# DAY 31- 111 DAYS VERIFICATION CHALLENGE

Topic: Verilog basics

Skill: Verilog

DAY 31 CHALLENGE:

## 1. What is the difference between VHDL & Verilog?

**Origin and History:**

- **VHDL**: Developed in the early 1980s by the U.S. Department of Defense under the VHSIC (Very High-Speed Integrated Circuits) program.
- **Verilog**: Developed in the mid-1980s by Gateway Design Automation and later acquired by Cadence Design Systems.

**Syntax and Semantics:**

- **VHDL**: Has a more verbose and strongly-typed syntax, similar to Ada or Pascal. It requires explicit declarations for types and objects, making the code more descriptive and self-documenting.
- **Verilog**: Has a more concise and C-like syntax, which can be easier for those familiar with C programming. It is less strict about types, making it more flexible but sometimes less explicit.

**Use Cases:**

- **VHDL**: Often used in industries requiring high-reliability systems, such as aerospace, defense, and certain commercial applications. Its strong typing and verbosity make it suitable for complex designs that benefit from detailed documentation.
- **Verilog**: Widely used in commercial and consumer electronics industries. Its concise syntax and flexibility make it popular for rapid prototyping and commercial hardware design.

**Libraries and Packages:**

- **VHDL**: Supports packages and libraries extensively, allowing for reusable code and modular design. VHDL's strong typing system is advantageous when dealing with large, complex systems.

- **Verilog**: Supports modules and tasks, which are also conducive to modular design, but libraries and packages are less emphasized compared to VHDL.

**Simulation and Synthesis:**

- **VHDL**: Provides robust support for simulation with detailed timing and concurrent operations. Its synthesis support is equally strong but may require more explicit coding practices.
- **Verilog**: Equally strong in both simulation and synthesis. Verilog is often preferred for its simulation speed and synthesis capabilities in commercial EDA tools.

**Community and Tool Support:**

- **VHDL**: Has strong support in certain regions and industries, with extensive tool support from vendors like Mentor Graphics, Synopsys, and Xilinx.
- **Verilog**: Enjoys widespread industry adoption with extensive tool support from the same vendors, as well as additional support from open-source tools and a large community.

## 2. What are advantages of Verilog over VHDL?

- **Concise and Familiar Syntax:** Verilog's syntax is more concise and similar to the C programming language, which many engineers and programmers find easier to learn and use.
- **Ease of Use:** Verilog is generally considered easier to write and understand, especially for small to medium-sized designs. Its less verbose nature can make the code quicker to write and read.
- **Simulation Speed:** Verilog simulations are often faster, which can be beneficial during the iterative design and testing phases.
- **Widely Used in Industry:** Verilog is widely adopted in the commercial electronics industry, particularly in the United States. This widespread use means there is a large base of existing code, extensive support from EDA tools, and a strong community of engineers familiar with Verilog.
- **Support for System Verilog:** System Verilog is an extension of Verilog that adds many advanced features for design and verification. Engineers who start with Verilog can easily transition to System Verilog to leverage these additional capabilities.
- **Flexible Data Types:** Verilog’s flexible and less strict typing system can make certain types of design and verification tasks more straightforward and less cumbersome.

- **Better for Digital Design:** Verilog is often preferred for digital design because of its straightforward representation of hardware constructs like gates, flip-flops, and other digital elements.
- **Modularity:** Verilog's module-based design approach is intuitive and aligns well with hierarchical design methodologies, which is common in digital design.
- **Simplicity in Testbench Creation:** Creating testbenches for simulation is typically simpler in Verilog due to its straightforward syntax and constructs for modelling test scenarios.
- **Integration with Analog/Mixed-Signal Design:** Verilog-A and Verilog-AMS extend Verilog for analog and mixed-signal modelling, allowing for a unified approach in designs that combine digital and analog components.

## 3. Explain below methodologies for Digital Design:

### Top-Down Design Methodology

Starts from the system level and breaks down into components. Suitable for complex systems where high-level design is crucial.

1. **System Specification**: Define overall requirements and functionality.
2. **High-Level Design**: Create a block diagram of major components and interactions.
3. **Functional Decomposition**: Break down high-level blocks into smaller sub-blocks.
4. **Detailed Design**: Decompose sub-blocks into detailed modules/components.

### Bottom-Up Design Methodology

Starts from individual components and integrates them into the system. Useful when component reuse and individual module design are priorities.

1. **Component Design**: Design and verify individual components or modules.
2. **Module Integration**: Integrate components to form larger modules.
3. **System Integration**: Combine modules into the complete system.
4. **System Verification**: Test and verify the integrated system to ensure it meets specifications.

## 4. What is a module in Verilog? Write the basic syntax of declaring a module?

a module is the basic building block used to define a hardware component. It encapsulates a piece of hardware design, specifying its inputs, outputs, and internal functionality. Modules can be instantiated within other modules, allowing hierarchical design.

```
Module module_name(
        //port declaration (input or output)
);
//internal declaration (e.g., register or wire)
//behavioural or structural description of a module
endmodule
```

## 5. Explain the difference between module & module instance in Verilog with an example.

In Verilog, a **module** is a design entity that defines the hardware component's behaviour and structure. A **module instance** is a specific instantiation of a module within another module, allowing you to create multiple copies of a module with different parameters or connections.

**Module:-**

```
module AND_Gate (
        input wire A,
        input wire B,
        output wire Y
);
        assign Y = A & B;
endmodule
```

**Module instance :-**

```
module TopModule (
```

```
    input wire A,
    input wire B,
    input wire C,
    output wire Y1,
    output wire Y2
);
    AND_Gate and1 (.A(A), .B(B), .Y(Y1));  // module instance
    AND_Gate and2 (.A(A), .B(C), .Y(Y2));
endmodule
```

## 6. Describe below abstraction levels with an example:

## Behavioural

Specifies what the system does, without detailing how it is done. It's the highest level of abstraction, often used for writing testbenches or high-level design.

```
module Mux2to1 (
    input wire sel,
    input wire a,
    input wire b,
    output reg y
);
    always @(*) begin
        if (sel)
            y = b;
        else
            y = a;
    end
endmodule
```

## Data Flow

Describes how data flows through the system using continuous assignments. It focuses on the flow of data between registers and how outputs depend on inputs.

```
module Mux2to1 (
```

```
    input wire sel,
    input wire a,
    input wire b,
    output wire y
);
    assign y = sel ? b : a;
endmodule
```

## Gate Level

Describes the circuit in terms of logic gates and their interconnections. It is more detailed and closer to the actual hardware implementation.

```
module Mux2to1 (
    input wire sel,
    input wire a,
    input wire b,
    output wire y
);
    wire sel_n, a_and_sel_n, b_and_sel;

    not (sel_n, sel);
    and (a_and_sel_n, a, sel_n);
    and (b_and_sel, b, sel);
    or (y, a_and_sel_n, b_and_sel);
endmodule
```

## Switch Level

Describes the circuit at the transistor level using switches (MOSFETs). It is the lowest level of abstraction and directly represents the physical hardware.

```
module Mux2to1 (
    input wire sel,
    input wire a,
    input wire b,
```

```
    output wire y
);
    wire sel_n, y_a, y_b;

    // Complement of the select signal
    assign sel_n = ~sel;

    // Transistor-level implementation
    pmos (y_a, a, sel_n); // PMOS switch
    nmos (y_b, b, sel);   // NMOS switch
    assign y = y_a | y_b; // Output
endmodule
```

## 7. Explain following blocks:

## Stimulus block

The stimulus block, also known as the testbench, is used to generate the input signals to test the design under various conditions. It simulates the external environment interacting with the design and verifies if the design behaves as expected.

## Design block

The design block is the actual hardware component being designed and tested. It encapsulates the logic and functionality of the hardware module.