RAMAIAH
SKILL ACADEMY

Verilog Programming

Prepared By
Darshan M R
Application Engineer
Ramaiah Skill Academy

contents

Introduction to Verilog HDL

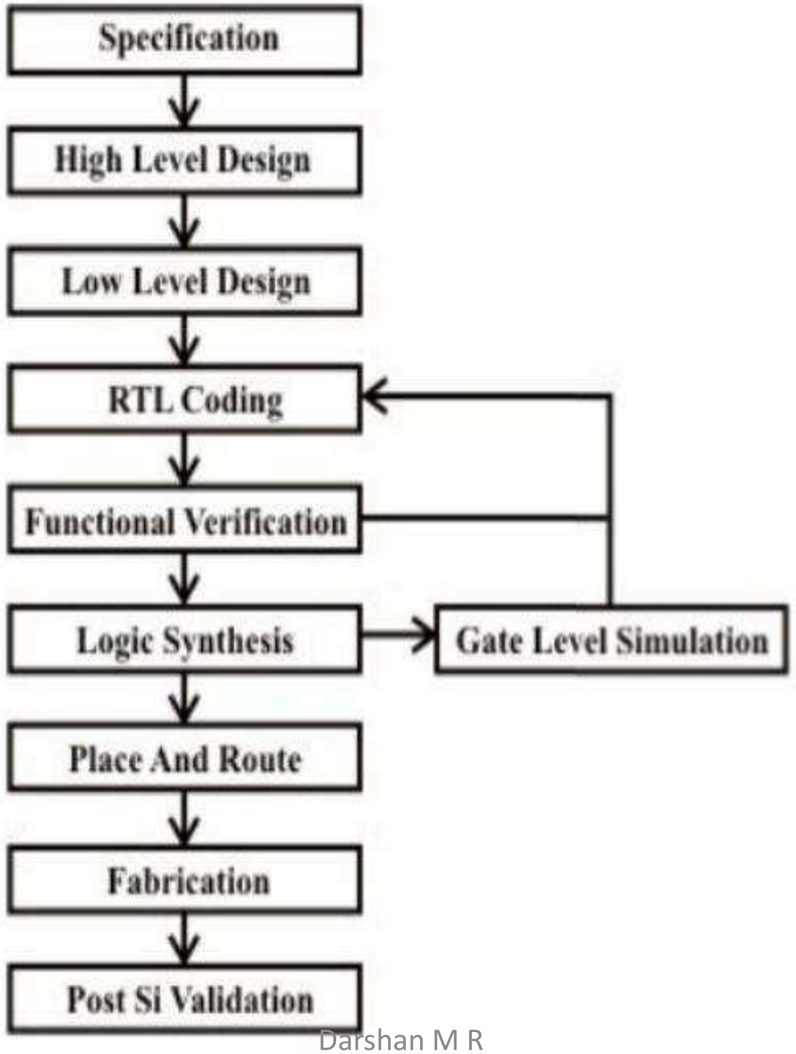
- **What is Verilog?**
  An HDL for designing digital systems.
  IEEE 1364 Standardized Language.

- **Design Flow in Verilog:**
  Coding → Simulation → Synthesis → Implementation.

History:

- Need: a simple, intuitive and effective way of describing digital circuits for modeling, simulation and analysis.
- Developed in 1984-85 by Philip Moorby
- In 1990 Cadence opened the language to the public
- Standardization of language by IEEE in 1995

Design flow


Specification
High Level Design
Low Level Design
RTL Coding
Functional Verification
Logic Synthesis
Gate Level Simulation
Place And Route
Fabrication
Post Si Validation

Definition of Module

- Interface: port and parameter declaration
- Body: Internal part of module
- Add-ons (optional)

```
module(port list);
    //port declaration;
        //design body
   //Add-ons
  endmodule;
```

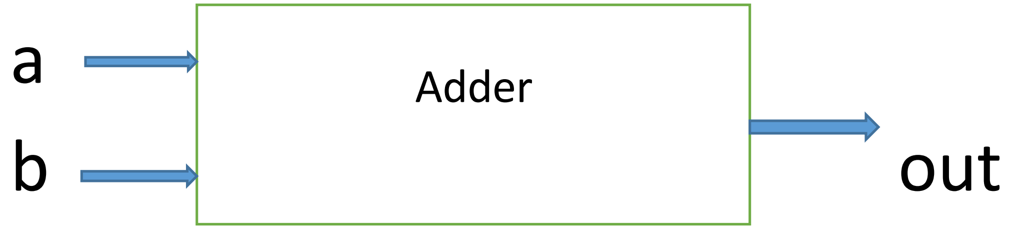
Commenting in verilog

- The name of Module

- Comments in Verilog
  - One line comment (// ………….)
  - Block Comment (/*…………….*/)

- Description of Module (optional but suggested)

The Module Interface

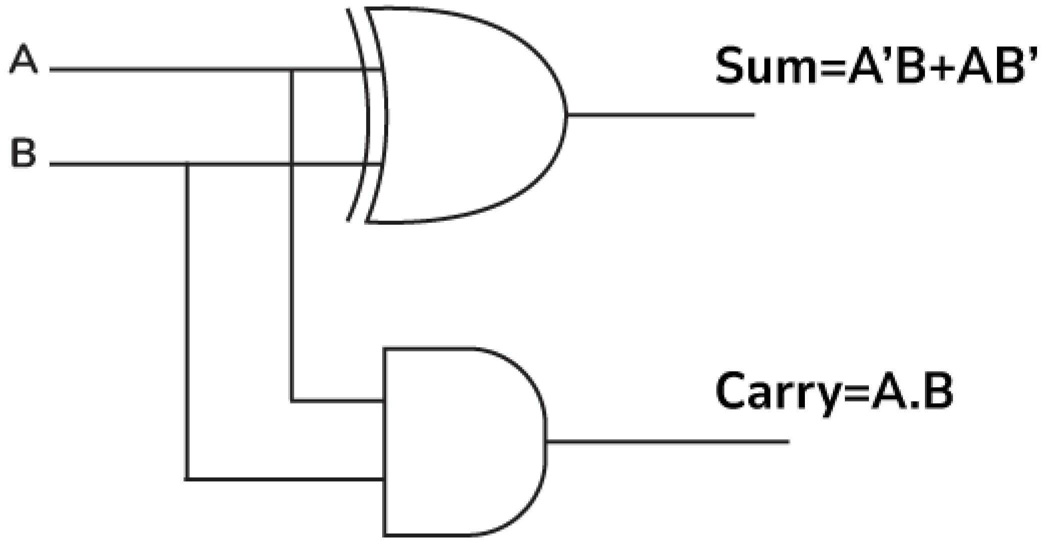


```
module adder(a,b,out);//port list
   input a,b;  // port declaration
   assign out=a+b;   //body
endmodule
```

Different modelling styles

- Structural level model
- Gate level
- Data flow level model
- Behavioral level model
- Switch level

Half adder



Gate level model

- module HalfAdder_GateLevel (input A, // 1-bit input A

```
                              input B, // 1-bit input B
                              output SUM, // 1-bit sum output
                              output CARRY // 1-bit carry output );
                         xor (SUM, A, B); // XOR gate for sum
                         and (CARRY, A, B); // AND gate for carry
  endmodule
```

Data flow model

- module HalfAdder_Dataflow (input A, // 1-bit input A

```
module HalfAdder_Dataflow ( input A, // 1-bit input A
                            input B, // 1-bit input B
                            output SUM, // 1-bit sum output
                            output CARRY // 1-bit carry output );
          assign SUM = A ^ B; // XOR for sum
          assign CARRY = A & B; // AND for carry
endmodule
```

Behavioral model

- module HalfAdder_Behavioral (input A, // 1-bit input A

```
                              input B, // 1-bit input B
                              output reg SUM, // 1-bit sum output
                              output reg CARRY // 1-bit carry output );
          always @(*) begin
                SUM = A ^ B; // XOR for sum
                CARRY = A & B; // AND for carry
            end
endmodule
```

Data Types, Constants, Parameters

Data types:

- Nets: physical connection between hardware elements

  -> wire

- Registers: Store value even if disconnected

  -> reg

Constants: signed and unsigned

30 , -2

2’b10 , 6’d-4

Parameter:

parameter red=1;

Data types

- Real :

  delta=4e10;   delta=2.34;

- Vector:

  wire [0:10]bus;   reg [0:10]buffer;

- Integer:

  integer i=10;

- Time :

  $time;

Array

One dimensional array

```
reg   data[0:10];
reg [7:0]data [0:10];
```

Multi dimensional array

```
reg data[0:10][0:10];
```

Memory

```
reg  mem1bit[0:1023];    //1kb   with 1bit
reg  [7:0] mem8bit[0:1023];   //1k with 8bit
```

Assignment

- Continuous Assignment

```
module continuous(input statin, output statout);
        assign statout= ~statin;
 endmodule
```

- Procedural Assignment

I. Blocking

II. Non-Blocking

Blocking Assignment

Symbol: =

```
module blocking(input [0:2]preset ,
                      output reg [3:0] count)
      always @(preset)
            count=preset+1;   //blocking procedural Assignment .
 endmodule
```

Non-Blocking Assignment

- Symbol: <=

```
module Nonblocking( input [3:0] regA,Mask,
                    output reg [3:0]regB);
          always @( regA or Mask)
                regB<=regA & Mask;  //non-blocking assignment
endmodule
```

Operations

- Arithmetic operations
- Logical operations
- Relational operations
- Equality operations
- Shift operations
- Vector operstion
- Part selector
- Bit selector
- Conditional expression

IF Statement

```
Module selectone(input a,b
                        output z);
        always @( a or b)
             if(a>b)
                 z=a;
             else
                  z=b;
         end
```

Case Statement

```
module alu(input a,b,
                      input [1:0]sel,
                     output out);
   always @(a or b or sel)
           begin
             case(sel)
                     00:out=a+b;
                     01:out=a-b;
                     10:out=a&b;
                     11:out=a|b;
                endcse
             end
endmodule
```

Case Statement

- **Casex**

```
module casex_example;
   reg [3:0] input_val;
   reg output_val;

   always @(*) begin
      casex (input_val)
         4'b1xx0: output_val = 1'b1; // Matches any value where MSB is 1 and LSB is 0
         4'b0x1x: output_val = 1'b0; // Matches any value where MSB is 0 and bit 2 is 1
         default: output_val = 1'bx; // For all other cases
      endcase
   end
endmodule
```

Case Statement

- **Casez**

```
module casez_example;
  reg [3:0] input_val;
  reg output_val;

  always @(*) begin
    casez (input_val)
      4'b1z10: output_val = 1'b1; // Matches any value with MSB=1, second bit high-impedance (z)
      4'b0z1z: output_val = 1'b0; // Matches any value with MSB=0, second and fourth bit high-impedance
      default: output_val = 1'bx; // For all other cases
    endcase
  end
endmodule
```

Case statement

```
module two_variable_case_example;
  reg [1:0] var1, var2; // Two variables
  reg [3:0] combined_var; // Combined variable for case
  reg output_val;

  always @(*) begin
    combined_var = {var1, var2}; // Concatenate var1 and var2
    case (combined_var)
      4'b0000: output_val = 1'b0; // var1=00, var2=00
      4'b0001: output_val = 1'b1; // var1=00, var2=01
      4'b0010: output_val = 1'b0; // var1=00, var2=10
      4'b0011: output_val = 1'b1; // var1=00, var2=11
      default: output_val = 1'bx; // Unmatched values
    endcase
  end
endmodule
```

Module instance & Instantiation

```
module TopModule;
   reg [3:0] A, B;   // 4-bit inputs
   reg Cin;          // Carry input
   wire [3:0] Sum;   // 4-bit Sum output
   wire Cout;        // Carry output

   // Instantiate the Adder4Bit module
   Adder4Bit myAdder (
      .A(A),        // Connect A
      .B(B),        // Connect B
      .Cin(Cin),    // Connect Cin
      .Sum(Sum),    // Connect Sum
      .Cout(Cout)   // Connect Cout
   );
endmodule
```

Module instance & Instantiation

```
module Adder4Bit(
   input [3:0] A,    // 4-bit input A
   input [3:0] B,    // 4-bit input B
   input Cin,        // Carry input
   output [3:0] Sum, // 4-bit Sum output
   output Cout       // Carry output
);
   assign {Cout, Sum} = A + B + Cin; // Addition with carry
endmodule
```

Tasks and Functions

- **Functions in Verilog**

  A function that does not consume simulation time, returns a single value or an expression, and may or may not take arguments.

- **Keywords used:** function and endfunction.

Function

- //Method 1

```
function <ret_type> <func_name> (input   <port_list>, inout    <port_list>,  output <port_list>);
          ...
        return <val or expression>
  endfunction
```

- // Method 2

```
function <return_type> <func_name> ();
         input <port_list>;
         inout <port_list>;
         output <port_list>;
              ...
           return <val or exp>
 endfunction
```

Function Example

```
module func;
    function compare(input int val1,val2);
        if(val1>val2)
                $display(“val1 is greater than val2”);
        else if(val1<val2)
                $display(“val1 is less than val2”);
        else
                $display(“val1 is equal to val2”);
        return 1;  //not mandatory
endfunction
```

```
Initial begin
   compare(10,10);
   compare(5,9);
   compare(9,5);
end
endmodule
```

Task

```
//method 1
  task <task_name>(input <port_list>,output <port_list>);
        …….
   endtask
//method 2
   task <task_name>();
         input <port_list>;
         output <port_list>;
       ……………………
    endtask
```

Task Example

```
module task_ex;
    task compare(input int val1,val2, output done);
       if(val1>val2)
              $display("val1 is greater than val2");
       else if(val1<val2)
              $display("val1 is less than val2");
       else
             $display("val1 is equal to val2");
       #20;
       done = 1;  //not mandatory
endtask
```

```
Initial begin
  bit done;
  compare(10,10,done);
  if(done)$display("comparison dine at
time =%0t",$time);
  compare(5,9);
 if(done)$display("comparison dine at
time =%0t",$time);
  compare(9,5);
 if(done)$display("comparison dine at
time =%0t",$time);
end
endmodule
```

Difference between Task & Function

Feature	Task	Function
Declaration	task, endtask	function, endfunction
Arguments	input, output, inout	input only
Timing Controls	Allowed (#, @, wait)	Not Allowed
Return Value	output, inout	Directly returns a value
Call Syntax	Procedural (task_name)	Expression (function_name)
Usage Context	Procedural only	Procedural and continuous

Reference

1. Verilog HDL Synthesis A practical primer by J Bhasker -BS Publications