be combined to create complex expressions that describe the behaviour of a digital circuit or system. Verilog HDL also supports conditional expressions (ternary operator), concatenation operator, reduction operators, etc., which allow the designer to create more sophisticated expressions.

## FUNCTION CALLS:

In Verilog HDL, function calls can be used to perform specific tasks or calculations within a module. Functions are defined using the function keyword and can be called from within other parts of the module using the function name followed by the function arguments in parentheses.

Here an example of a Verilog function that calculates the square of a given input value:

```
function [7:0] square(input [7:0] x);  
  begin  
    square = x * x;end  
endfunction
```

This function takes an 8-bit input value  $x$  and returns its square as an 8-bit output value.

The function can be called from within the module using the following syntax:

```
output [7:0] y;  
input [7:0] x;  
assign y = square(x);
```

In this example, the square function is called with the input value  $x$ , and the output value is assigned to the  $y$  output.

It's important to note that functions in Verilog cannot access or modify any module-level variables or signals. They can only use their input arguments and return a result based on those arguments. Additionally, Verilog functions cannot be used to model hardware directly. Instead, they are typically used to perform mathematical calculations or other non-hardware related tasks within a module.

## MODULES:

- In Verilog HDL, modules help us to understand the behaviour of a digital circuit. A module can be thought of as a black box that takes in inputs and produces outputs based on its internal logic. A module can be instantiated multiple times within a larger design, allowing for complex circuits to be built from smaller building blocks.

- Here is an example of a simple Verilog module that implements a two-input AND gate:

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

- This module takes two input signals, a and b, and produces an output signal c that is the logical AND of the two inputs. The assign statement assigns the value of a & b to the output c.

- To use this module in a larger design, it can be instantiated as follows:

```
wire a, b, c;
and_gate and1(a, b, c);
```

- In this example, the wire keyword is used to declare three signals a, b, and c. The and\_gate module is then instantiated with these signals as inputs and outputs using the syntax and\_gate and1(a, b, c). The resulting circuit will perform the AND operation on the a and b inputs and output the result on the c output.
- Modules in Verilog can be hierarchical, meaning that a module can contain other modules as sub-modules. This allows for complex circuits to be built up from smaller building blocks. Additionally, Verilog modules can include behavioural code such as procedural blocks, conditional statements, and loops, allowing for complex behaviour to be implemented.

### **MODULE DECLARATION:**

- In Verilog, a module declaration is used to define a module, which is a self-contained unit of hardware description that can be instantiated multiple times in a larger design. A module declaration consists of the following components:
- The module keyword: This is used to indicate the start of a module declaration.

- The name of the module: This is a unique identifier that is used to reference the module from other parts of the Verilog code. The module name must start with a letter and can contain letters, numbers, and underscores.
- A list of input and output ports: These ports define the signals that are used to interface with the module. Inputs are specified using the input keyword, while outputs are specified using the output keyword. Each port must have a unique name and a bit width.
- The module body: This is where the behavior of the module is defined using Verilog statements.

- Here is an example module declaration in Verilog:

```
module my_module (
    input clk,
    input  [7:0]  data_in,
    output reg [7:0] data_out
);
endmodule
```

- This declares a module named "my\_module" with three ports: "clk" as an input, "data\_in" as an 8-bit input, and "data\_out" as an 8-bit output. The module body would then include the Verilog code that defines the functionality of the module using combinational or sequential logic.
- Once a module has been defined, it can be instantiated in other modules or in the top-level design. For example, to instantiate the and\_gate module, we can use the following code:

```
wire a, b, c;
and_gate and1(a, b, c);
```

- This code declares three wire signals a, b, and c and then instantiates the and\_gate module using these signals as inputs and outputs. The resulting circuit will perform the AND operation on the a and b inputs and output the result on the c output.
- It's important to note that in Verilog, the order of port declarations in the module header determines the order in which ports are mapped when the module is instantiated. So, in the example above, the first input a will be mapped to the first input of the instantiated and\_gate module, the second input b will be mapped to the second input, and the output c will be mapped to the output.



## MODULE INSTANTIATION:

In Verilog HDL, module instantiation is the process of creating an instance of a module within another module or within the top-level design.

- Here is an example of a module instantiation for a simple two-input AND gate:

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

```
module top_module;  
    wire a, b, c;  
    and_gate and_inst(a, b, c);  
endmodule
```

- In this example, the `and_gate` module is instantiated within the `top_module`. The `wire` keyword is used to declare the signals `a`, `b`, and `c`, and then the `and_gate` module is instantiated with these signals as inputs and outputs using the syntax `and_gate and_inst(a, b, c)`.
- The resulting circuit will perform the AND operation on the `a` and `b` inputs and output the result on the `c` output, which can then be used elsewhere in the `top_module`.
- It's important to note that the flow of the signals in the module instantiation statement must coincide with the port declarations in the module header. In the example above, `a` will be mapped to the first input of the `and_gate` module, `b` will be mapped to the second input, and `c` will be mapped to the output.
- Additionally, multiple instances of a module can be created within a single design, and each instance can have different signal names. Here is an example of instantiating two instances of the `and_gate` module with different input and output signal names:

```
module top_module;  
    wire x, y, z;  
    and_gate and1(x, y, z);
```

```
and_gate and2(a, b, c);
```

```
assign x = a;
```

```
assign y = b;
```

```
assign z = c;
```

```
endmodule
```

- In this example, two instances of the `and_gate` module are instantiated within the `top_module`. The first instance is named `and1` and uses the signals `x`, `y`, and `z` as inputs and outputs. The second instance is named `and2` and uses the signals `a`, `b`, and `c` as inputs and outputs. The signals `x`, `y`, and `z` are assigned to the signals `a`, `b`, and `c`, respectively, using the `assign` statements.

## OPERATORS:

Verilog HDL (Hardware Description Language) provides a variety of operators that allow designers to perform operations on digital signals and `data` types. These operators can be used in expressions to perform arithmetic, logical, relational, and bitwise operations. Here are some of the most common operators in Verilog HDL:

**Arithmetic operators:** Arithmetic operators are used to perform arithmetic operations on signals and `data` types. The most common arithmetic operators are:

Addition (+)

Subtraction (-)

Multiplication (\*)

Division (/)

Modulus (%)

**Logical operators:** Logical operators are used to perform logical operations on signals and `data` types. The most common logical operators are:

Logical AND (&&)

Logical OR (||)

Logical NOT (!)

**Relational operators:** Relational operators are used to compare two values and return a Boolean value. The most common relational operators are:

Equal to (==)

Not equal to (!=)

Greater than (>)

Less than (<)

Greater than or equal to (>=)

Less than or equal to (<=)

**Bitwise operators:** Bitwise operators are used to perform bitwise operations on signals and data types. The most common bitwise operators are:

Bitwise AND (&)

Bitwise OR (|)

Bitwise XOR (^)

Bitwise NOT (~)

Left shift (<<)

Right shift (>>)

- These operators can be combined to create complex expressions that describe the behaviour of a digital circuit or system. Verilog HDL also supports conditional expressions (ternary operator), concatenation operator, reduction operators, etc., which allow the designer to create more sophisticated expressions.

## CHAPTER-5

### TROJAN INSERTION IN IP's AND SIMULATION RESULTS

#### INTRODUCTION TO IP:

- IP stands for Intellectual Property, and in the context of Vivado, it refers to pre-designed and pre-verified building blocks that can be used for faster development of complex digital systems.
- Vivado provides a wide range of IP blocks, including processors, memory controllers, communication interfaces, and specialized logic functions. These IP blocks are designed to be highly configurable, allowing designers to tailor their functionality to the specific needs of their design.
- To use an IP block in Vivado, designers can either create a new IP block from scratch or use an existing IP block from Vivado's IP catalog. The IP catalog contains a library of pre-designed IP blocks that can be added to a design simply by dragging and dropping them into the design canvas.
- Once an IP block is added to a design, it can be configured and customized using the IP Integrator tool in Vivado. The IP Integrator tool provides a graphical interface for connecting IP blocks together and configuring their parameters. It also allows designers to run a validation check on the design to ensure that it is functionally correct.
- IP blocks in Vivado provide a way for designers to leverage pre-designed and pre-verified building blocks to accelerate the development of complex digital systems. Vivado's IP catalog and IP Integrator tool make it easy to add and customize IP blocks in a design, while also providing tools for verifying the correctness of the design.

#### IP ENCRYPTION:

IP encryption refers to the process of encrypting the intellectual property (IP) contained in electronic systems or devices. This is done to protect the IP from being stolen or copied by unauthorized individuals or competitors.

There are different methods and techniques for encrypting IP, and the specific approach used can depend on the type of IP being protected and the level of security required. Here are some examples:

**Symmetric-key encryption:** This involves using a single secret key to both encrypt and decrypt the IP. The same key is shared between the sender and receiver of the encrypted IP.

**Asymmetric-key encryption:** This involves using a pair of keys – a public key for encryption and a private key for decryption. The public key can be freely shared, while the private key must be kept secret.

**Hardware encryption:** This involves using specialized hardware to encrypt the IP. This can provide high levels of security as the encryption is integrated into the hardware itself, making it difficult to tamper with or bypass.

**Obfuscation:** This involves altering the code or design of the IP in a way that makes it difficult to understand or reverse engineer. This can include techniques such as code obfuscation, logic locking, or layout randomization.

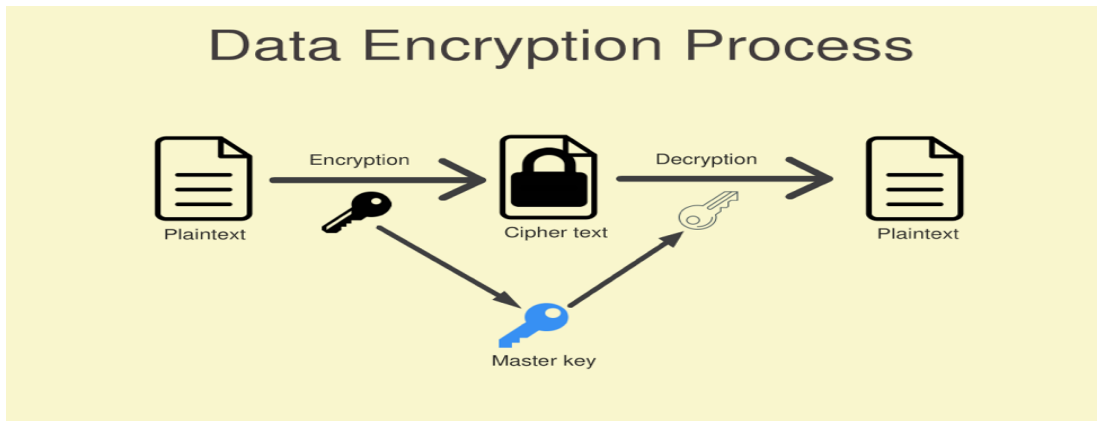


Fig 13: IP Encryption

### WORKING OF IP SECURITY:

IP security (IPsec) is a set of protocols and standards used to secure internet protocol (IP) communications. IPsec provides confidentiality, integrity, and authenticity for IP packets, ensuring that data is protected from unauthorized access, modification, or interception.

Here is a high-level overview of how IPsec works:

**Authentication:** Before IP packets are transmitted, the sender and receiver authenticate each other's identity using a shared secret key or digital certificates. This ensures that packets are only sent and received by authorized parties.

**Encryption:** IP packets are encrypted using a symmetric key algorithm such as Advanced Encryption Standard (AES) or Triple Data Encryption Standard (3DES). This ensures that the data is protected from interception or eavesdropping by unauthorized parties.

**Integrity:** A hash function is applied to the IP packet to generate a message digest or checksum. This ensures that the packet has not been tampered with or modified in transit.

**Decryption:** The receiver decrypts the IP packet using the shared symmetric key, and verifies the message digest to ensure the packet's integrity.

**Anti-replay protection:** IPsec includes a mechanism to prevent replay attacks, where an attacker captures and retransmits a legitimate packet. This is done by including a sequence number in each packet and rejecting packets with duplicate sequence numbers.

**Key management:** IPsec also includes a mechanism for managing the shared secret keys used for encryption and authentication. This can be done using manual key management, where keys are configured manually by network administrators, or using automated key management protocols such as Internet Key Exchange (IKE).

Overall, IPsec provides a secure and robust framework for protecting IP communications. However, it is important to configure and manage IPsec correctly to ensure that it does not impact network performance or cause interoperability issues with other network protocols.

### **Components of IP Security –**

It has the following components:

#### **1. Encapsulating Security Payload (ESP) :**

It provides data integrity, encryption, authentication and anti replay. It also provides authentication for payload.

#### **2. Authentication Header(AH):**

It also provides data integrity, authentication and anti replay and it does not provide encryption. The anti replay protection, protects against unauthorized transmission of packets. It does not protect data's confidentiality.

#### **3. Internet Key Exchange(IKE):**

- It is a network security protocol designed to dynamically exchange encryption keys and find a way over Security Association (SA) between 2 devices. The Security Association (SA) establishes shared security attributes between 2 network entities to support secure communication. The Key Management Protocol (ISAKMP) and

Internet Security Association (ISA) which provides a framework for authentication and key exchange tells how the set up of the Security Associations (SAs) and how direct connections between two hosts that are using IPsec.

- Internet Key Exchange (IKE) provides message content protection and also an open frame for implementing standard algorithms such as SHA and MD5. The algorithm's IPsec users produces a unique identifier for each packet. This identifier then allows a device to determine whether a packet has been correct or not. Packets which are not authorized are discarded and not given to receiver.

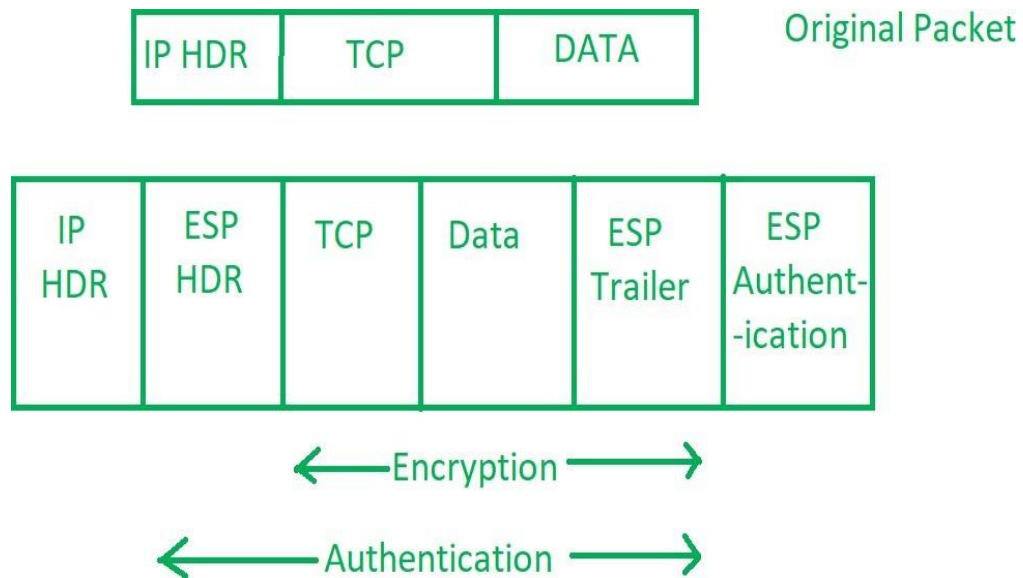


Fig 14: Internet Key Exchange

### Uses of IP Security :

IP security (IPsec) is a widely-used set of protocols and standards that provides secure communication over the internet and private networks. Here are some common uses of IPsec:

**Virtual private networks (VPNs):** IPsec is often used to secure VPN connections, which allow remote users to securely access corporate networks or cloud services from any location. IPsec ensures that all data transmitted between the VPN client and server is encrypted and authenticated.

**Site-to-site communication:** IPsec can be used to secure communication between different locations of an organization, such as branch offices or data centers. This provides a secure and private network for transmitting sensitive data.

**E-commerce:** IPsec can be used to secure online transactions and payments, protecting the confidentiality and integrity of sensitive financial data.

**Voice over IP (VoIP):** IPsec can be used to secure VoIP traffic, ensuring that voice conversations are protected from eavesdropping and tampering.

**Mobile device security:** IPsec can be used to secure communication between mobile devices and corporate networks or cloud services, protecting against unauthorized access or data leakage. Overall, IPsec provides a flexible and versatile framework for securing communication across different types of networks and devices. By using IPsec, organizations can ensure that their data is protected from unauthorized access, interception, or modification.

#### **ADDER OR SUBTRACTOR IP FROM IP CATALOGUE IN VIVADO:**

An adder/subtractor IP from the IP Catalogue in Vivado Xilinx can perform addition and subtraction operations on binary numbers. The IP block is designed to provide efficient and optimized performance for arithmetic operations.

#### **Generating and Customizing IP from IP Core:**

To use the adder/subtractor IP in Vivado Xilinx, follow these steps:

- Open Vivado and create a new project or open an existing one.
- Open the IP Catalogue by clicking on the "IP Catalog" tab in the left-hand pane.
- Browse or search for the "Adder/Subtractor" IP block. You can use the search bar or browse through the categories and subcategories in the IP Catalogue.
- Double-click on the IP block to open the customization window.
- Customize the settings for the IP block, such as the number of bits, operation mode (addition or subtraction), and the input and output ports. The settings can be changed according to your specific requirements.
- Once the customization is complete, click on the "Generate" button to generate the IP block.



- The generated IP block will appear in the "Sources" tab of your project. You can drag and drop the IP block onto your design and connect it to other blocks as required.

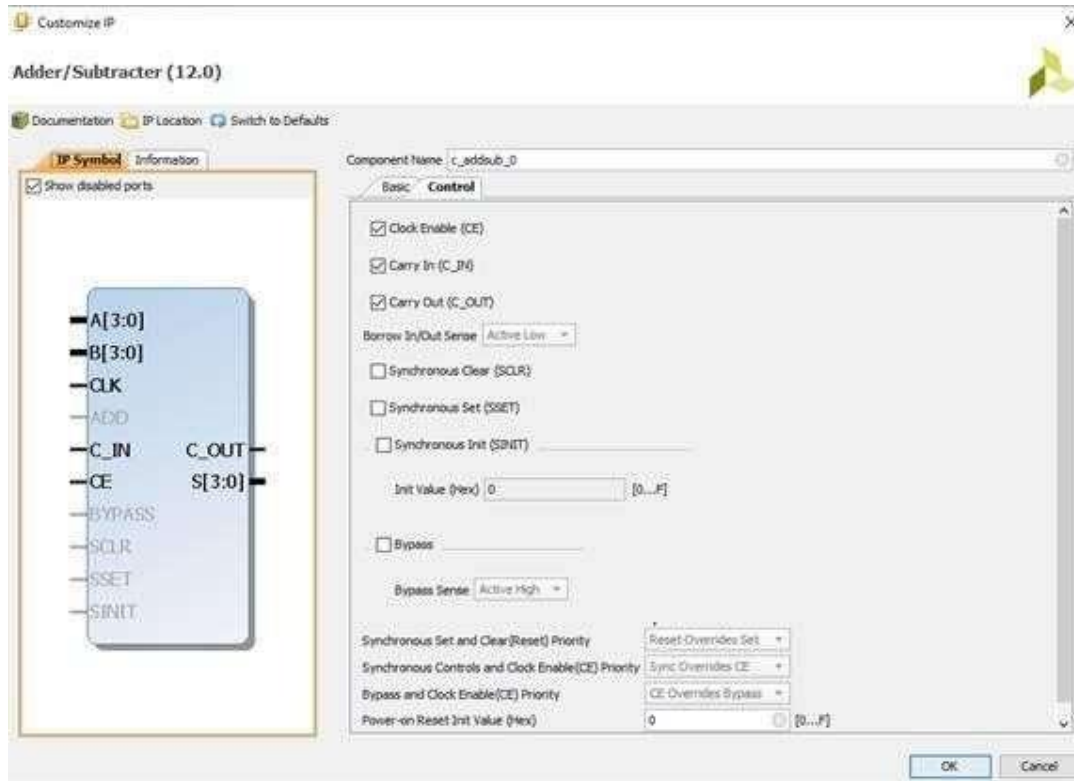


Fig 15: Core design for adder/subtractor IP

### Core Parameters:

The core parameters for the Adder/Subtractor IP in Xilinx Vivado depend on the specific necessity of the design. However, some of the common core parameters that can be customized for the Adder/Subtractor IP are:

- **Data width:** This parameter determines the number of bits that the Adder/Subtractor IP will operate on.
- **Operation mode:** This parameter specifies whether the IP should perform addition, subtraction, or both.
- **Input and output ports:** The input and output ports can be customized to match the requirements of the design.
- **Overflow mode:** This parameter determines how the IP should handle overflow or underflow conditions.

- **Bit growth:** This parameter specifies whether the output should have the same number of bits as the input or whether it should be expanded to accommodate the result of the operation.
- **Clocking options:** This parameter determines how the IP should be clocked, such as using a common clock or separate clocks for the adder and subtractor components.
- **Implementation options:** This parameter allows the designer to choose between different implementation options, such as using LUTs or dedicated DSP blocks.
- **IP customization:** This parameter allows the designer to customize the IP to suit their specific requirements, such as adding or removing features or changing the IP behavior. These are some of the core parameters that can be customized for the Adder/Subtractor IP in Xilinx Vivado. However, the list of parameters may vary depending on the version of Vivado and the specific IP block being
- **Constant Input and Constant Value:** The constant inputs allow you to specify a constant value that is added or subtracted from the input data. This can be useful in cases where you want to implement a circuit that performs a fixed offset operation, such as adding or subtracting a constant value to a sensor reading or a digital signal.
- The constant values option allows you to specify a constant value that is added or subtracted from every bit in the input data. This can be useful in cases where you want to implement a circuit that performs a fixed scaling operation, such as multiplying or dividing the input data by a constant value.
- To use the constant input or constant value options in the Adder/Subtractor IP core in Vivado, you need to configure the IP core using the IP Integrator tool. In the configuration settings, you can select the constant input or constant value option and specify the value that you want to add or subtract
- **Output Width:** In general, the output width for an Adder/Subtractor IP core is equal to the maximum width of the input signals plus one additional bit for the carry-out signal. This is because the addition or subtraction operation can result in a carry-out bit, which needs to be included in the output signal.
- For example, if you have two input signals A and B with a width of 8 bits each, the maximum possible sum or difference between them would be 9 bits (8 bits + 1 carry-out bit). Therefore, the output width for the Adder/Subtractor IP core in this case would be 9 bits.
- However, it is important to note that the output width can be configured to a different value based on the specific requirements of the design. In Vivado, you can use the IP

Integrator tool to configure the Adder/Subtractor IP core and specify the desired output width.

- If the output width is configured to be less than the maximum possible width, the most significant bits of the result will be truncated, which can result in loss of data. Therefore, it is important to ensure that the output width is sufficient to accommodate the expected range of results for the specific application.

#### **Modes to customize adder or subtractor IP:**

- **Add Mode:** An adder/subtractor IP is a digital circuit that performs both addition and subtraction operations. It typically consists of two main components: an adder and a subtractor. To add a mode to an adder/subtractor IP, you could include a control input that selects between addition mode and subtraction mode. In addition mode, the IP would perform addition operations only, while in subtraction mode, it would perform subtraction operations only.
- **Carry In:** When designing an IP core that includes an adder/subtractor circuit, it is important to include documentation that clearly explains the use of the carry-in parameter and any other relevant parameters. This documentation should describe the purpose of the carry-in parameter, how it affects the operation of the adder/subtractor circuit, and any constraints or limitations on its use. Additionally, the documentation should provide examples of how to use the carry-in parameter in typical applications and any other relevant information that the user needs to know in order to effectively use the IP core.
- **Carry Out:** In general, the carry-out parameter specifies the output value of the carry output of the adder/subtractor. This can be a single bit or a multi-bit value, depending on the width of the adder/subtractor. The default value of the carry-out parameter is usually set to zero, which assumes that there is no carry-out from the circuit. However, the user can override this default value and set a specific value for the carry-out parameter.
- **Bypass:** The "Bypass" parameter in the Adder/Subtractor IP determines whether the IP block should be bypassed or not. When the bypass parameter is set to "Yes", the IP block is disabled and the input data is directly passed through to the output. This can be useful in certain scenarios where the design requires a temporary or permanent bypass of the adder/subtractor functionality.

Some of the scenarios where the bypass parameter can be useful are:

**Debugging:** During the design and testing phases, the bypass parameter can be used to temporarily disable the adder/subtractor IP to help isolate issues and debug the design.

**Design flexibility:** The bypass parameter can provide additional flexibility to the design by allowing the designer to easily bypass the adder/subtractor IP when it is not needed.

**Power optimization:** In certain low-power designs, the bypass parameter can be used to turn off the adder/subtractor IP block when it is not required, thus reducing power consumption.

- **Synchronous Controls and Clock Enable (CE):** Synchronous Controls and Clock Enable (CE) Priority are techniques used to control the operation of digital circuits and manage power consumption in FPGA designs.

In synchronous designs, the operations of the digital circuits are synchronized to a clock signal. The synchronous controls refer to the use of synchronous signals to enable or disable various blocks or circuits in the design. The CE signal is one such synchronous control signal that is commonly used to enable or disable the operation of the circuit based on the clock signal.

The CE signal can be used to reduce power consumption by disabling the operation of the circuit when it is not required. However, in some cases, multiple CE signals may be present in the design, and the priority of these signals may need to be determined to ensure proper operation of the circuit.

The CE priority technique involves assigning priority levels to the various CE signals in the design. This ensures that the circuit is only enabled when the highest priority CE signal is active. The priority levels can be assigned based on the criticality of the circuit or the importance of the CE signal.

- **Sync Set and Clear (Reset) Priority:** Sync Set and Clear (Reset) Priority is a technique used to manage the reset signals in digital circuits and ensure proper operation of the design.

In digital circuits, reset signals are used to initialize the state of the circuit or restore it to a known state after an error or malfunction. The Sync Set and Clear (Reset) signals are a type of reset signal that are synchronized to the clock signal.

The Sync Set and Clear (Reset) Priority technique involves assigning priority levels to the various Sync Set and Clear (Reset) signals in the design. This ensures that the circuit is reset

properly and in the correct order when multiple Sync Set and Clear(Reset) signals are present in the design. The priority levels can be assigned based on the criticality of the circuit or the importance of the Sync Set and Clear (Reset) signal.

- **Latency Configuration:**The latency configuration for an adder/subtractor IP in Xilinx Vivado can be set by adjusting the "Pipeline Stages" parameter in the IP configuration GUI. This parameter determines the number of clock cycles that the adder/subtractor takes to complete an operation.

To configure the latency, follow these steps:

- Open the IP configuration GUI for the adder/subtractor IP
- In the "Basic" tab, locate the "Pipeline Stages" parameter.
- Set the "Pipeline Stages" parameter to the desired value.
- Click "OK" to save the configuration.

The latency of the adder/subtractor IP will now be set according to the number of pipeline stages specified. Keep in mind that increasing the number of pipeline stages will increase the latency but may also increase the clock frequency.

- **Borrow IN/OUT Sense:**The "Borrow In/Out Sense" parameter in an adder/subtractor IP determines the polarity of the borrow signal that is used in the subtraction operation. It specifies whether the borrow signal is active high or active low.

In Xilinx Vivado, the "Borrow In/Out Sense" parameter is typically configured in the "Advanced" tab of the IP configuration GUI for the adder/subtractor IP. The possible values for this parameter are:

- **Active High:** When this option is selected, the borrow signal is active high, meaning that a logic high indicates that a borrow must be taken during the subtraction operation.
- **Active Low:** When this option is selected, the borrow signal is active low, meaning that a logic low indicates that a borrow must be taken during the subtraction operation.

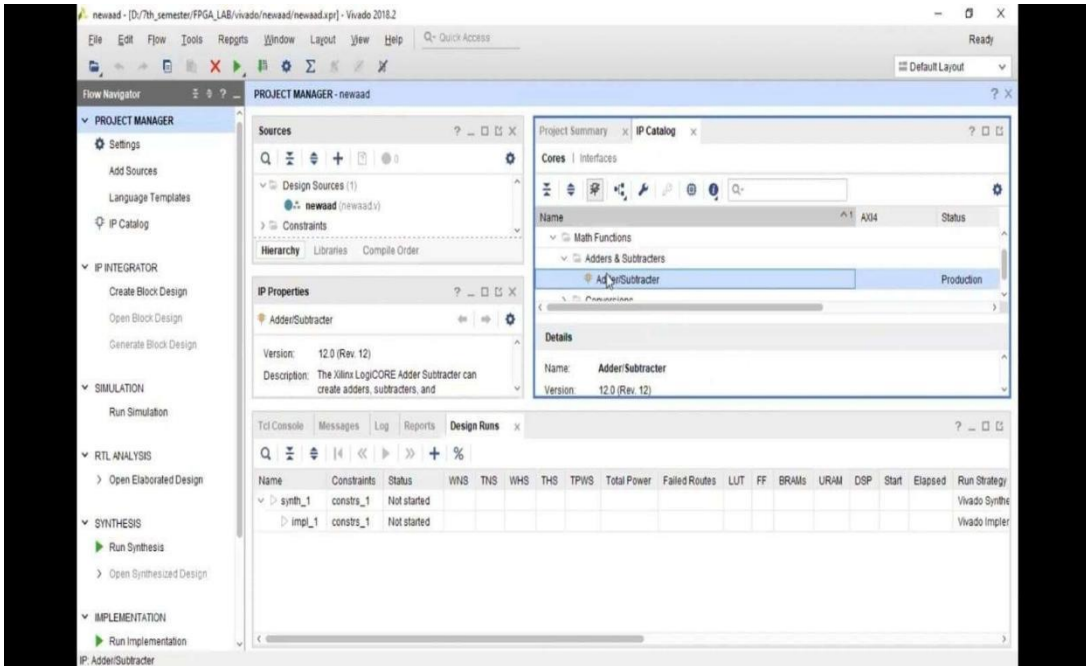


Fig 16: Adder/Subtractor IP from IP catalogue

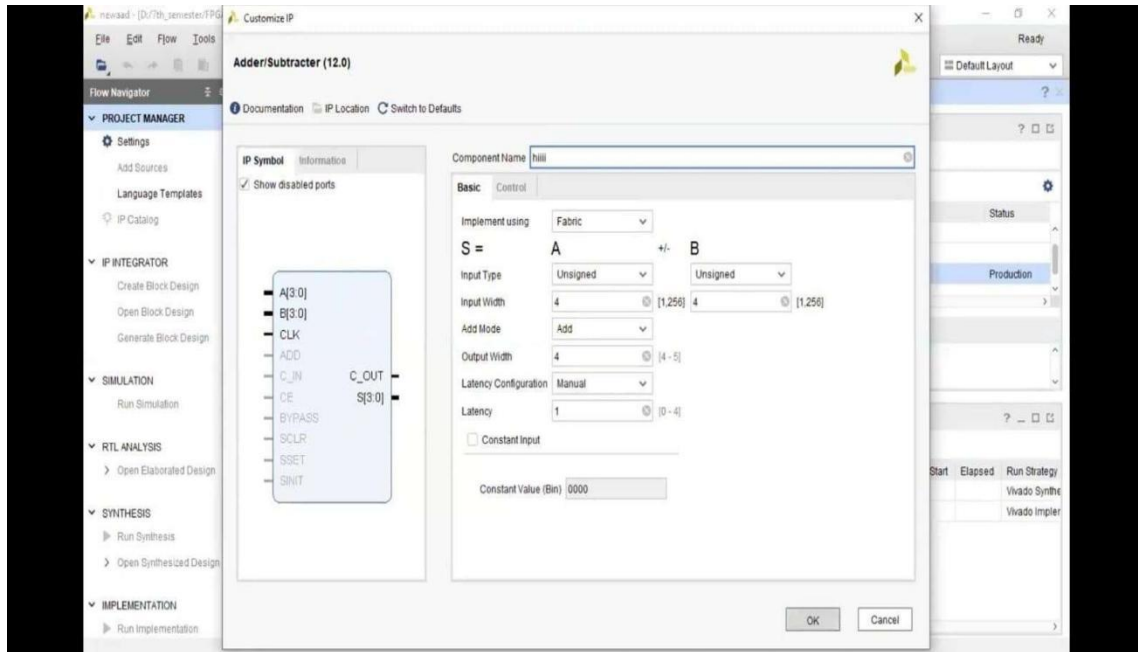


Fig 17: Designing of core IP and Customization

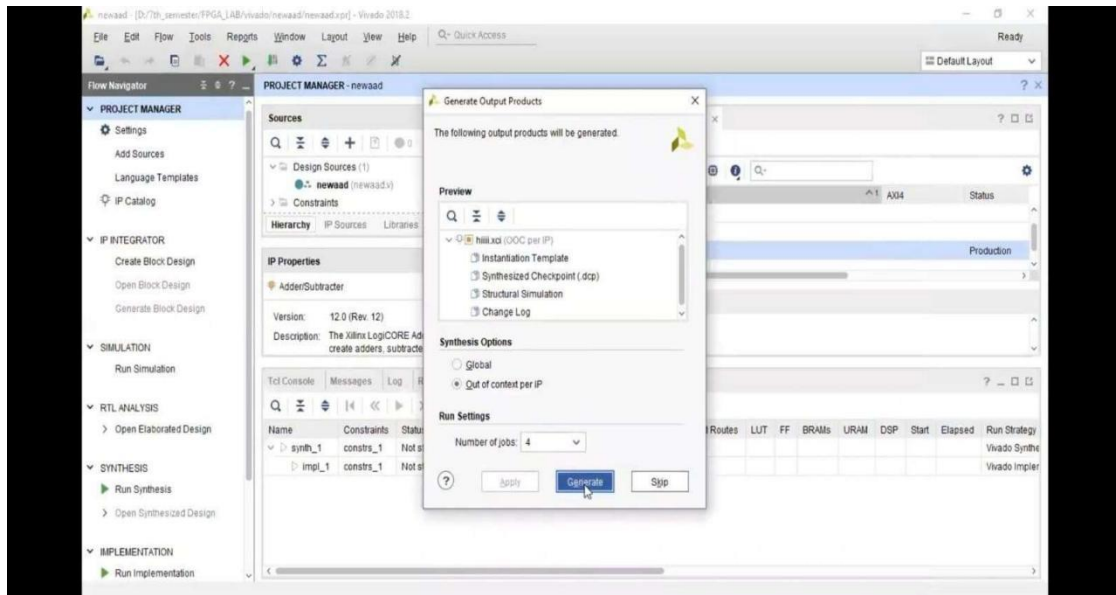


Fig 18: Generation Of Output Products

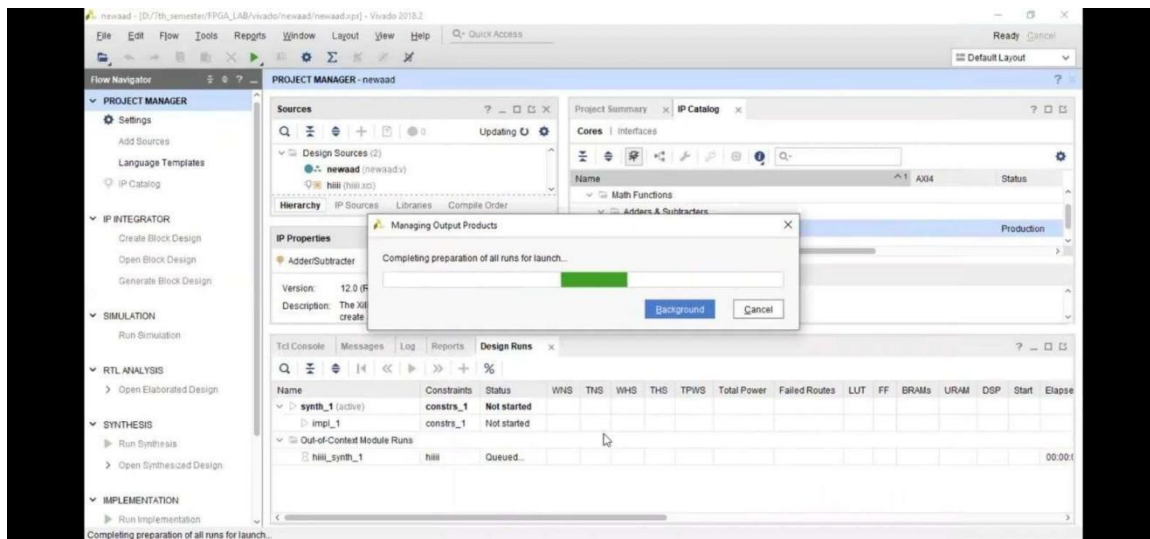


Fig 19: Preparing to launch the adder/subtractor IP

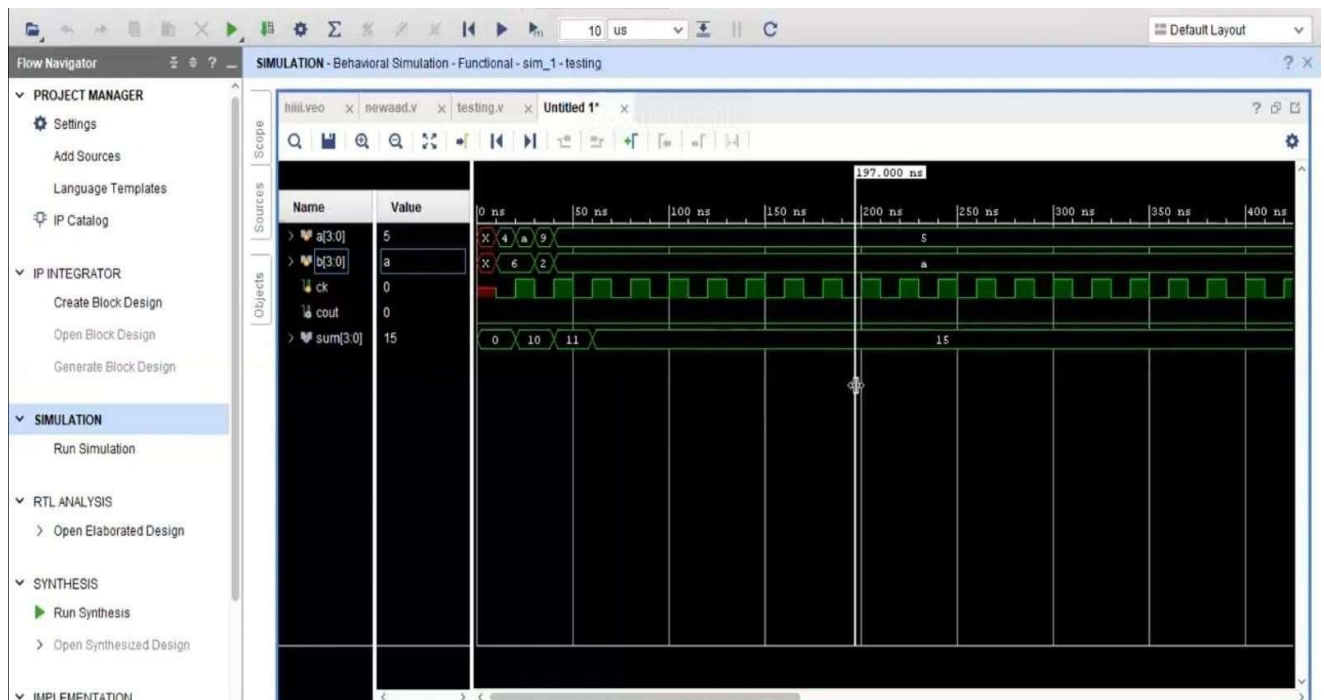


Fig 20: Simulation results before inserting the trojan for adder/subtractor IP

### TROJAN INSERTION AT ALGORITHM LEVEL:

- The insertion of a Trojan at the algorithm level refers to the intentional modification of an algorithm's code to introduce a security vulnerability that can be exploited by an attacker.
- This type of attack can be particularly dangerous because it can go unnoticed for a long time and can affect a large no.of users or systems.
- There are several ways a Trojan can be inserted at the algorithm level. One way is to modify the source code of the algorithm directly. Another way is to introduce a Trojan during the compilation or build process.
- This can be done by infecting the compiler or build tools with malware, or by replacing legitimate files with Trojan-infected ones.
- Once the Trojan is inserted, it can be used to carry out a variety of malicious actions, such as stealing sensitive data, altering the behaviour of the algorithm, or disrupting the operation of the system.
- Detecting a Trojan at the algorithm level can be difficult, as it may not exhibit any obvious symptoms and can be designed to evade detection by antivirus software and other security measures.
- To protect against this type of attack, it is important to use secure development practices, such as code reviews, testing, and version control.



- Additionally, it is important to keep software and tools up to date with the latest security patches and to monitor for any suspicious activity or changes in behaviour.

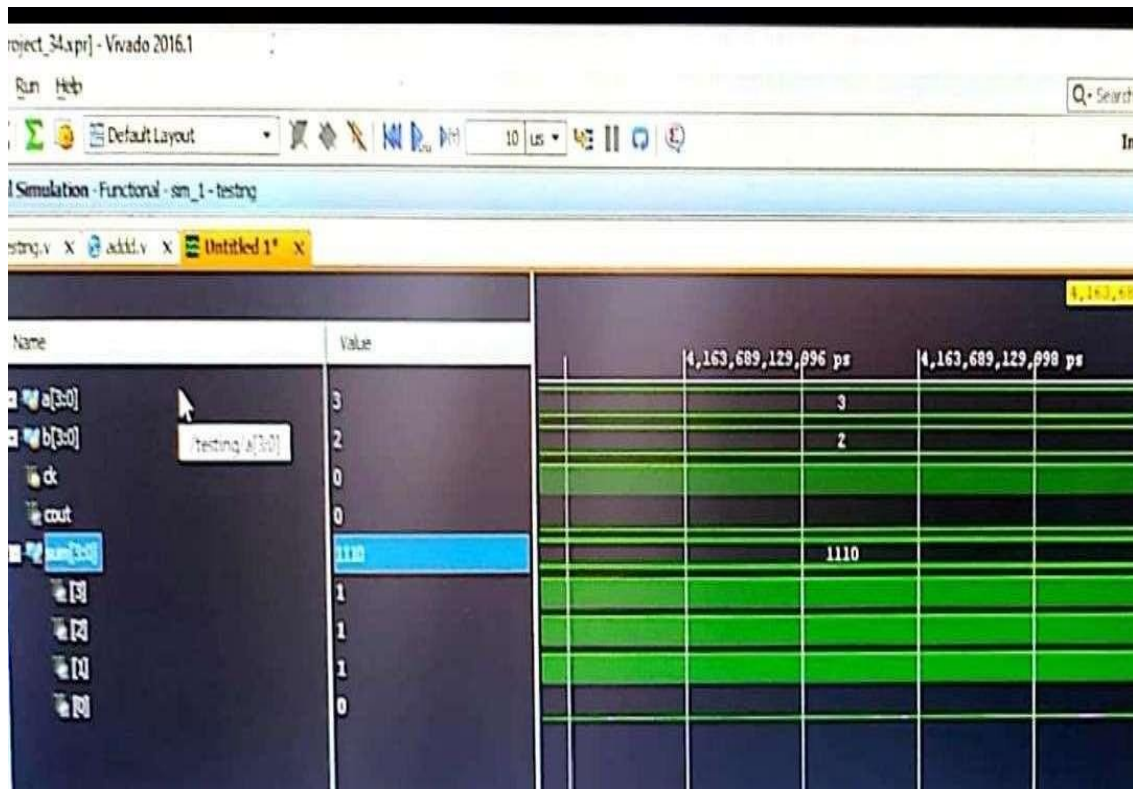


Fig 21: Simulation results after insertion of trojan for adder/subtractor IP

## CHAPTER-6

### CONCLUSION AND FUTURE SCOPE

#### **Conclusion:**

Overall, it's important to remember that trojans are a persistent security threat, and as technology evolves, so will the methods used to implement and detect them. Therefore, it is essential to stay vigilant and proactive in developing new defenses against these threats in future.

Trojan horses have been a persistent security threat for many years, and unfortunately, they are likely to remain a threat in the future. As technology advances, there will be new opportunities for trojan implementation, and as such, there will always be a need for effective detection methods. Here are some potential future directions for both trojan implementations and their detection

#### **Future Scope for Trojan Implementations:**

**Artificial Intelligence:** As artificial intelligence (AI) technology continues to develop, trojans could be programmed to leverage this technology in new ways. For example, a trojan could be programmed to learn about its target and adapt its behavior accordingly to evade detection.

**Internet of Things (IoT):** The increasing use of IoT devices, trojans could be designed to exploit the vulnerabilities in these devices, potentially gaining access to a network or compromising a system.

**Blockchain:** Blockchain technology has the potential to be used as a means of distributing trojans in a decentralized manner, making them more difficult to detect and trace.

#### **Future Scope for Trojan Detection:**

**Machine Learning:** Machine learning algorithms could be used to analyze large datasets of known malware samples and learn to recognize patterns that are indicative of trojan behavior. This could lead to more accurate and effective detection methods.

**Behavior-Based Detection:** Rather than relying solely on signature-based detection, behavior-based detection methods could be used to analyze the behavior of a system and detect unusual activity that may be indicative of a trojan.

**Hardware-Level Detection:** Hardware-level detection methods, such as those based on hardware-based security features or on-chip security, could provide a more robust means of detecting trojans that are designed to evade software-based detection methods.

**Collaborative Defense:** Sharing information about new malware samples and attack techniques among security researchers, industry experts, and government agencies could help to create a collaborative defense that can more effectively combat trojan threats.

## References:

- [1] S. Pagliarini, J. Sweeney, K. Mai, S. Blanton, S. Mitra, and L. Pileggi, "Split-chip design to prevent ip reverse engineering," *IEEE Design &Test*, 2020.
- [2] S. Yu, C. Gu, W. Liu, and M. O'Neill, "A novel feature extraction strategy for hardware trojan detection," in *2020 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–5, 2020.
- [3] K. Hasegawa, M. Yanagisawa, and N. Togawa, "Trojan-feature extraction at gate-level netlists and its application to hardware-trojan detection using random forest classifier," in *2017 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 1–4, 2017.
- [4] K. Yang, M. Hicks, Q. Dong, T. Austin, and D. Sylvester, "A2: Analog malicious hardware," in *2016 IEEE Symposium on Security and Privacy (SP)*, pp. 18–37, 2016.
- [5] S. Bhasin and F. Regazzoni, "A survey on hardware trojan detection techniques," in *2015 IEEE International Symposium on Circuits and Systems (ISCAS)*, pp. 2021–2024, 2015.
- [6] U. Guin et al., "Counterfeit integrated circuits: A rising threat in the global semiconductor supply chain," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1207–1228, 2014.
- [7] R. Kumar, P. Jovanovic, W. Burleson, and I. Polian, "Parametric trojans for fault- injection attacks on cryptographic hardware," in *2014 Workshop on Fault Diagnosis and Tolerance in Cryptography*, pp. 18–28, 2014.
- [8] M. Rostami, F. Koushanfar, and R. Karri, "A primer on hardware security: Models, methods, and metrics," *Proceedings of the IEEE*, vol. 102, no. 8, pp. 1283–1295, 2014.
- [9] Y. Liu, Y. Jin, and Y. Makris, "Hardware trojans in wireless cryptographic ics: Silicon demonstration & detection method evaluation," in *Int. Conf. on Computer-Aided Design (ICCAD)*, pp. 399–404, 2013.
- [10] J. Rajendran, V. Jyothi, O. Sinanoglu, and R. Karri. Design and analysis of ring oscillator based design-for-trust technique. In *VLSI Test Symposium (VTS), 2011 IEEE 29th*, pages 105 –110, may 2011.
- [11] X. Zhang and M. Tehranipoor. Case study: Detecting hardware Trojans in third-party digital ip cores. In *2011 IEEE International Symposium on Hardware-Oriented Security and Trust (HOST)*, pages 67 –70, June 2011.
- [12] S. Wei and M. Potkonjak. Scalable hardware trojan diagnosis. *Very Large Scale Integration (VLSI) Systems*, *IEEE Transactions on*, PP(99):1–9, 2011.
- [13] R. Torrance and D. James, "The state-of-the-art in semiconductor reverse engineering," *Design Automation Conference*, pp. 333–338, 2011.
- [14] J.-F. Gallais et al., "Hardware trojans for inducing or amplifying sidechannel leakage of cryptographic software," in *Trusted Systems*, pp. 253– 270, 2011.
- [15] A. Waksman and S. Sethumadhavan. Silencing hardware backdoors. In *Proceedings of the 2011 IEEE Symposium on Security and Privacy, SP '11*, pages 49–63. *IEEE Computer Society*, 2011.
- [16] D. Du, S. Narasimhan, R. Chakraborty, and S. Bhunia. Self-referencing: a scalable side- channel approach for hardware trojan detection. In *Proceedings of the 12th international conference on Cryptographic hardware and embedded systems, CHES'10*, pages 173–187, 2010.

- [17] M. Tehranipour and F. Koushanfar, "A survey of hardware trojan taxonomy and detection," *IEEE Design Test of Computers*, vol. 27, no. 1, pp. 10–25, 2010.
- [18] A. Baumgarten, A. Tyagi, and J. Zambreno. Preventing IC piracy using reconfigurable logic barriers. *IEEE Des. Test*, 27(1):66–75, Jan. 2010.
- [19] Y. Jin and Y. Makris, "Hardware trojans in wireless cryptographic ics," *IEEE Design Test of Computers*, vol. 27, no. 1, pp. 26–35, 2010.
- [20] M. Potkonjak, A. Nahapetian, M. Nelson, and T. Massey. Hardware trojan horse detection using gate-level characterization. In *Proceedings of the 46th Annual Design Automation Conference, DAC '09*, pages 688–693, 2009.
- [21] Y. Alkabani and F. Koushanfar. Consistency-based characterization for IC trojan detection. In *Proceedings of the 2009 International Conference on Computer-Aided Design, ICCAD '09*, pages 123–127, 2009.
- [22] M. Banga and M. S. Hsiao. A novel sustained vector technique for the detection of hardware trojans. In *Proceedings of the 22nd International Conference on VLSI Design*, pages 327–332, 2009.
- [23] L. Lin et al., "Trojan side-channels: Lightweight hardware trojans through side-channel engineering," in *Cryptographic Hardware and Embedded Systems - CHES 2009*, pp. 382–395, 2009.
- [24] L. Lin, W. Bursleson, and C. Paar, "Moles: Malicious off-chip leakage enabled by side-channels," in *2009 IEEE/ACM International Conference on Computer-Aided Design*, pp. 117–122, 2009.
- [25] X. Wang, H. Salmani, M. Tehranipour, and J. Plusquellic. Hardware trojan detection and isolation using current integration and localized current analysis. In *Proceedings of the IEEE International Symposium on Defect and Fault Tolerance of VLSI Systems*, pages 87–95, 2008.
- [26] P. Kocher, J. Jaffe, and B. Jun, "Differential power analysis," in *Advances in Cryptology — CRYPTO' 99* (M. Wiener, ed.), pp. 388–397, 1999.
- [27] Cadence Design Systems, "Virtuoso Layout Suite," [Online]. Available at: [https://www.cadence.com/en\\_US/home/tools/custom-ic-analog-rf-design/layout-design/virtuoso-layout-suite.html](https://www.cadence.com/en_US/home/tools/custom-ic-analog-rf-design/layout-design/virtuoso-layout-suite.html).
- [28] M. Lapedus, "Big trouble at 3nm," [Online]. Available at: <https://semiengineering.com/big-trouble-at-3nm/>.
- [29] Synopsys, "Synopsys Custom Design Platform," [Online]. Available at: <https://www.synopsys.com/implementation-and-signoff/custom-design-platform.html>.
- [30] Mentor, "Calibre ®," [Online]. Available at: [https://www.mentor.com/products/ic\\_nanometer\\_design/](https://www.mentor.com/products/ic_nanometer_design/).