VLSI DESIGN FLOW: RTL TO GDS

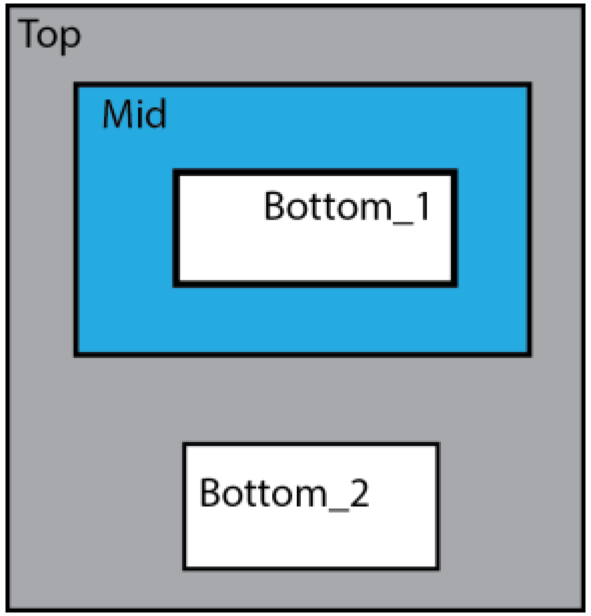
Lecture 10
Hardware Modeling: Introduction to Verilog-II

Sneh Saurabh
Electronics and Communications Engineering
IIIT Delhi

NPTEL

Modules

- Modules are building blocks of a Verilog design: starts with a keyword **module** and ends with **endmodule**
- Modules are instantiated inside another modules to create a design hierarchy
  - **Instantiation** of a module means using that module in another higher-level module



```
module Top;
      Mid m1;
      Bottom_2 b2;
endmodule
```

```
module Mid;
      Bottom_1 b1;
endmodule
```

```
module Bottom_1;
endmodule

module Bottom_2;
endmodule
```

S. Saurabh, “Introduction to VLSI Design Flow”. *Cambridge University Press*, 2023.

Ports

- Ports allow communication between the module and the environment
- Ports can be declared with keywords **input**, **output** or **inout**
- Can be scalar or vector (similar to net and variables)

```
module top(clock, reset, test_port, data, counter_out);

        input clock;
        input reset;
        inout test_port;
        input [3:0]data;
        output [3:0]counter_out;

endmodule
```

S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

NPTEL

Instances and Connections

Connections for an instance can be specified in two ways:

1. By order (implicit connection)
   - Ports connected automatically in the instantiation based on order specified in the module declaration
   - Difficult to debug
2. By name (explicit connection)
   - Ports connected by explicitly giving names
   - Order not important

```
module mid(clock, d_in, d_out);
        input clock;
        input d_in;
        output d_out;
endmodule
```

```
module top(c, p_in, p_out);

      mid m1(c, p_in, p_out);

endmodule
```

```
module top(c, p_in, p_out);

      mid m1(.clock(c), .d_in(p_in), .d_out(p_out));

endmodule
```

S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

Parameterized Module

- Verilog allows definition of constants for a module using the keyword **parameter**.
  - ➢ The module in which a parameter is defined is referred to as a parameterized module.

- Declare and provide a default value for the parameter in the module.
  - ➢ Can be overridden during instantiation by providing non-default values within #().

```
module counter(clk, rst, en, count);
        parameter WIDTH=4;
        input clk, rst, en;
        output [WIDTH-1:0] count;
        reg [WIDTH-1:0] count;
        ...
endmodule
```

```
module top();
        ...
        counter #(.WIDTH(16)) C1(clk, rst, en, count1);
        counter #(128) C2(clk, rst, en, count2);
        counter C3(clk, rst, en, count3);
        ...
endmodule
```

S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

NPTEL

Built-in Gates

- Verilog primitive gates available for modelling logic gates
  - Using keyword **and, nand, or, nor, xor, xnor**
  - First pin is output pin and rest are inputs

```
module mygates(a, b, en, y1, y2);

    input a, b, en;
    output y1, y2;

    and a1(y1, a, b);
    and a2(y2, a, b, en);

endmodule
```

S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

NPTEL

Operators and Expressions

Operators:

- Verilog supports operators with symbols similar to C programming language
  - ➢ Arithmetic operators: +, -, * / etc.
  - ➢ Logical: operators !, &&, || etc.
  - ➢ Bitwise operators: ~, !, |, ^, ~^ etc.
  - ➢ Relational: <, >, == etc.
  - ➢ Shift operators: >> << etc.

- Operators act on operands to produce results
- Operands can be of various data types such as net, variable, bit-select, part-select, or array element.
- Number of operands for an operator is defined by the language
  - ➢ Example: One: !, Two: <, Three: (c ? a : b)

Expression

- Expressions can be formed by combining operators with operands that yield some result
- Paratheses can be used for specifying precedence

!(a&b);

p<q

*(x+y)*z*

S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

NPTEL

Special Operators

- &, ~&, |, ~|, ^, ~^: **Reduction operator** perform a bit-wise operation on a single operand to produce a single bit result.
  - ➢ The computation starts from the LSB and moves toward the MSB

- Assume that A={1011}
- Then, Y = &A = 0

- {}: **Concatenation** operator, combines (concatenates) the bits of two or more data objects

- Assume that:
  a =2'b00,
  b =4'b1111,
  and c =1'b0.
- Then, {a, b, c} is 7'b0011110.

- {N{}}: **Replication** operator (Multiple concatenations performed N times)

- Assume that:
  a =2'b00,
  b =4'b1111,
  and c =1'b0.
- Then, {4{a}, b, 2{c}} is 14'b00000000111100

NPTEL

Block of statement

- A group of statements within the keywords **begin** and **end** forms a block
- A name can be given to a block

```
begin : my_block_name
        statement-1
        statement-2
        statement-3
end
```

- Statement executed sequentially within begin-end block
  - ➢ Known as sequential block

```
fork : my_block_name
        statement-1
        statement-2
        statement-3
join
```

- Statement executed concurrently within fork-join block
  - ➢ Known as parallel block

S. Saurabh, “Introduction to VLSI Design Flow”. *Cambridge University Press*, 2023.

NPTEL

Control Statements

- Verilog supports control statements such as **if-else**, **case**, **for, while**, **repeat** similar to traditional programming languages such as C

```
if (sel == 1'b0) begin
            y = a;
end
else begin
            y = b;
end
```

```
input [1:0]sel;
case (sel)
            0: y = a;
            1: y =b;
            2: y = c;
            default: y = d;
endcase
```

- **casez**: treats z as don't care
- **casex**: treats x and z as don't care

```
for (i = 1; i < 8; i = i + 1 )
begin
      state[i]  = 1'b1;
      y[i]= c[i];
end
```

```
while (i < 8) begin
      y[i]= c[i];
      i = i + 1;
end
```

```
repeat (8) begin
      y[i]= c[i];
      i = i + 1;
end
```

S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

Structured Procedures

Verilog supports structured procedures using four constructs:

- Initial Block and Always block
- Function and Task.

Initial and Always Block

- Initial block: block of code starting with the keyword **initial**
- Always block: block of code starting with the keyword **always**

- Both types of blocks are enabled when the simulation begins (time is 0).
- Initial block executes only once and stops when it reaches the end of the block
- Always block executes repeatedly throughout the simulation.

```
module myTop(datain, dataout);
        input datain;
        output dataout;

        reg clock, counter, dataout;

        initial begin
                clock = 1'b0;
                counter = 1'b0;
        end

        always begin
                #10 clock = ˜clock;
        end
endmodule
```

S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

Always block: Event Control

- Controlling event can be a value change on a net or a variable (level sensitive)

```
always @(en) begin
        rega = regb;
end
```

S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

- Can also define event based on the direction of change of a net or variable (edge sensitive).

Two types edge sensitivity:

- **posedge** : towards 1
  - ➢ Transition from 0, x, and z to 1
  - ➢ Transition from 0 to z or x.
- **negedge** : towards 0
  - ➢ Transition from 1, x, and z to 0
  - ➢ Transition from 1 to z or x.

```
always @(posedge  clk)  begin
        q=d;
end
```

Always block: Sensitivity List

- Can specify the logical OR of multiple events such that any one of the events can trigger the associated block or statement.
- The keyword **or**/comma (,) is used to specify logical OR of event.
- The or-separated list of triggering events is called the **sensitivity list**

```
always @(posedge  clk or negedge rst)  begin
        q=d;
end
```

- Sometimes we need to put all signals read in an always block in the sensitivity list
  - ➢ While modelling combinational logic block
- Can use @*

```
always @* begin
        q=((a&b|c)^d)|e;
end
```

S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

NPTEL

Usage of Initial and Always Block

- Initial blocks used in test benches to initialize our design for simulation.
- Always blocks are essential element for RTL and behavioral modeling.

- A module can have any number of Initial and Always blocks.
- All these blocks get executed at the beginning of the simulation time.
    - No predefined order in starting the execution of these Initial and Always blocks.
    - Initial blocks need not get executed before Always blocks

- Initial and Always blocks provide a mechanism to model the concurrency of the hardware.
    - Different hardware components working in parallel and whose order of starting the execution is not within our control
        - a simulation tool is free to choose any arbitrary order

NPTEL

Functions and Tasks (1)

- Functions and tasks can be used to model repeated code
- Keywords are: **function**, **endfunction**, **task**, **endtask**

	Functions	**Tasks**
Input / Output	Can have one or multiple inputs, but only one output	Can have zero or multiple inputs, outputs, inouts
Timing Delay	Cannot have delays modelled by **posedge, negedge, #**	Can have delays modelled by **posedge, negedge, #**
Model	Only combinational circuit	Both combinational and sequential circuits
Call	Can call another function but cannot call other task	Can call other task or function

Functions and Tasks (2)

```
module top(a, b, c, d, out1, out2);
        input a, b, c, d;
        output out1, out2;

        function myfunc;
                input x, y, z;
                begin
                        myfunc = x-y+z;
                end
        endfunction;

        assign out1 = myfunc(a, b, c);
        assign out2 = myfunc(b, c, d);

endmodule
```

```
module top(a, b, c, s1, c1, s2, c2);
        input a, b, c;
        output s1, c1, s2, c2;

        task mytask;
                input x, y;
                output sum, carry;
                begin
                        sum = x ŷ;
                        carry = x & y;
                end
        endtask;

        mytask(a, b, s1, c1);
        mytask(b, c, s2, c2);

endmodule
```

S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

Assignments

Two types of assignments in Verilog:

- Continuous assignment
- Procedural assignment

Continuous assignment:

- Provide values to the nets
- Using keyword **assign**
- Models combinational circuit

```
module mymux(a, b, s, out);
        input a, b, s;
        output out;

        assign out = (s) ? a : b;

endmodule
```

Procedural assignment:

- Provides values to the variables (such as **reg** or **integer** types)
  - ➢ Do not change their values until the next procedural assignment
- Used inside structured procedures (initial block, always block, functions, and tasks)
- Two types:
  - ➢ Blocking assignment (=)
  - ➢ Non-blocking assignment (<=)

```
always @(posedge  clk)
begin
        q = d;
end
```

```
always @(posedge  clk)
begin
        q <= d;
end
```

S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

Blocking and Non-blocking Assignments

	Blocking Assignment	Non-blocking Assignment
Symbol	=	<=
Execution	Execution of next statement blocked until the current statement is executed	Execution of next statement not blocked until the current statement is executed
Multiple statement	Executes sequentially	Executes parallelly

```
module top( );
     reg a, b, c, p, q, r;

     initial begin
          a = #10 1'b1; //at time = 10
          b = #30 1'b1; //at time = 40
          c = #20 1'b1; //at time = 60
     end

     initial begin
          p <= #10 1'b1; //at time = 10
          q <= #30 1'b1; //at time = 30
          r <= #20 1'b1; //at time = 20
     end

endmodule
```

S. Saurabh, "Introduction to VLSI Design Flow". *Cambridge University Press*, 2023.

System Tasks and Functions

- Verilog supports some in-built system tasks and system functions that help in debugging and verification

- Names start with $

- Examples: **$display, $probe, $monitor, $stop, $finish, $reset, $random, $time** etc..

NPTEL

References

- S. Saurabh, “Introduction to VLSI Design Flow”. Cambridge: *Cambridge University Press*, 2023.
- R. Seisyan. S. Palnitkar, “Verilog HDL: a guide to digital design and synthesis”, *Pearson Education India*, 2003
- “IEEE standard Verilog hardware description language.” IEEE Std 1364-2001 (2001), pp. 1–792.

soprotection.com