

*Heaven's Light is Our Guide*

**Rajshahi University of Engineering and Technology**

Department of Electrical & Electronic Engineering



Report on VLSI Training Session at

**THiNK Silicon**

**Supervised By:**

Dr. Mohammad Abdul Motin  
Professor  
Dept. of Electrical & Electronic Engineering Rajshahi  
University of Engineering and Technology

**Prepared by:**

Muhammad Sudipto Siam Dip  
Roll: 1801023  
Course No.: EEE 4100  
Course Title: Industrial Training

## Declaration

This is to certify that the attachment work entitled “**Report on VLSI Training Session at THiNK Silicon**” has been carried out by **Muhammad Sudipto Siam Dip (1801023)** under my supervision in Department of Electrical & Electronic Engineering, Rajshahi University of Engineering & Technology (RUET), Rajshahi.

Attachment Supervisor

.....

Dr. Mohammod Abdul Motin

Professor

Dept. of Electrical & Electronic Engineering

Rajshahi University of Engineering and Technology

## Acknowledgement

Industrial training allows students to apply theoretical knowledge gained in classrooms to real world scenarios. It bridges the gap between theory and practice, helping students understand how concepts and principles are implemented in practice settings. We had the great privilege of participating in the Industrial Training program offered by the Department of Electrical and Electronic Engineering, Rajshahi University of Engineering & Technology (RUET) at **THiNK Limited**, which allowed us to gain a thorough understanding of VLSI industry as well as the electronic section in practical life. The industrial attachment visit to **THiNK Limited** from March 26, 2023 to April 05, 2023 has been an important milestone in my educational journey. This attachment has provided me with practical exposure of different design like basic gate and chip design in VLSI section and has increased my understanding of various crucial aspects related to VLSI industry.

I would like to express my sincere appreciation to the management, especially to the Ashiqur Tanim, Founder & CEO, THiNK Limited and Sirajul Alam Khan, Director & CTO, THiNK Limited. Also I am thankful to Akash Ghosh, SoC Design Engineer, THiNK Limited for conducting this session and other staff of THiNK Limited for providing me with the opportunity to be a part of the training and gain practical insights into the operations of a real-life VLSI industry.

Finally, I am deeply indebted and grateful to **Dr. Mohammad Abdul Motin**, Professor, Department of Electrical and Electronic Engineering, Rajshahi University of Engineering & Technology, for his invaluable guidance, mentorship, and supervision throughout this entire process.

Muhammad Sudipto Siam Dip

Roll: 1801023

Dept. of EEE, RUET

## **Abstract**

This report provides a comprehensive overview of the industrial attachment conducted at the THiNK Limited, spanning Eleven days. The attachment aimed to provide firsthand exposure to overall idea about chip design and VLSI. Throughout the attachment period, the author gained valuable insights into the application of VLSI and different design process of basic gates and chip through Verilog software and observed the practical implementation of theoretical concepts learned in the classroom.

The report is structured into several chapters, each delving into specific topics related to VLSI. These chapters cover essential areas such as different tools for designing different chip, specially Verilog software and provide a comprehensive understanding of the underlying principles and functioning of these components.

Overall, this report serves as a documentation of the industrial attachment experience at THiNK Limited, providing valuable knowledge about VLSI and offering a platform for further research and exploration in the field of electronic engineering.

# Table of Content

<b>DECLARATION</b> .....	<b>2</b>
<b>ACKNOWLEDGEMENT</b> .....	<b>3</b>
<b>ABSTRACT</b> .....	<b>4</b>
<b>CERTIFICATE</b> .....	<b>5</b>
<b>TABLE OF CONTENT</b> .....	<b>6</b>
<b>LIST OF FIGURES</b> .....	<b>8</b>
<b>LIST OF TABLES</b> .....	<b>9</b>
<b>CHAPTER 1</b> .....	<b>10</b>
<b>INTRODUCTION</b> .....	<b>10</b>
INTRODUCTION ABOUT TRAINING ESTABLISHMENT .....	10
1.1. KEY PEOPLE.....	11
1.2. TRAINING APPROACH.....	11
1.3. GRADUATION DAY.....	12
<b>CHAPTER 2</b> .....	<b>13</b>
<b>REGISTER TRANSFER LEVEL &amp; INTEGRATED CIRCUIT</b> .....	<b>13</b>
2.1. CLASSIFICATION .....	13
2.2. INTEGRATED CIRCUIT .....	14
2.3. IC DESIGN.....	15
2.4. TYPES OF ICs.....	16
<b>CHAPTER 3</b> .....	<b>21</b>
<b>VERILOG</b> .....	<b>21</b>
3.1. VHDL .....	21
3.2. VERILOG .....	22
3.3. ABSTRACTIONS LEVEL OF VERILOG .....	24
3.4. MODULES.....	26
3.5. SIMULATION, SYNTHESIS, AND DESIGN METHODOLOGY IN VERILOG.....	28
3.6. DATA TYPE .....	29
3.7. OPERATORS IN VERILOG.....	31
<b>CHAPTER 4</b> .....	<b>32</b>
<b>VERILOG CODES WITH OUTPUTS</b> .....	<b>32</b>
4.1. BASIC LOGIC GATES.....	33
4.2. MULTIPLEXERS & DEMULTIPLEXERS.....	34
4.3. ENCODER & DECODER .....	36
4.4. EVEN PARITY CHECKER.....	37
4.5. ADDER AND SUBTRACTOR.....	38
<b>CONCLUSION</b> .....	<b>39</b>
6.1. LEARNINGS.....	40
6.2. FIELDS TO IMPROVE .....	40
6.3. REFERENCES .....	41

## List of Figures

FIGURE 1.1 FLOW CHART OF THE TRAINING SESSION .....	11
FIGURE 1.2 PICTURE WITH ASHIQUR TANIM SIR AND MD SIRAJUL ALAM KHAN SIR .....	12
FIGURE 1.3 AT THE TIME WHEN SIRAJUL ALAM SIR DELIVERED SPEECH.....	12
FIGURE 2. 1 INTEGRATED CIRCUIT .....	14
FIGURE 2. 2 DIGITAL IC DESIGN PROCESS.....	15
FIGURE 2. 3 MIXED-SIGNAL IC DESIGN PROCESS .....	16
FIGURE 2. 4 TYPES OF ICs.....	17
FIGURE 2. 5 APPLICATION-SPECIFIC INTEGRATED CIRCUIT (ASIC).....	17
FIGURE 2. 6 APPLICATION SPECIFIC STANDARD PART.....	18
FIGURE 2. 7 FIELD PROGRAMMABLE GATE ARRAY .....	18
FIGURE 2. 8 SYSTEM ON CHIP .....	19
FIGURE 2. 9 MICROPROCESSOR AND ITS DIFFERENT ACCESSORIES .....	19
FIGURE 3. 1 VERILOG FRAMEWORK.....	26
FIGURE 3. 2 MUX 4:1 USING 2:1 MUX.....	27
FIGURE 3. 3 SIMULATION MODEL .....	28
FIGURE 3. 4 SYNTHESIS MODEL.....	28
FIGURE 3. 5 DESIGN METHODOLOGY (A) TOP-DOWN (B) BOTTOM-UP.....	29
FIGURE 4. 1 CODE & RTL OUTPUT OF 2:1 MUX.....	34
FIGURE 4. 2 CODE & RTL OUTPUT OF 4:1 MUX.....	34
FIGURE 4. 3 CODE & RTL OUTPUT OF DEMUX 1:8.....	35
FIGURE 4. 4 CODE & RTL OUTPUT OF ENCODER (8X1) .....	36
FIGURE 4. 5 DECODER 3X8 CODE .....	36
FIGURE 4. 6 RTL OUTPUT OF 3X8 DECODER.....	37
FIGURE 4. 7 CODE & RTL OUTPUT OF PARITY CHECKER .....	37
FIGURE 4. 8 CODE & RTL OUTPUT OF HALF ADDER .....	38
FIGURE 4. 9 CODE & RTL OUTPUT OF FULL ADDER .....	38
FIGURE 4. 10 CODE & RTL OUTPUT OF HALF SUBTRACTOR .....	39
FIGURE 4. 11 CODE & RTL OUTPUT OF FULL SUBTRACTOR.....	39

**List of Tables**

TABLE 3. 1 THE DIFFERENCES BETWEEN VHDL AND VERILOG ..... 24

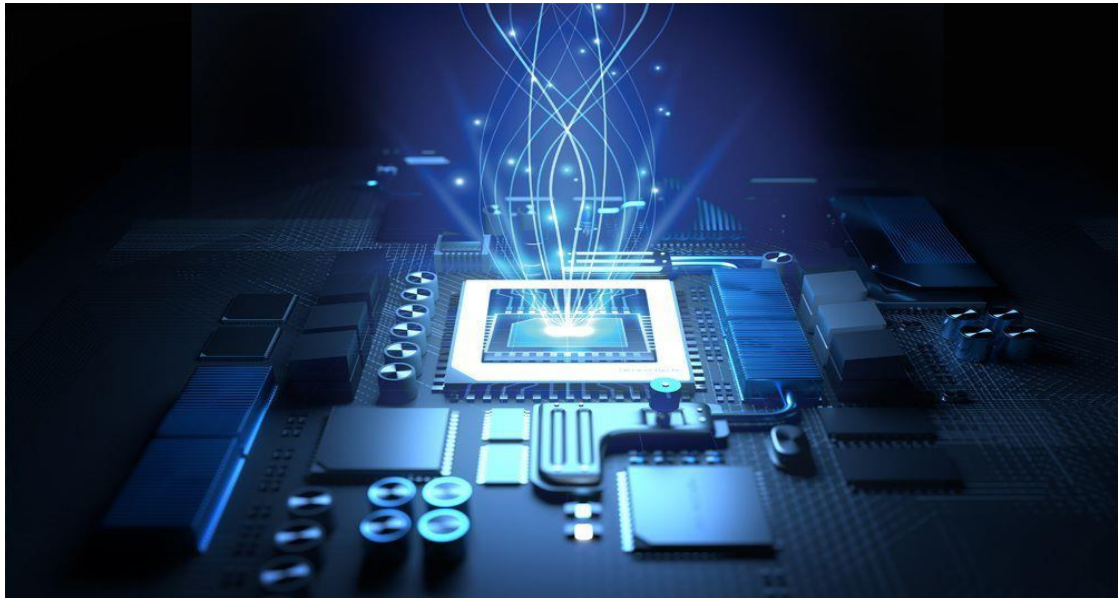
TABLE 3. 2 EXAMPLE OF GATE LEVEL ..... 25

TABLE 4. 1 BASIC LOGIC GATES ..... 33

# Chapter 1

## Introduction



### Introduction about Training Establishment

THiNK is a design, engineering and manufacturing enterprise that makes smart automation devices and IIoT (Industrial Internet of Things) equipments for a variety of industries. With in-house R&D and manufacturing capabilities, THiNK prides itself as a one-stop solutions provider, where it can handle projects from concept to completion.

THiNK headquarters is located in a prime location in Dhaka's business centre. The office hosts our in-house R&D team, manufacturing facilities, testing facilities, and business teams. This enables us to have a cohesive and connected system of operations negating any form of information lag and communication delays.

The company has three service units:

1. Embedded system design
2. Semiconductor design
  - i. VLSI Frontend design
  - ii. VLSI Backend design
  - iii. FPGA prototyping
3. IoT core



It has six service units:

1. IoT core
2. BenchWorks
3. THINK LABs
4. THiNK Education
5. THiNK Silicon
6. THiNK Digital

## Training Approach

The internship session lasted for 10 working days, which included 7 working days during which we covered a variety of topics. Initially, we were introduced to THiNK Group Ltd and its operational procedures. Subsequently, the actual internship commenced, and we were introduced to RTL engineering and the basics of Verilog, which involved learning about different coding methods such as switch level, gate level, dataflow level, and behavioral level. We also gained knowledge about test benches.

Our training was conducted by the THiNK silicon-physical design and engineering team, who guided us through the VLSI frontend part of designing a physical circuit using CAD tools, specifically Quartus software. The training was expertly led by Mr. Akash Gosh and Shazarul Islam, both members of THiNK Silicon.

The topics covered during the training are presented below in the form of a flow chart:

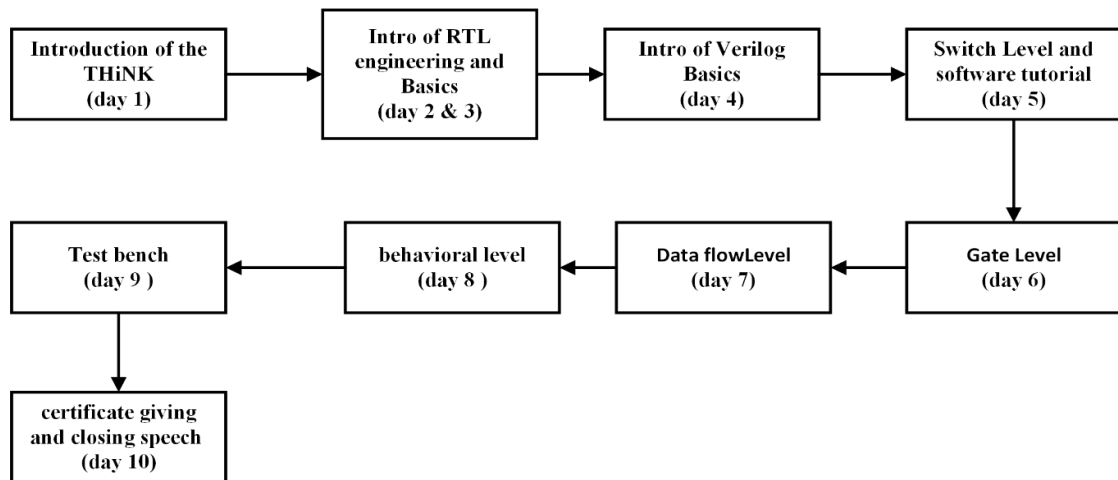


Figure 1. 1 Flow Chart of the training session

## **1.1. Graduation Day**

On this day, Ashiqur Tanim, the Founder & CEO, and Md Sirajul Alam Khan, the CTO, delivered a brief speech discussing the future of the VLSI industry in Bangladesh and emphasized the opportunities for career development in this field. They also shared some exciting plans for the expansion of THiNKin the future. Following their insightful speeches, we were presented with certificates, and we had the privilege of taking memorable photographs alongside them.

## Chapter 2

### Register Transfer Level & Integrated Circuit

RTL stands for Register Transfer Level, and it is a digital design abstraction used in digital circuit design and VLSI (Very Large-Scale Integration) design. It represents a level of abstraction in the design process where the behavior of a digital circuit is described in terms of the flow of data between registers. RTL is an intermediate level between the high-level behavioral description and the low-level gate-level description of a digital circuit.

In RTL design, the circuit's behavior is defined using hardware description languages (HDLs) like Verilog or VHDL. The focus is on specifying the data flow between registers, the logic operations performed on the data, and the control signals that govern the flow of data between the registers.

RTL descriptions do not specify the exact implementation details, such as specific gates or transistors. Instead, it focuses on the functional behavior of the circuit, making it a more abstract and portable representation. The RTL description is used as the basis for further synthesis and optimization steps to generate the gate-level implementation or to program an FPGA (Field-Programmable Gate Array).

RTL design allows designers to work at a higher level of abstraction, making it easier to express the intended functionality of the circuit and facilitating design verification. It is widely used in digital design for various applications, including ASIC (Application-Specific Integrated Circuit) design and FPGA design.

#### 2.1. Classification

RTL (Register Transfer Level) can be classified based on its usage and purpose in digital design. Here are the common classifications of RTL:

**2.1.1. Behavioral RTL:** This type of RTL describes the functional behavior of a digital circuit without specifying the implementation details. It focuses on data flow, operations, and control signals between registers. Behavioral RTL is primarily used for early-stage design exploration, functional verification, and algorithmic-level simulations.

**2.1.2. Structural RTL:** Structural RTL describes the circuit in terms of interconnected hardware components such as registers, multiplexers, and logic gates. It provides a more detailed representation of the digital circuit compared to behavioral RTL. Structural RTL is used for logic synthesis and implementing the design in hardware, such as FPGA or ASIC.

**2.1.3. RTL Synthesizable:** RTL code that is synthesizable is written in a way that can be automatically converted into gate-level representations through logic synthesis tools. It follows specific coding guidelines and rules, ensuring that the design can be efficiently synthesized without ambiguity.

**2.1.4. RTL Non-Synthesizable:** This type of RTL code is written for simulation and verification purposes and may include constructs that cannot be directly translated into hardware during logic synthesis. Non-synthesizable RTL is used to model complex behaviors, testbenches, and assertions for functional verification.

**2.1.5. Behavioral RTL vs. Register Transfer RTL:** Behavioral RTL primarily focuses on the algorithmic description of the circuit's functionality and is used during the early design phases for system-level analysis. Register Transfer RTL, on the other hand, explicitly represents the flow of data between registers and the associated control logic and is used for logic synthesis and hardware implementation.

**2.1.6. Combinational RTL vs. Sequential RTL:** Combinational RTL describes digital circuits where the output depends solely on the current input, without considering any previous state. Sequential RTL, on the other hand, includes elements of memory (registers or flip-flops) to store and maintain state information between clock cycles.

Each classification of RTL serves a specific purpose in the digital design flow and aids in different stages of the design process, from system-level exploration to gate-level implementation. The choice of RTL type depends on the design requirements, the target technology (ASIC or FPGA), and the goals of the design team.

## 2.2. Integrated Circuit

IC stands for Integrated Circuit. It is a miniaturized electronic circuit that consists of multiple electronic components (such as transistors, resistors, capacitors, and diodes) and their interconnections built on a single semiconductor material or chip. The components and interconnections are fabricated using various semiconductor manufacturing processes to create a functional electronic circuit.

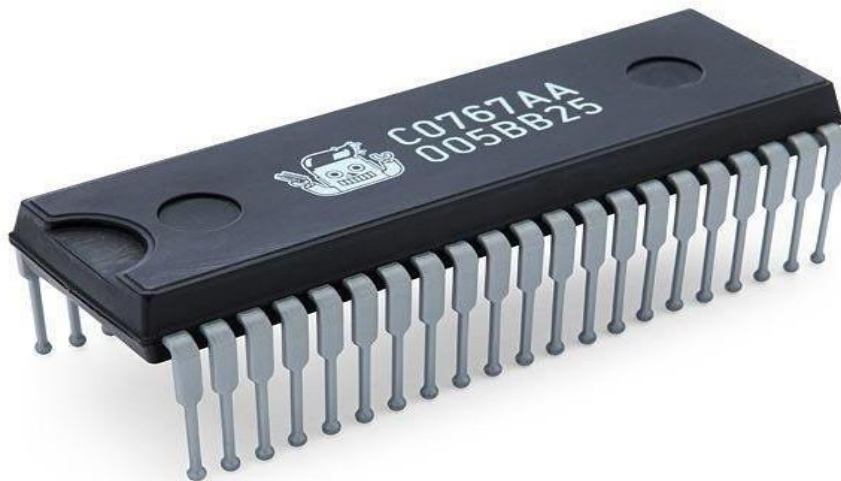


Figure 2. 1 Integrated Circuit

The invention of the integrated circuit revolutionized the electronics industry and paved the way for modern computing and electronic devices. Prior to ICs, electronic circuits were built using discrete components, which were large, bulky, and consumed more power. ICs enabled a significant reduction in size, power consumption, and cost, while also increasing the performance and reliability of electronic systems.

Integrated circuits come in various types and sizes, ranging from simple logic gates and microcontrollers to complex microprocessors and memory chips. They can be categorized into different types based on their functionality and application, such as:

**2.2.1. Analog Integrated Circuits (ICs):** These circuits process continuous signals and are used in applications such as amplification, filtering, and signal conditioning.

**2.2.2. Digital Integrated Circuits (ICs):** These circuits process discrete binary signals (0s and 1s) and are the foundation of modern digital electronics, used in computers, microcontrollers, and digital signal processors (DSPs).

**2.2.3. Mixed-Signal Integrated Circuits (ICs):** These circuits combine both analog and digital components and are used in applications like analog-to-digital converters (ADCs) and digital-to-analog converters (DACs).

**2.2.4. Memory Integrated Circuits:** These ICs are used for storing and retrieving data in electronic devices. Examples include RAM (Random Access Memory) and ROM (Read-Only Memory) chips.

**2.2.5. Microprocessors and Microcontrollers:** These are specialized digital ICs that contain a central processing unit (CPU) along with other components, making them the "brains" of computers and embedded systems, respectively.

ICs are widely used in various industries, including telecommunications, consumer electronics, automotive, aerospace, and healthcare, among others. Their compact size, low power consumption, and high reliability have made them an indispensable part of modern electronic devices and systems.

## 2.3. IC Design

IC design can be categorized into three main categories based on the type of circuits being designed:

**2.3.1. Digital IC Design:** This category focuses on designing integrated circuits that process discrete binary signals (0s and 1s). Digital ICs are the building blocks of modern digital electronic devices, such as microprocessors, microcontrollers, memory chips, and application-specific integrated circuits (ASICs). Digital IC design involves creating logic circuits, sequential circuits, and various digital components to implement complex digital systems.

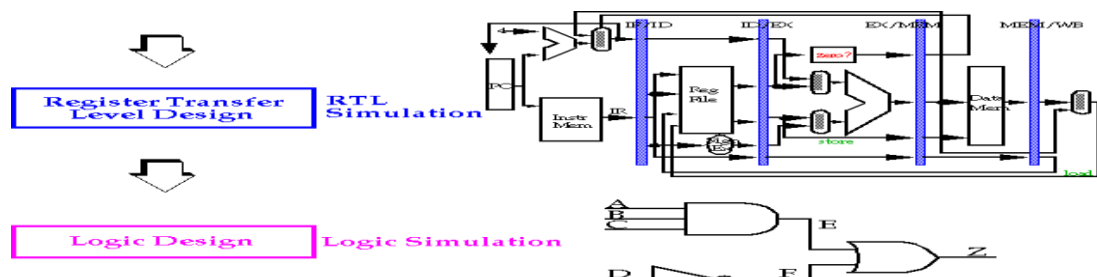
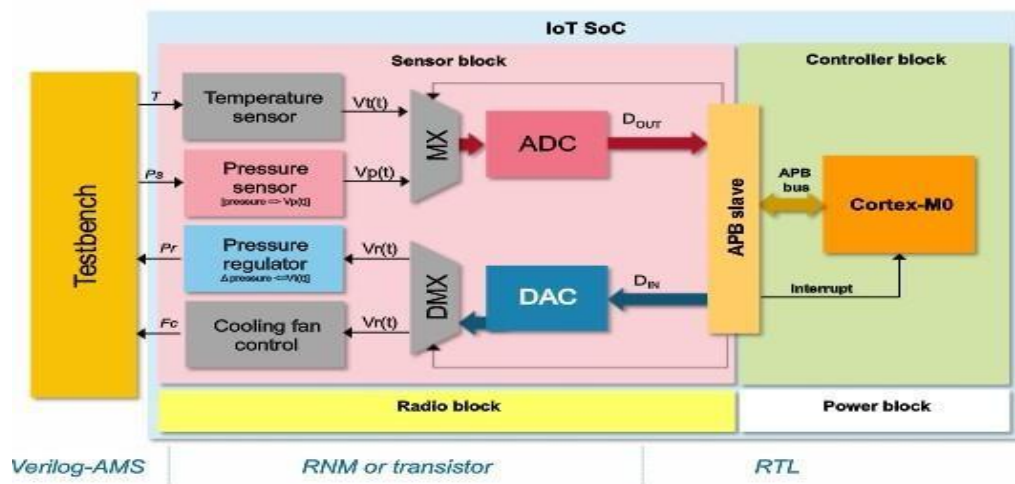


Figure 2. 2 Digital IC design process

**2.3.3. Mixed-Signal IC Design:** Mixed-signal IC design combines both analog and digital components on the same chip. These circuits interface between the analog and digital domains, enabling seamless communication between the two. Mixed-signal ICs are prevalent in applications like data communication, sensor interfaces, power management, and signal processing.



## 2.4. Types of ICs

- i. ASIC: Application-Specific Integrated Circuit
- ii. ASSP: Application-Specific Standard Product
- iii. SoC: System-on-Chip
- iv. FPGA: Field-Programmable Gate Array
- v. SoC FPGA: System-on-Chip FPGA
- vi. Microcontroller
- vii. Microprocessor

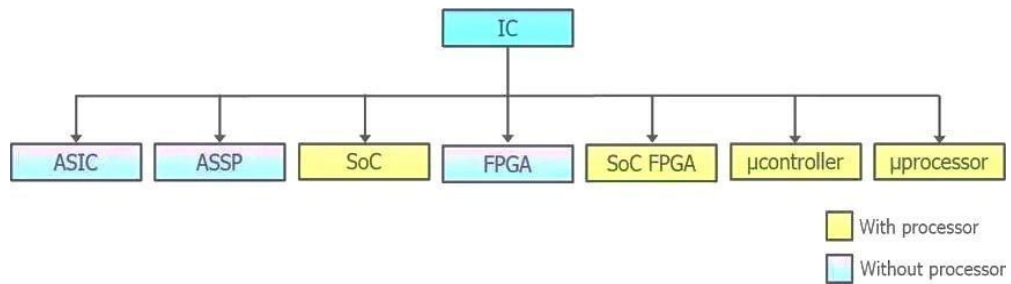


Figure 2. 4 Types of ICs

#### 2.4.1. ASIC (Application-Specific Integrated Circuit):

An ASIC can be defined as an integrated circuit customized for a particular application, specifically tailored to meet the specific requirements of that application. These chips are designed from the groundup, starting with a clean slate, to ensure optimal performance and efficiency for the intended purpose. Due to their bespoke nature, ASICs are ideally suited for high-volume productions where the cost can be justified by the increased performance and functionality tailored precisely for the application's needs. While ASIC development involves higher initial costs, the benefits of unparalleled performance and power efficiency make them a preferred choice in scenarios demanding high levels of customization and efficiency.



Figure 2. 5 Application-Specific Integrated Circuit (ASIC)

The main characteristics of ASICs are as follows:

- 1) Performs the same function throughout its lifetime.
- 2) Does not have a processor.
- 3) Design cycle is time-consuming and expensive.
- 4) Manufactured in higher volumes.
- 5) High speed and low power consumption.
- 6) Can be digital, analog, or both.

**Applications:** Routers, switches, modems, etc.



### 2.4.2. ASSP (Application Specific Standard Part):

Designed for a specific application, much like ASICs, ASSP (Application-Specific Standard Product) offers a targeted solution to meet particular requirements. However, unlike ASICs, ASSPs are not customized for a specific system or customer. Instead, they are more generalized devices intended for

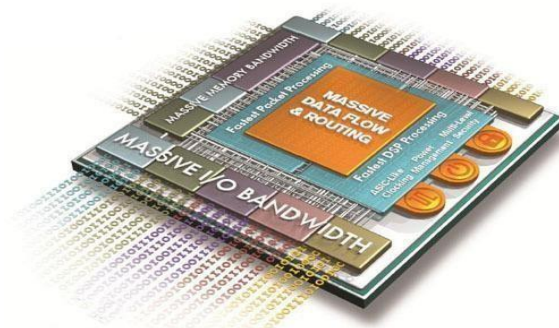


Figure 2. 6 Application Specific Standard part

use by multiple system design houses. While an ASSP shares similarities with an ASIC in terms of application focus, its key distinction lies in its market approach. An ASSP is manufactured and sold to more than one company, making it a standardized product that can be used across various applications and industries, providing a cost-effective and time-efficient solution for companies with similar needs.

The company manufactures a range of ASSP products tailored for specific applications, including Ethernet controllers, PCIe controllers, USB interfaces, and more.

### 2.4.3. FPGA (Field Programmable Gate Array):

FPGAs (Field-Programmable Gate Arrays) are versatile integrated circuits programmed to execute a tailored function for a specific purpose. The FPGA programming code is written in Hardware Description Languages (HDL) such as VHDL and Verilog, enabling designers to define complex functionalities with ease. The beauty of FPGAs lies in their programmable interconnections, offering users the flexibility to modify and adapt the circuit to perform desired functions conveniently.

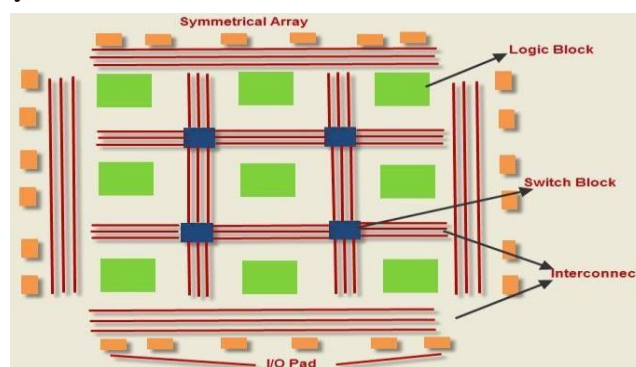


Figure 2. 7 Field Programmable Gate Array



Unlike ASICs (Application-Specific Integrated Circuits), FPGAs do not require custom mask layers, which significantly accelerates the design implementation process from weeks to a matter of hours. This rapid turnaround time empowers designers to swiftly prototype and iterate through designs, making FPGAs an attractive solution for a wide range of applications and industries.

#### 2.4.4. SoC (System on Chip):

System-on-Chip (SoC) refers to a compact and efficient integration of all vital components within a device, such as CPU, GPU, Memory, Ports, and DRAMs, into a single chip. Whether it's an ASIC or ASSP with a processor, the result is an SoC. Within the SoC, you may find a microprocessor ( $\mu$ P or MPU), microcontroller ( $\mu$ C or MCU), digital signal processor (DSP), or graphics processor. To put it into perspective, envision the entire motherboard of your computer condensed into a tiny chip, and that's precisely what a SoC represents. This level of integration ensures seamless performance, power efficiency, and reduced space requirements, making SoCs a fundamental building block in modern electronic devices.

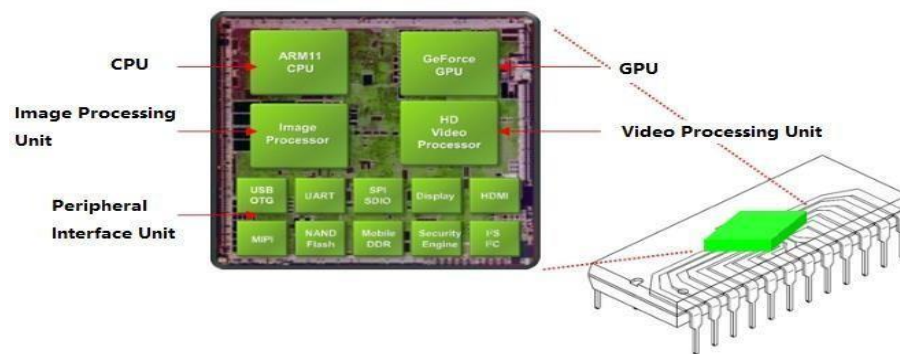


Figure 2. 8 System on Chip

#### 2.4.5. Microprocessor:

A CPU built on a single Integrated Circuit (IC) is called a microprocessor. It is a remarkable electronic device, programmable and multipurpose, driven by a clock and based on registers. The microprocessor's primary function is to execute a collection of machine instructions that guide its actions.

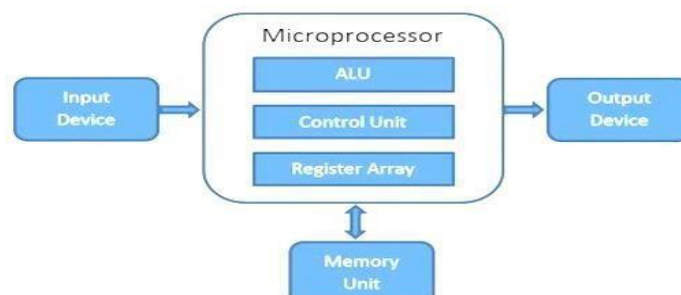


Figure 2. 9 Microprocessor and its different accessories

There are three fundamental tasks that a microprocessor can accomplish:

- **Mathematical Operations:** The microprocessor is equipped with an Arithmetic/Logic Unit (ALU), enabling it to perform various mathematical operations such as addition, subtraction, multiplication, and division. In modern microprocessors, there are advanced floating-point processors capable of handling intricate operations on large floating-point numbers.
- **Data Movement:** Another essential function of a microprocessor is to transfer data from one memory location to another. This ability enables efficient storage and retrieval of information during the execution of programs.
- **Decision-Making and Branching:** A significant aspect of a microprocessor's capabilities lies in its ability to make decisions based on certain conditions or inputs. Depending on the outcome of these decisions, the microprocessor can jump to a new set of instructions, allowing for dynamic and conditional execution paths.

These three core functionalities make microprocessors the central processing unit of computers and various other electronic devices. Their versatility and power have been instrumental in driving the advancement of technology, making them a cornerstone of modern computing.

## Chapter 3

### Verilog

HDL stands for "Hardware Description Language." It is a specialized type of programming language used in digital circuit design and hardware development. HDLs enable engineers and designers to describe the behavior and structure of digital hardware systems, such as microprocessors, integrated circuits, FPGA (Field-Programmable Gate Array) designs, and other digital logic circuits.

There are two primary hardware description languages: Verilog and VHDL (VHSIC Hardware Description Language). Engineers use these languages to model the behavior of digital circuits at various levels of abstraction, ranging from high-level functional specifications to low-level gate-level representations.

Using HDL, designers can define the functionality of hardware components, describe the interactions between various hardware modules, and specify the data flow within the system. The HDL code serves as a blueprint for the actual hardware implementation, whether it's in an ASIC (Application-Specific Integrated Circuit), FPGA, or other custom hardware.

HDLs are essential in the hardware design process because they offer several advantages:

- **Abstraction:** HDL allows designers to focus on the functional behavior of the hardware without worrying about the specific details of the underlying physical implementation.
- **Simulation:** HDL code can be simulated to verify the correctness of the design before committing to the costly fabrication process.
- **Reusability:** HDL promotes modular design, enabling designers to reuse and adapt existing components in new projects.
- **Flexibility:** HDL-based designs can be easily modified and iterated upon, reducing the time to market for new hardware products.

Both Verilog and VHDL have their strengths and are widely used in the industry. The choice between the two often depends on the design team's preferences, existing infrastructure, and project requirements. This chapter will provide a description of Verilog.

### 3.1. VHDL

VHDL, which stands for VHSIC Hardware Description Language, is a hardware description language widely used in digital circuit design and hardware development. Developed in the 1980s by the U.S. Department of Defense as part of the VHSIC (Very High-Speed Integrated Circuit) program, VHDL has become an IEEE standard (IEEE 1076) and an industry-standard language for hardware description.

VHDL serves as a powerful programming language that allows engineers and designers to describe the behavior and structure of digital hardware systems. With VHDL, designers can model complex digital circuits, such as microprocessors, integrated circuits, FPGA (Field-Programmable Gate Array) designs, and other digital logic circuits.

In VHDL, designers can define the functionality of hardware components, specify the interactions between different hardware modules, and describe the data flow within the system. The language supports various levels of abstraction, ranging from high-level behavioral descriptions to low-level structural representations, catering to different design needs and styles.

One of the key strengths of VHDL is its suitability for verification and testing. VHDL code can be used to simulate the behavior of the designed hardware, enabling designers to test and validate the correctness and functionality of their designs before proceeding to the actual fabrication process.

VHDL promotes modularity and reusability in hardware design. Designers can create reusable and scalable modules, which can be easily integrated into various projects. This approach streamlines the design process, reduces duplication of effort, and fosters a structured and organized design methodology.

Additionally, VHDL is suitable for system-level design, enabling designers to describe not only digital circuits but also complete systems that include multiple interconnected hardware components and software elements.

Over time, VHDL has evolved, and various extensions and enhancements have been introduced, such as VHDL-93 and VHDL-2008, which offer additional features and improvements for design and verification tasks.

Overall, VHDL remains a significant and widely adopted hardware description language, offering powerful capabilities for modeling, simulating, and implementing complex digital systems. As technology continues to advance, VHDL will continue to play a crucial role in the hardware design process, empowering engineers to create innovative and reliable digital solutions.

### **3.2. Verilog**

Verilog is a hardware description language (HDL) widely used in digital circuit design and hardware development. Originally developed by Gateway Design Automation in the early 1980s, it became an IEEE standard (IEEE 1364) in 1995, which further solidified its popularity in the industry.

As an HDL, Verilog serves as a programming language for describing the behavior and structure of digital hardware systems. It allows engineers and designers to model complex digital circuits, such as microprocessors, integrated circuits, FPGA (Field-Programmable Gate Array) designs, and other digital logic circuits. The key strength of Verilog lies in its ability to provide a higher-level abstraction that simplifies the representation of hardware behavior.

In Verilog, designers can define the functionality of hardware components, specify how different hardware modules interact and communicate with each other, and describe the data flow within the system. The language offers different levels of abstraction, ranging from high-level functional specifications to low-level gate-level representations. This flexibility allows designers to work at a level that matches their specific project requirements and expertise.

The Verilog code serves as a blueprint for the actual hardware implementation, whether it's in an ASIC (Application-Specific Integrated Circuit), FPGA, or other custom hardware. This code can be synthesized into a gate-level netlist, which is then used for the physical design and manufacturing of the hardware.

Using Verilog, designers can take advantage of several essential features that facilitate efficient and reliable hardware design:

- **Abstraction:** Verilog allows designers to focus on the functional behavior of the hardware, shielding them from the intricacies of the underlying physical implementation. This abstraction simplifies the design process and enables the creation of complex circuits with relative ease.
- **Simulation:** Verilog code can be simulated to verify the correctness and functionality of the design. Simulation plays a crucial role in catching errors, validating designs, and ensuring that the hardware functions as intended before fabrication.
- **Modularity and Reusability:** Verilog promotes a modular design approach, enabling designers to create reusable components that can be easily integrated into different projects. This modularity simplifies the design process, reduces redundancy, and fosters a more organized and scalable design methodology.
- **Time-to-Market:** The flexibility of Verilog-based designs allows for quicker iterations and modifications, resulting in shorter development cycles and faster time-to-market for new hardware products.

Over the years, Verilog has evolved, and new standards have been introduced, such as System Verilog, which incorporates additional features for verification and design automation. Despite these advancements, Verilog remains a fundamental and widely adopted HDL, playing a crucial role in shaping the modern digital design landscape. As technology continues to advance, Verilog will continue to be a valuable tool for hardware designers, enabling them to create innovative and efficient digital systems.

Table 3. 1 The differences between VHDL and Verilog

Feature	VHDL	Verilog
<b>Origin</b>	Developed by DoD (VHSIC program) in the early 1980s.	Developed by Gateway Design Automation in the early 1980s.
<b>Abstraction Levels</b>	Supports high-level and low-level behavioural descriptions.	Primarily used for low-level behavioral and structural descriptions.
<b>Coding Style</b>	Emphasizes verbose and explicit coding style.	Emphasizes compact and concise coding style.
<b>Ecosystem</b>	Widely used in Europe and academia.	Popular in the USA and industry.
<b>Usage</b>	Preferred for complex systems and system-level design.	Preferred for hardware design, especially for ASIC/FPGA.
<b>Strengths</b>	Strongly typed, with rich simulation capabilities, and more suitable for complex designs.	More concise, better suited for rapid prototyping and synthesis.
<b>Modularity</b>	Supports records, packages, and procedures for modularity.	Supports modules and interfaces for modularity.
<b>Reusability</b>	Emphasizes code reusability and scalable designs.	Supports code reusability and component-based design.
<b>Documentation</b>	Generally involves more verbose and detailed documentation.	Generally involves less documentation.

### 3.3. Abstractions Level of Verilog

There exist four distinct abstraction levels that can be utilized to execute Verilog code. These levels of abstraction encompass switch level, gate level, dataflow level, and behavioral level. Indeed, the syntax employed at each abstraction level in Verilog differs due to the varying levels of detail and granularity they represent.

#### 3.3.1. Switch Level

The switch level is an even more detailed level of abstraction. It involves describing the behavior of individual transistors within a digital circuit. This level is often used in highly specialized cases, such as designing analog circuits or dealing with specific power or noise considerations. Switch-level modeling provides precise control over transistor behavior.

At the switch level, we're working with transistors and their characteristics directly. This level is quite low-level and rarely used in modern digital design. Here's a highly simplified example:

```

module
    and_gate_switc
    h_level (input
    wire a, b,
    output wire out
    );

    nmos n1 (a, out, 1);
    nmos n2

    (b, out, 1);

endmodule

module nmos (input, output, control);
    // Simplified NMOS
    implementationEndmodule




```

### 3.3.2. Gate Level

At this level, the module is implemented using logic gates. The Gate Level represents the lowest level of abstraction, where fundamental logic gates are accessible as pre-defined primitives. The syntax for this involves utilizing the primitive's name and instance name along with specified output and input connections.

Syntax: Primitive\_name instance\_name(output, inputs)

Table 3. 2 Example of Gate Level

1. AND G1(out,A,B)	
2. NAND G2(out,A,B)	
3. OR G3(out,A,B)	

### 3.3.3. Data Flow Level

At this level, the module is designed through the specification of data flow, where signals are allocated using equations that manipulate the data. The design is realized using continuous assignments. All these assignments operate concurrently, employing the 'assign' keyword. Here are a few examples:

```

assign Z = ix & y;
assign p = q & r;
assign z = ~y;

```

### 3.3.4. Behavioral Level

At the pinnacle of abstraction in Hardware Description Languages (HDL) lies the Behavioral Level. This level serves as the highest tier of HDL abstraction. It provides an encompassing view of a system by describing its behavior rather than delving into intricate implementation details. Within this framework, a range of elements, including functions, tasks, and blocks, is employed to capture the essence of the system's operation. At the core of the Behavioral Level are two pivotal constructs: 'initial' and 'always'. These constructs enable designers to outline the initial values and the continuous behavior of various components, ensuring a comprehensive representation of a system's functionality. This abstraction layer proves instrumental in conceiving and visualizing complex systems before moving onto more detailed design considerations at lower abstraction levels.

Example: (2\*1 mux)

```

always @(a, b,
      sel)
begin
  if
    (sel)
    out = b;
  else
    out = a;
end

```

## 3.4. Modules

In the realm of Verilog, a module stands as the fundamental cornerstone. Serving as the foundational unit, a module can encompass individual elements or even comprise an assemblage of lower-level design blocks. This versatile entity offers vital functionality to higher-level constructs through its defined port interface, adeptly concealing its internal workings. This encapsulation affords a crucial advantage: it empowers designers to make alterations to the module's inner workings without engendering disturbances across the wider design. This modular approach forms a linchpin of efficient design, enabling seamless modifications and enhancements while upholding the integrity of the overall architecture.

### Verilog Framework:

```

module module_name (x,y,z);
  input x,y;
  output z;
  statements ;
  ..... ;
  ..... ;
endmodule

```

Figure 3. 1 Verilog Framework



Instantiation within Verilog holds the key to constructing a well-organized hierarchy within its descriptions. This essential process entails generating concrete objects from module templates, and these generated entities are aptly termed instances. While embarking on this approach, it's important to note that Verilog prohibits the nesting of modules—embedding one module definition within another is not permitted. However, this restriction doesn't hinder the integration of other modules into a design. By means of instantiation, we can seamlessly integrate copies of existing modules, forging a structured and modular framework. This practice not only promotes systematic design but also facilitates the amalgamation of distinct functional components to bring forth a cohesive and intricate system.

Examples: 4:1 mux using 2:1 mux

```
1. module mux_2to1 (i0, i1,
    sel, out);
    input i0, i1,
    sel;
    output out;
    always
    @(i0, i1,
    sel)begin
        if (sel)
            out = i1;
        else
            out = i0;
    end
endmodule
```

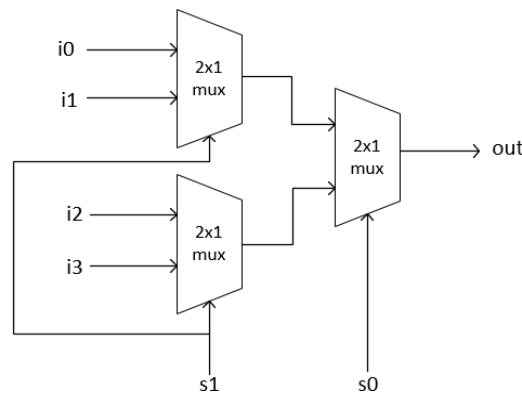


Figure 3. 2 Mux 4:1 using 2:1 mux

```
2. module mux_4to1 (i0, i1, i2, i3, s1, s0,
    out);
    input i0, i1, i2, i3, s1, s0;
    output out;
    wire x1, x2;
    mux_2to1 ma (i0, i1, s1, x1);
    mux_2to1 m2 (i2, i3, s1, x2);
    mux_2to1 m3 (x1, x2, s0, out);
endmodule
```

### 3.5. Simulation, Synthesis, and Design methodology in Verilog

Simulation plays a pivotal role in validating the integrity of digital designs constructed using Hardware Description Languages (HDL) such as Verilog. To achieve this, various input stimuli are meticulously applied to the design at distinct points in time. This process serves the essential purpose of assessing whether the Register-Transfer Level (RTL) code conforms to its intended functionality. Through methodical simulation, the behavior of the digital design can be thoroughly examined, ensuring that it operates in the envisaged manner.

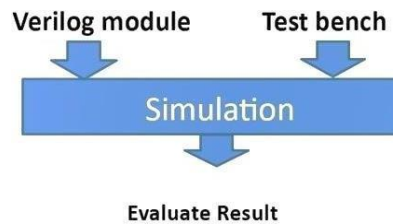


Figure 3. 3 Simulation model

Synthesis represents a pivotal stage wherein a digital design, conceptualized using Hardware Description Languages (HDL), undergoes a transformation into a tangible implementation defined by a network of logic gates. This intricate process involves translating the abstract design into a concrete configuration, optimized according to the chosen operational strategy. Additionally, synthesis provides insight into resource utilization, offering an assessment of the available resources and how they are consumed in the realized design. Through synthesis, the bridge is built between the high-level design concept and the physical realization, offering a crucial step in bringing digital designs to life.

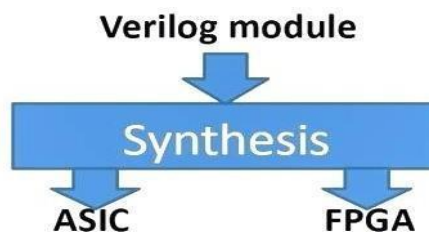


Figure 3. 4 Synthesis model

In the realm of Very Large-Scale Integration (VLSI) design, two fundamental approaches are employed: top-down design methodology and bottom-up design methodology. These methodologies offer distinct strategies for creating complex integrated circuits (ICs) while catering to diverse aspects of the design process.

#### 3.5.1. Top-down design methodology

The initial step in our design process involves outlining the top-level block and discerning the essential sub-blocks indispensable for constructing the overarching structure. These sub-blocks subsequently undergo further division, cascading down the hierarchy until we reach the leaf-cells—the elemental building blocks that resist further segmentation. This hierarchical deconstruction enables a comprehensive understanding of the system's architecture, allowing for meticulous planning and efficient realization.

### 3.5.2. Bottom-Up design methodology

The foundational approach commences with the identification of the available building blocks at our disposal. These fundamental units serve as the groundwork upon which we assemble larger, more intricate blocks. This cascading arrangement facilitates the creation of cells, which are subsequently employed in constructing higher-level blocks. This modular progression continues until we achieve the realization of the design's pinnacle: the top-level block. In this manner, a structured hierarchy is established, ensuring a systematic and cohesive development process.

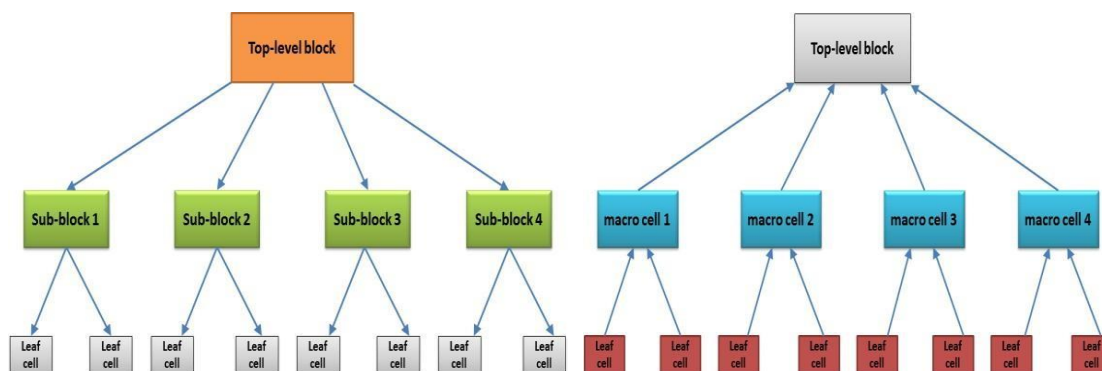


Figure 3. 5 Design methodology (a) Top-down (b) Bottom-up

## 3.6. Data Type

A data type serves as a classification framework that delineates the nature of values permissible for assignment to a variable. It essentially defines the realm within which a variable can operate, influencing the kind of operations that can be performed on it. This classification also extends to determining the array of mathematical operations applicable to the particular data type, thus shaping the permissible computations that can be carried out with precision and coherence. Two data types are used in Verilog system, they are-

**3.6.1. Register data type:** At its core, a register stands as a foundational data storage element within digital systems. Operating akin to a variable, it possesses the ability to contain and retain a value, making it a fundamental component for data manipulation and storage. Declared through the keyword "reg," registers are integral to the realm of digital design. Their default value, commonly denoted as 'X,' underscores their uninitialized nature. Furthermore, registers encapsulate a broader classification of data types, encompassing a range of variables including integers, real numbers, and timestamps. By accommodating this spectrum of values, registers offer versatility and functionality that underpin the intricate workings of digital circuits. Verilog supports various register data types, including:

- **reg:** The most commonly used data type for general-purpose data storage.
- **integer:** Used primarily for loop counting and numerical operations.
- **real:** Employed to store floating-point numbers
- **time:** Tracks simulation time for accurate modeling

Register data type usage:

- Single-bit register variable: reg count;
- 8-bit bus: reg [7:0] bus;
- Registers represent fundamental data storage elements in Verilog.
- They function as variables capable of holding values.
- Registers are typically declared using the keyword "reg."
- The default value of a "reg" data type is commonly represented as 'X'.
- The register data type encompasses various subtypes, including "reg," "integer," "real," and "time."

Example :

```
reg reset;
initial begin|
    reset = 1'b0;
    #10 reset = 1'b1;
    #10 reset = 1'b0;
End
```

**3.6.2. Net data type :** Nets are a fundamental aspect of digital circuit design, embodying the connectivity that links various hardware elements together. Unlike storage elements like registers, nets do not possess the capability to retain values; they must be in a state of continuous drive. Typically, the keyword "wire" is employed to declare nets. Their default value, often represented as 'z' or 'x', signifies their unassigned state. Moreover, nets encompass a broader category of data types including wire, wand, wor, tri, triand, trior, and trireg. Through this versatile range, nets provide the essential pathways for signal propagation, ensuring seamless communication among diverse components within the circuit.

- Nets represent connections between hardware elements.
- They must be continuously driven, meaning they cannot store values.
- Nets are primarily declared using the keyword "wire".
- The default value of nets is often 'z' or 'x'.
- The net data type encompasses a variety of classes such as wire, wand, wor, tri, triand, trior, trireg, etc.

Example :

```
module half add (
    input a, b,
    output sum, carry
);
    wire sum, carry;
    assign sum = a ^ b;
    assign carry = a & b;
endmodule
```

## 3.7. Operators in Verilog

Verilog, a versatile hardware description language, equips designers with an array of operators catering to diverse aspects of digital system modeling. These operators, classified into distinct categories, facilitate the precise expression of circuit behavior and functionality.

### 3.7.1. Arithmetic Operators:

- Addition (+): Performs addition between operands.  
Example: `sum = a + b;`
- Subtraction (-): Performs subtraction between operands.  
Example: `difference = x - y;`
- Multiplication (\*): Performs multiplication between operands.  
Example: `product = p * q;`
- Division (/): Performs division between operands.  
Example: `quotient = dividend / divisor;`
- Modulus (%): Computes the remainder of division between operands.  
Example: `remainder = num % divisor;`

### 3.7.2. Logical Operators:

- AND (&&): Performs logical AND operation between operands.  
Example: `result = (a && b);`
- OR (||): Performs logical OR operation between operands.  
Example: `output = (x || y);`
- NOT (!): Performs logical NOT operation on a single operand.  
Example: `output = !flag;`

### 3.7.3. Bitwise Operators:

- AND (&): Performs bitwise AND operation between operands.  
Example: `result = a & b;`
- OR (|): Performs bitwise OR operation between operands.  
Example: `output = x | y;`
- XOR (^): Performs bitwise XOR operation between operands.  
Example: `result = p ^ q;`
- Bitwise Complement (~): Inverts all the bits of an operand.  
Example: `result = ~data;`

### 3.7.4. Equality Operators:

- Equal to (==): Checks if two operands are equal.  
Example: `is_equal = (a == b);`
- Not equal to (!=): Checks if two operands are not equal.  
Example: `not_equal = (x != y);`

### 3.7.5. Relational Operators:

- Greater than (>): Checks if the left operand is greater than the right.  
Example: `greater = (p > q);`

- Less than (<): Checks if the left operand is less than the right.
- Example: `lesser = (x < y);`
- Greater than or equal to (>=): Checks if the left operand is greater than or equal to the right.
- Example: `greater_equal = (a >= b);`
- Less than or equal to (<=): Checks if the left operand is less than or equal to the right.
- Example: `lesser_equal = (x <= y);`

#### **3.7.6. Reduction Operator:**

- Bitwise AND (&): Reduces multiple bits using a bitwise AND operation.
- Example: `result = &{a, b, c};`

#### **3.7.7. Shift Operator:**

- Left Shift (<<): Shifts bits to the left by a specified number of positions.
- Example: `shifted = data << 2;`
- Right Shift (>>): Shifts bits to the right by a specified number of positions.
- Example: `shifted = value >> 3;`

#### **3.7.8. Concatenation:**

- Concatenation ({ }): Combines different bit sequences into a single value.
- Example: `combined = {a, b, c};`

#### **3.7.9. Conditional Operator:**

- Conditional (`condition ? true_expression : false_expression`): Provides a shorthand for conditional expressions.
- Example: `largest = (a > b) ? a : b;`

These examples showcase how each operation is applied to operands and how they yield results based on the specified conditions or calculations.

## Chapter 4

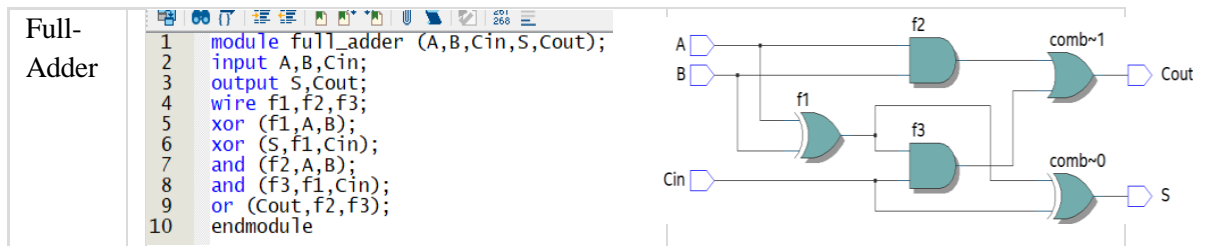
### Verilog Codes with Outputs

In the upcoming chapter, I will delve into the Verilog code implementations presented in our attachment. Additionally, this chapter will showcase the corresponding outputs generated by these codes.

#### 4.1. Basic Logic Gates

Table 4. 1 Basic Logic Gates

Gate/ Ckt name	Code	RTL Output
AND gate	<pre> 1 module basic_and (A,B,Y); 2   input A,B; 3   output Y; 4   and (Y,A,B); 5 endmodule </pre>	
NAND gate	<pre> 1 module basic_nand (A,B,Y); 2   input A,B; 3   output Y; 4   nand (Y,A,B); 5 endmodule </pre>	
NOT gate	<pre> 1 module basic_not (in,out); 2   input in; 3   output out; 4   not (out,in); 5 endmodule </pre>	
OR gate	<pre> 1 module basic_or(in1,in2,out); 2   input in1,in2; 3   output out; 4   or (out,in1,in2); 5 endmodule </pre>	
XNOR gate	<pre> 1 module basic_xnor (A,B,Y); 2   input A,B; 3   output Y; 4   xnor (Y,A,B); 5 endmodule </pre>	
XOR gate	<pre> 1 module basic_xor (A,B,Y); 2   input A,B; 3   output Y; 4   xor (Y,A,B); 5 endmodule </pre>	
Half- Adder	<pre> 1 module half_adder (a,b,s,c); 2   input a,b; 3   output s,c; 4   xor (s,a,b); 5   and (c,a,b); 6 endmodule </pre>	



## 4.2. Multiplexers & demultiplexers

### 4.2.1. Mux 2:1

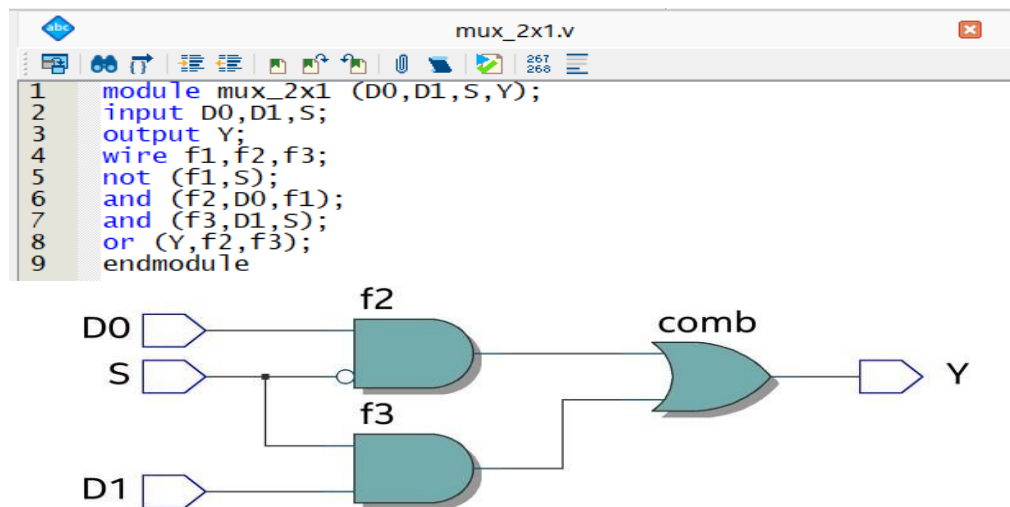


Figure 4. 1 Code & RTL output of 2:1 Mux

### 4.2.2. Mux 4:1

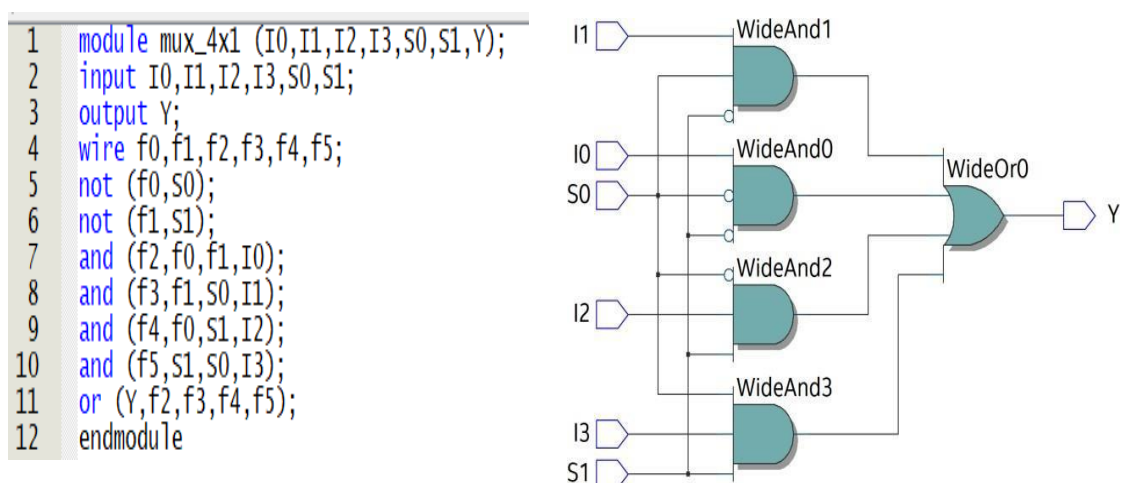


Figure 4. 2 Code & RTL output of 4:1 Mux



### 4.2.3. Demultiplexers (1x8)

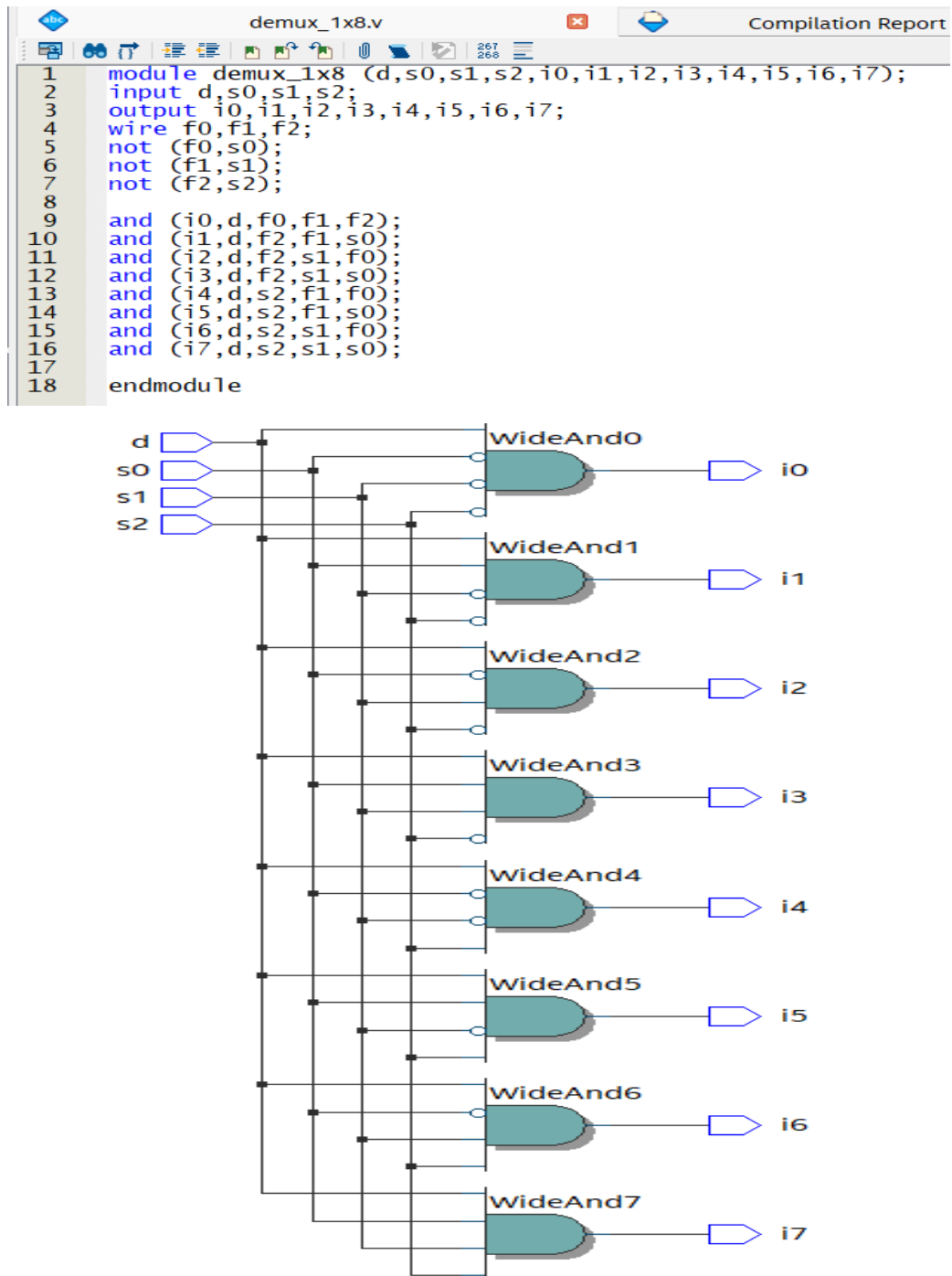


Figure 4. 3 Code & RTL output of Demux 1:8

## 4.3. Encoder & Decoder

### 4.3.1. Encoder Circuit (8x1)

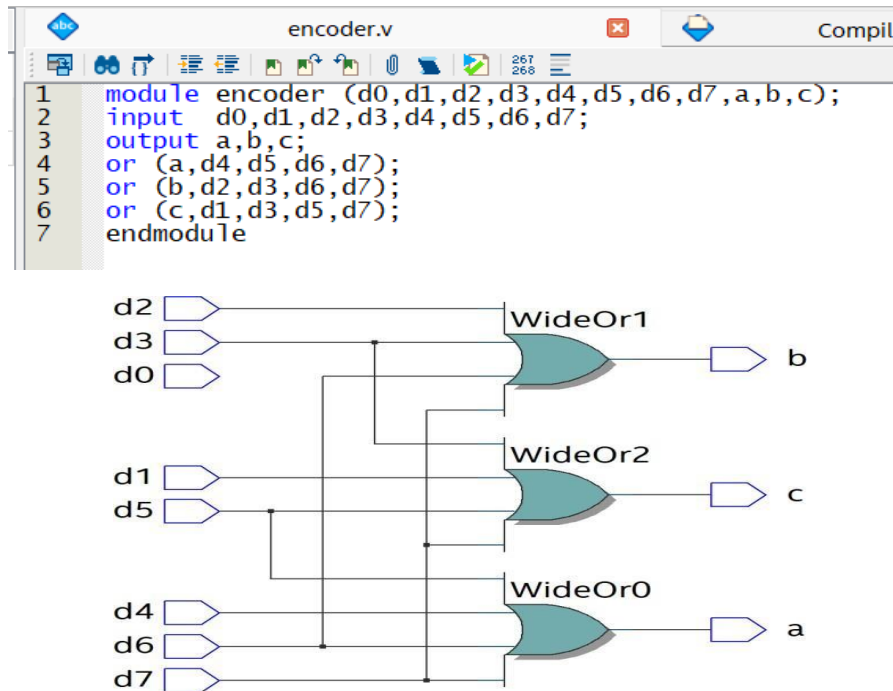


Figure 4. 4 Code & RTL output of Encoder (8x1)

### 4.3.2. Decoder Circuit (3x8)

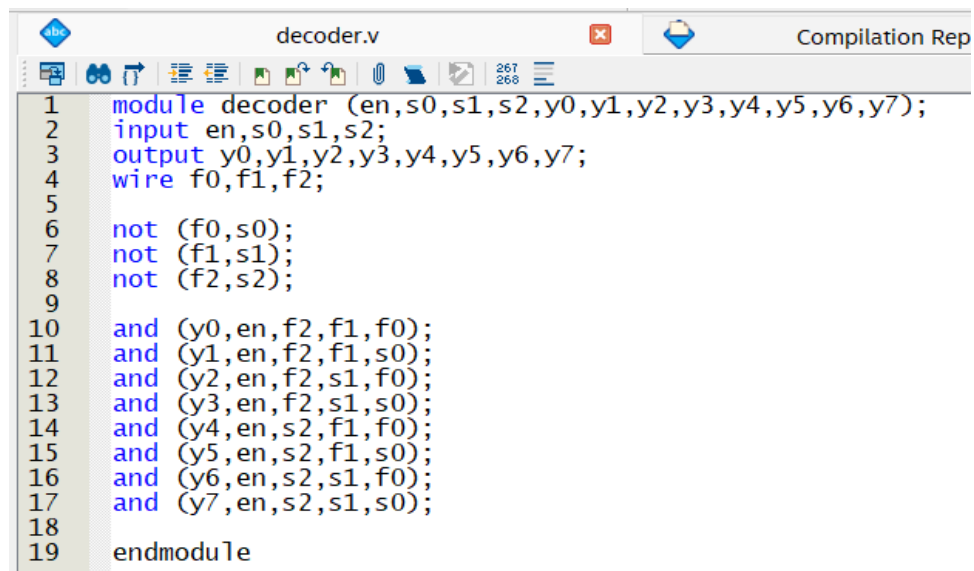


Figure 4. 5 Decoder 3x8 code

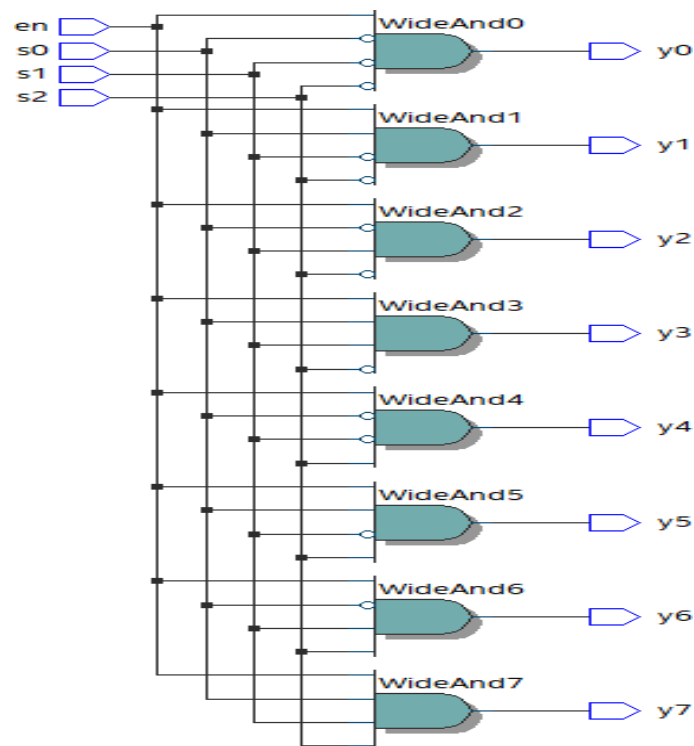


Figure 4. 6 RTL output of 3x8 Decoder

#### 4.4. Even Parity Checker

```

1  module evenparitychecker (a,b,c,p,y);
2  input a,b,c,p;
3  output y;
4  wire f1,f2;
5
6  xor (f1,a,b);
7  xor (f2,c,p);
8  xor (y,f1,f2);
9
10 endmodule

```

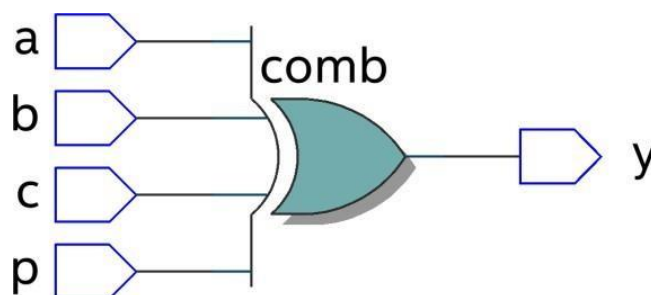


Figure 4. 7 Code & RTL output of Parity Checker

## 4.5. Adder and Subtractor

### 4.5.1. Half Adder

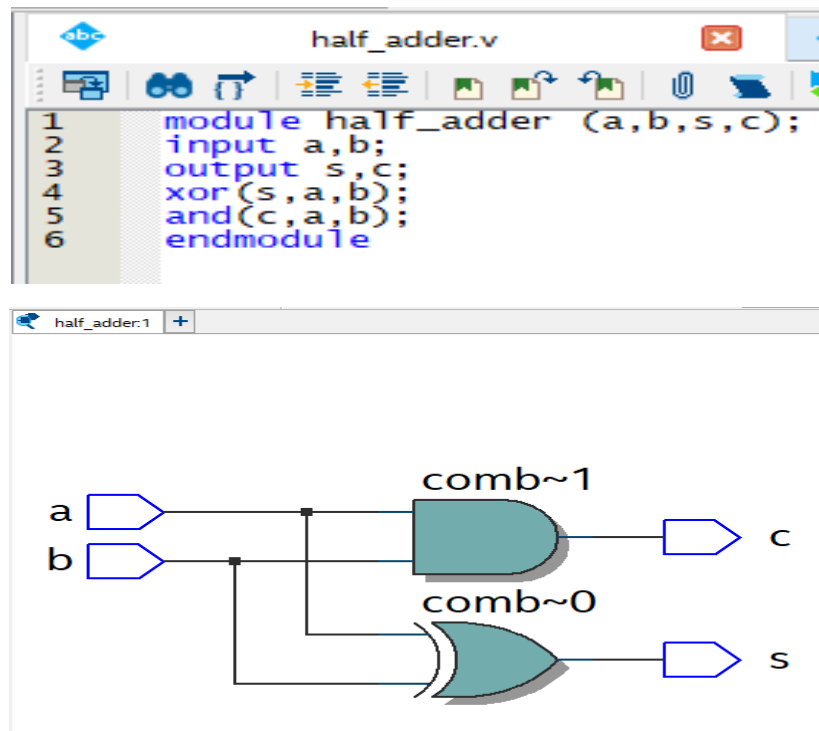


Figure 4. 8 Code & RTL output of Half adder

### 4.5.2. Full Adder

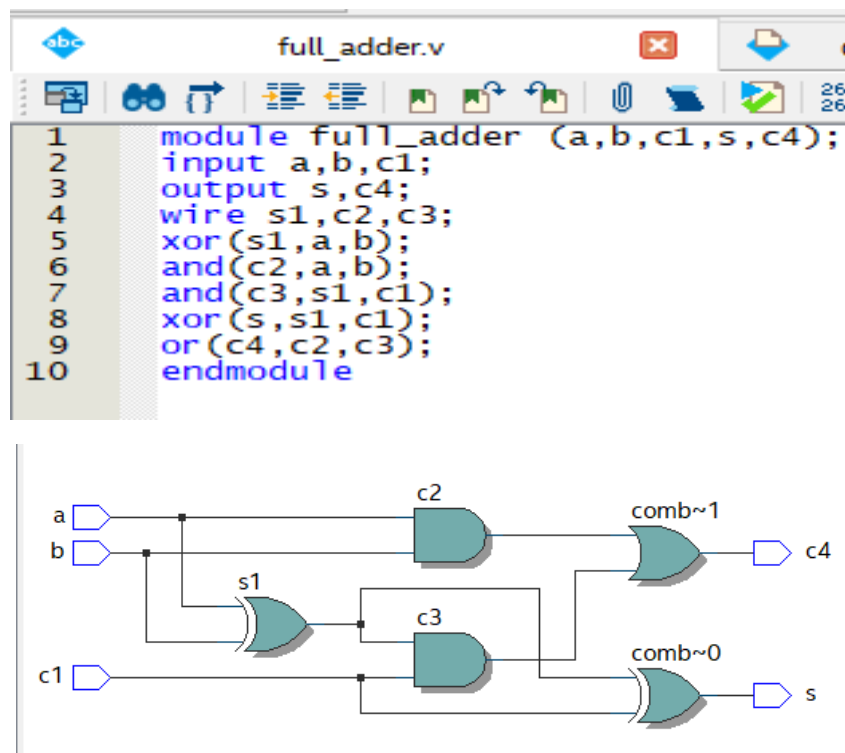


Figure 4. 9 Code & RTL output of Full Adder

### 4.5.3. Half-Subtractor

```

1  module half_subtractor (a,b,diff,bor);
2  input a,b;
3  output diff,bor;
4  wire f1;
5
6  xor (diff,a,b);
7  not (f1,a);
8  and (bor,f1,b);
9
10 endmodule

```

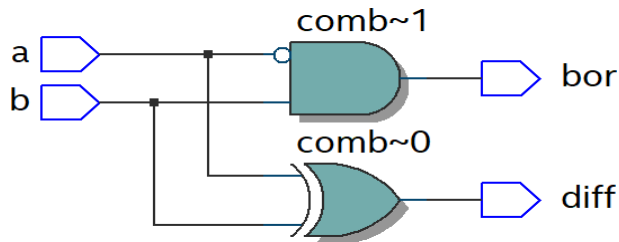


Figure 4. 10 Code & RTL output of Half Subtractor

### 4.5.4. Full-Subtractor

```

1  module full_subtractor (a,b,b_in,diff,b_out);
2  input a,b,b_in;
3  output diff,b_out;
4  wire f1,f2,f3,f4,f5;
5
6  xor (f1,a,b);
7  xor (diff,f1,b_in);
8  not (f2,a);
9  and (f3,f1,b);
10 not (f4,f1);
11 and (f5,f4,b_in);
12 or (b_out,f3,f5);
13
14 endmodule
15

```

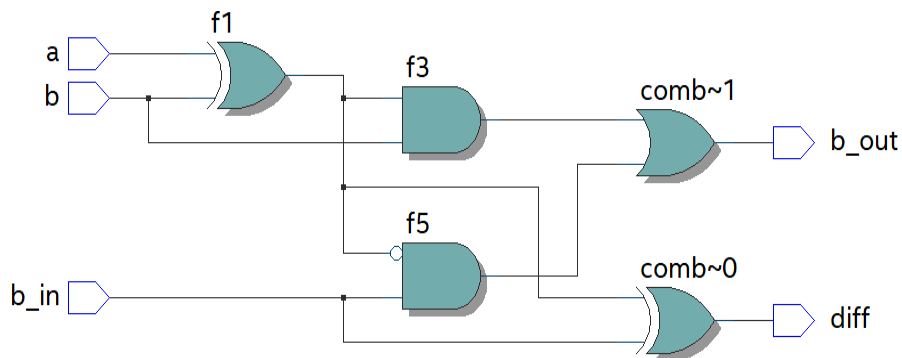


Figure 4. 11 Code & RTL output of Full Subtractor

## Chapter 6

### Conclusion

The culmination of the 14-day internship session, which encompassed a span of 10 productive workingdays, marked its conclusion on the 5th of April in the year 2023. Over the course of this dedicated internship period, a wealth of knowledge and insights was gleaned through a comprehensive visit to the company's premises. This immersive experience shed light on the intricate interplay between the theoretical concepts studied and their real-world application within the operational dynamics of the production line.

The inherent value of industrial training emerged as a significant contributor to the collective progress of various stakeholders, with far-reaching implications for the advancement of the nation as a whole. This symbiotic relationship between academic institutions and industries is a critical conduit for knowledge exchange and skills enhancement. As aspiring students engage in industrial training, they are bestowed with invaluable opportunities to not only acquire firsthand industrial experiences but also to immerse themselves in the authentic and often dynamic work environment present at the industrial training site.

Such exposure serves as a transformative platform for the cultivation of multidimensional skills. This amalgamation of theoretical grounding and practical application fosters the nurturing of innovative thinking, allowing students to channel their creativity into tangible solutions and real-world applications. Moreover, this immersive experience provides an opportune moment for students to internalize the ethical dimensions of their chosen profession, thereby ingraining professional values that will serve as a guiding compass as they navigate their future career trajectories.

In essence, the culminating industrial training opportunity provides a holistic learning experience that extends beyond the confines of traditional classroom education. It bridges the gap between academia and industry, nurturing not only adept professionals but also well-rounded individuals who are primed to contribute meaningfully to the ever-evolving landscape of their chosen field.

### 6.1. Learnings

#### 6.1.1. Orientation in RTL Engineering:

- Receive comprehensive training in the realm of RTL (Register Transfer Level) engineering.
- Acquire knowledge about the principles and techniques used in designing digital systems.

#### 6.1.2. Foundational Exploration of Verilog:

- Engage in an in-depth exploration of Verilog, a hardware description language.
- Understand the basics of how digital circuits can be described and simulated using Verilog.

### **6.1.3. Extensive Coverage of Coding Methodologies:**

- Learn diverse coding methodologies used in digital circuit design.
- Gain exposure to switch-level, gate-level, dataflow-level, and behavioral-level coding approaches.

### **6.1.4. Acquaintance with Coding Techniques:**

- Attain a comprehensive understanding of various coding techniques.
- Familiarize oneself with the intricacies of switch-level, gate-level, dataflow-level, and behavioral-level coding.

### **6.1.5. Implementation and Utilization of Test Benches:**

- Acquire practical knowledge of creating and using test benches in digital design.
- Learn how test benches are instrumental in verifying the correctness of digital designs.

### **6.1.6. Profound Insight into Test Bench Functionality:**

- Understand the importance of test benches in ensuring accurate design functionality.
- Recognize the role of test benches in identifying potential design flaws and issues.

By engaging in these activities during the internship, participants not only expand their theoretical knowledge but also develop practical skills essential for their future endeavors in the field of digital design and engineering.

## **6.2. Fields to Improve**

During the Industrial Training (IT) sessions, due attention should be directed by the company toward the concerns pertinent to students. Recognizing that students are at the nascent stages of their exposure to the pre-employment realm, the provision of comprehensive guidance and mentorship, fostered through keen observation by the firm, emerges as an essential factor. This approach ensures that students, as budding professionals, are provided with the necessary support to bridge the transition from academia to practical application seamlessly.

Moreover, the incorporation of a reward system within the company's framework is highly recommended. Acknowledging and appreciating the efforts and contributions made by students through tangible rewards not only bolsters their morale but also cultivates a sense of accomplishment. Such recognition, coupled with the effective guidance offered by the company, possesses the potential to ignite and sustain the enthusiasm of the students. This dual approach serves to invigorate their commitment and engagement, consequently aligning their aspirations with the opportunities within the field.

Ultimately, the symbiotic relationship established through prioritizing student concerns and recognizing their endeavors creates an ecosystem conducive to both personal and professional growth. This approach not only nurtures the talents of these burgeoning professionals but also elevates the company's stature as a facilitator of experiential learning and a platform for fostering future talent.

### 6.3. References

1. [THiNK – Your ideas our solutions \(thinkgroup.com.bd\)](http://thinkgroup.com.bd)
2. [ULKASEMI – Largest Semiconductor Company in Bangladesh – ULKASEMI](#)
3. [RTL Verilog - javatpoint](#)
4. [What is RTL Design in VLSI? - Maven Silicon \(maven-silicon.com\)](http://maven-silicon.com)
5. [Integrated Circuit - Definition, Construction, Features, Types, FAQs \(byjus.com\)](http://byjus.com)
6. [Integrated Circuits - SparkFun Learn](#)



