Verilog

A Hardware Description Language (HDL)

Vedant Pahariya

Contents

1 Introduction

Verilog is not a Programming Language, it is a hardware description language which is used to model electronic systems, such as digital circuits, memory, microprocessors, and network switches. HDLs are used to describe the hardware of digital systems in textual form.

1.1 Types of Description/Description styles in Verilog

- **Behavioral Description**: Describes the behavior of the system. (Highest level of description)
- **Dataflow Description**: Describes the flow of data in the system.
- **Structural/Gate level Description**: Describes the structure of the system.
- **Switch Level Description**: Describes the system at the transistor level. (Lowest level of description)

1.1.1 Difference between Gate-level and Structural Description

Reference: https://www.youtube.com/watch?v=ic1UMeuCBA8

- **Gate-level**: Only uses basic logic gates or primitives for modeling the system. It doesn't use any predefined/user-defined modules.
- **Structural**: Along with the gate-level description, it also uses predefined/ user-defined modules. It is a higher level of abstraction than gate-level modeling.

2 Basic Template of Verilog

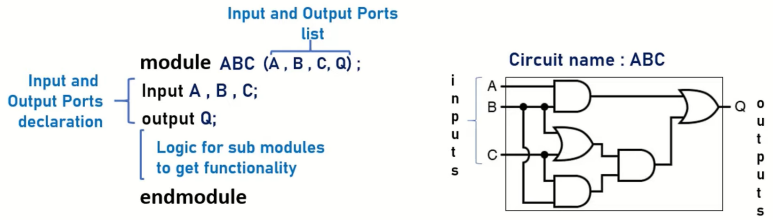
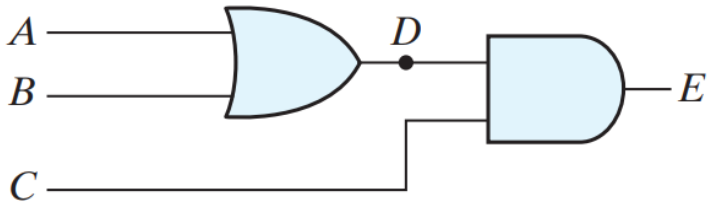


Figure 1: Boilerplate Code

Verilog is a Case Sensitive language.
The term "module" refers to the text enclosed by the keyword pair **module** ... **endmodule**. Module is the fundamental descriptive unit in Verilog language.
Keyword "module" is followed by the name of the design (ABC here) and parenthesis - enclosed list of ports.
Name of a design unit is an identifier (like variables in any programming language).
Logic is written by using "assign" keyword followed by the boolean equation.



Verilog Code for above circuit (AND and OR Gate)

```
module or_and(                  |   module or_and(A,B,C,E);
    input A,B,C                 |       input A,B,C;
    output E                    |       output E;
);                              |
    wire D;                     |       wire D;
    assign D = A || B;          |       assign D = A || B;
    assign E = C && D;          |       assign E = C && D;
endmodule                       |   endmodule
```

3 Data Types in Verilog

- **Wire**: Used to connect different components in the system.
- **Reg**: Used to store the value of a variable.
- **Parameter**: Used to store constant values.
- **Event**: Used to store event values.
- **Net**: Used to store net values.
- **Variable**: Used to store variable values.
- **Array**: Used to store array values.
- **Struct**: Used to store struct values.
- **Union**: Used to store union values.
- **Enum**: Used to store enum values.

4 Verilog Compilation in VS Code

Reference: GitHub Documentation
The command ' `iverilog -Wall -g2012 -o a.out design.sv testbench.sv` ' is used to compile Verilog and SystemVerilog code using Icarus Verilog, a popular open-source Verilog simulation and synthesis tool. Here's a breakdown of each part of the command:

- **iverilog**: This is the command to invoke the Icarus Verilog compiler.

- **-Wall**: This flag enables all the compiler's warning messages. It's helpful to catch potential issues in your code by making the compiler more verbose about any possible problems or best practice violations.
- **-g2012**: Specifies the IEEE 1800-2012 SystemVerilog standard.
- **design.sv**: The Verilog/SystemVerilog file containing the code for your hardware module.
- **testbench.sv**: The Verilog/SystemVerilog file containing the code for testing the hardware module.
- **-o a.out**: The -o option tells the compiler to place the compiled result into a file named a.out (Specifies the output file name).

Alternative Method:

- **Step 1**: Include the design.sv in the testbench.sv file using `‘include ”design.sv”`.
- **Step 2**: Run the command '`iverilog testbench.sv`'.
- **Step 3**: Run the command '`vvp a.out`' which will give the .vcd file as output for GTKWave simulation.

4.1 Difference between Verilog (.v) and SystemVerilog (.sv)

Refer: Reddit Post

- **Verilog**: Verilog is a hardware description language used to model electronic systems. It is the predecessor of SystemVerilog and is widely used in the industry.
- **SystemVerilog**: SystemVerilog is an extension of Verilog that adds new features for verification and design. It includes features like classes, randomization, and assertions that make it more suitable for verification tasks.

Verilog is subset of SystemVerilog. SystemVerilog is an extension of Verilog that includes additional features and capabilities, particularly for verification and design abstraction. Refer the above Reddit post for more details.

Think of SystemVerilog as C++ and Verilog as C. Everything you can write in C will work in C++, but C++ offers much more.

Vanilla Verilog is the pure, standard Verilog — without any SystemVerilog features. When someone says ”vanilla Verilog”, they usually mean ”just Verilog, no SystemVerilog”.

5 Verilog Testbench

5.1 Introduction

A testbench is a module that is used to test the functionality of another module. A testbench is written in the same way as any other Verilog module, but it is not synthesized into hardware. Instead, it is used to simulate the behavior of the design under different conditions.

5.2 Defining time precision

```
`timescale 1ns/1ps
```

The ‘timescale directive is used to define the time precision and time unit for the simulation. In this case, the time unit is 1ns (nanosecond) and the time precision is 1ps (picosecond). This means that the simulation will be done in nanoseconds, and the smallest time increment will be 1 picosecond.

```
#5; // 5 nanoseconds delay
```

Any number after ’#’ symbol is the delay in simulation time. In this case, the delay is 5 nanoseconds.

5.3 Reg and Wire Data Types

In Verilog, the reg and wire types are used for different purposes:

- **reg**: This type is used for variables that can hold a value and be assigned within procedural blocks such as initial and always. It is typically used for signals that are driven by procedural assignments.
- **wire**: This type is used for connecting different modules and for signals that are driven by continuous assignments (e.g., assign statements) or by the outputs of instantiated modules.

5.4 Instantiating the Module

In Verilog, instantiating a module means creating an instance of a module within another module. This is similar to creating an object from a class in object-oriented programming. There are two ways to do this:

- **Named Instantiation/ Named Port Mapping (Preffered)**: In this method, the instance name is explicitly specified in the instantiation statement. The syntax for instantiating a module is as follows:

```
and_gate and_gate_inst(
    .A(a),
    .B(b),
    .Y(y)
    );
```

  In this example, and_gate is the name of the module being instantiated, and and_gate_inst is the name of the instance. The .A, .B, and .Y are the

port connections between the module and the instance. The signals a, b, and y are the signals that are connected to the ports A, B, and Y of the and_gate module.

- **Ordered Instantiation/ Positional Port Mapping**: In this method, the instance name is not specified, and the ports are connected based on their order in the module definition.

```
and_gate and_gate_inst(a, b, y);
```

5.5 Simulation using GTKWave

```
$dumpfile("demo.vcd");
$dumpvars(0, tb_and_gate);
```

The above code generates a VCD file which can be viewed using GTKWave.

- **$dumpfile("demo.vcd")**: This command specifies the name of the VCD file that will be generated during the simulation. The waveform data will be stored in this file. The file extension .vcd stands for Value Change Dump, a standard format for waveform files.
- **$dumpvars(0, tb_and_gate)**: This command specifies the scope of the variables that will be dumped to the VCD file. The first argument to '$dumpvars' specifies the level of hierarchy to dump. A value of 0 means that only the variables in the specified module (tb_and_gate in this case) will be dumped. If you use a higher number, it will include variables in lower levels of hierarchy as well.

5.6 Multiple Initial Blocks

In Verilog, you can have multiple initial blocks in a testbench. Each initial blocks in a testbench run concurrently, not sequentially. Each initial block starts executing at time 0 and progresses independently of the others. This concurrency allows you to simulate different processes happening simultaneously in your testbench.
In the sample Testbench given below, there are two initial blocks. The first initial block generates the input signals A and B, while the second initial block monitors the output signal Y for different instances of time.

5.7 Monitor Changes

The monitor statement is used to display the values of signals at different instances of time during the simulation.

```
$monitor("Time = %0t: a = %b, b = %b, y = %b", $time, a, b, y);
```

Here is the breakdown of each part of the monitor statement:

- **$monitor**: This is the system task that sets up the monitoring. It will print the specified message whenever any of the variables in the list change.

- **"Time = %0t: a = %b, b = %b, y = %b"**: This is the format string that specifies how the values of the signals will be displayed. The %0t specifier is used to display the current simulation time, while the %b specifier is used to display the binary value of the signals a, b, and y.

  The 0 indicates that no minimum field width is specified. "minimum field width" refers to the minimum number of characters that should be used to display a value. If you had specified a minimum field width (e.g., %5t), the time would be padded with spaces to ensure it occupies at least 5 characters.

- **$time**: This system function returns the current simulation time in picoseconds.

- **a, b, y**: These are the signals whose values will be displayed by the monitor statement.

Sample Verilog Testbench Code

```
// Testbench for AND gate
`timescale 1ns/1ps
`include "design.v"
module tb_and_gate;

// Testbench signals
reg a;
reg b;
wire y;

// Instantiate the AND gate
and_gate uut (
    .a(a),
    .b(b),
    .y(y)
);

initial begin
    // Dump waves to VCD file
    $dumpfile("demo.vcd");
    $dumpvars(0, tb_and_gate);
    // 0 represents the hierarchy level of variables to be dumped
    // Initialize signals
    a = 0;
    b = 0;

    // Apply test vectors
    #10 a = 0; b = 0;
    #10 a = 0; b = 1;
    #10 a = 1; b = 0;
    #10 a = 1; b = 1;
    #10;

    // Finish simulation
    $finish;
```

```
    end

    // Monitor changes
    initial begin
        $monitor("Time = %0t: a = %b, b = %b, y = %b", $time, a, b, y);
    end

    endmodule
```

6 Blocking & Non-Blocking Assignments

In Verilog, there are two types of assignments: blocking and non-blocking.

- **Blocking assignments** (using the '=' operator) are executed sequentially. The next statement will not execute until the current statement is complete. This is similar to how assignments work in most programming languages.
- **Non-blocking assignments** (using the '¡=' operator) allow the next statement to execute without waiting for the current statement to complete. This is useful in sequential logic, where you want to model flip-flops and other memory elements.

6.1 Delay-Based Timining Control

In Verilog, we can use delay-based timing control to specify the time delay for the execution of a statement. This is done using the `'#'` symbol followed by the time delay value. This can be done in two ways:

- **Inter Assignment Delay:** This is used to specify the time delay for the execution of a statement. The statement will be executed after the specified time delay. The delay value is specified on the left hand side of the expression.
- **Intra Assignment Delay:** This is used to specify the time delay for the execution of a statement within a blocking assignment. The RHS to LHS assignment happens after the specified time delay. The delay value is specified on the right hand side of the assignment operator.

For both cases, the delay is specified in time units (e.g., nanoseconds, picoseconds, etc.) which is mentioned in the timescale directive at the top of the Verilog file.

Example-1:

Delay-Based Timing Control

```
    // Inter                    |        // Intra
 initial begin                 |        initial begin
    #10 b = 1;                 |            c = #10 a + b;
    #5 c = 5;                  |            d = #5 3;
end                            |        end
```

In the above example, in case of inter assignment delay, the value of `b` will be assigned after 10 time units, and the value of `c` will be assigned after 15 time units (10 + 5). In case of intra assignment delay, the value of `c` will be assigned after 10 time units but the value of `a + b` will be evaluated at time = 0 only. The value of `d` will be assigned after 15 time units (10 + 5).

Example-2:

Inter Assignment Delay in Non-Blocking Assignments

```
   // Case-1       |       // Case-2          |      // Case-3
initial begin     |     initial begin        |    initial begin
   a <= 0;        |        a <= 1;           |       a <= 1;
   b <= 1;        |        #5 b <= a + 1;    |       b <= #5 a + 1;
   c <= 2;        |        #10 c <= 3;       |       c <= #10 3;
   d <= 3;        |        #15 d <= 4;       |       d <= #15 4;
end               |     end                  |    end
```

Case-1: This is simple non-blocking assignment execution. All the assignments will be executed in parallel and the values of `a`, `b`, `c`, and `d` will be 0, 1, 2, and 3 respectively at time t = 0.

Case-2: Even though we are using the non-blocking assignments, this will be executed similar to blocking assignments because of inter assignment delays. The assignments to `b`, `c`, and `d` have delays specified. The value of `b` will be assigned after 5 time units, `c` after 15 time units, and `d` after 30 (5+10+15) time units. The values of `a`, `b`, `c`, and `d` will still be 1, 2, 3, and 4 respectively after the execution of the initial block.

Case-3: In this case, the assignments to `b`, `c`, and `d` have delays specified. The value of `b` will be assigned after 5 time units, `c` after 10 time units, and `d` after 15 time units. However, the values of `a`, `b`, `c`, and `d` will be 1, 1, 3, and 4 respectively. Here, the value of `b` is different from Case-2 assigned after 5 time units, but the value of `a + 1` is evaluated at time t = 0 only. So, at time t=0, the value of `a` = 0, so the value of `b` will become 1.

Example-3:

Intra Assignment Delay in Non-Blocking Assignments

```
initial begin
   a <= 0;
   b <= #5 1;
   c = #10 2;
   d <= #3 3;
   e <= #5 4;
end
```

Here, the value of `a`, `b`, `c`, will be assigned after 0, 5, 10 time units respectively. The c statement will block the execution of further statements until it is executed completely. So, the value of `d` and `e` will be assigned after 13 time units (10 + 3) and 15 time units (10 + 5) respectively.

7 Sample Code of 2:1 MUX in all Modeling Styles

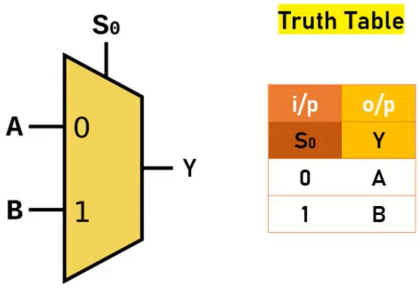


Figure 2: 2:1 MUX

7.1 Dataflow Modeling

Listing 1: Dataflow Modeling Verilog Code for MUX

```
module mux_2_1(A,B,S,Y);
    input A,B,S;
    output Y;
    assign Y = (~S & A) | (S & B);
endmodule
```

7.2 Gate-level Modeling

Listing 2: Gatelevel Modeling Verilog Code for MUX

```
module mux_2_1(A,B,S,Y);
    input A,B,S;
    output Y;
    wire w1,w2,w3;
    not no(w1,S); // here no is the instance name
    and ao(w2,A,w1);
    and a1(w3,B,S);
    or o1(Y,w2,w3);
endmodule
```

7.3 Behavioral Modeling

Listing 3: Behavioral Modeling Verilog Code for MUX

```
module mux_2_1(A,B,S,Y);
    input A,B,S;
    output Y;
    reg Y;
    always @(A,B,S)
    begin
```

```
7           if(S==0)
8               Y = A;
9           else
10              Y = B;
11      end
12  endmodule
```