Slide 6.

2021年10月19日 22:01

Verilog: Not a programming language, but a hardware description language (HDL) 硬件描述语言.

specify hardware in two ways { Structurally: what hardware looks like.
Behaviourally: what hardware does { why? { eliminate misunderstanding; nothing haven't thought about }
behavioural: Functional units. (small pieces) what each block does, then connect
why? allow forming our work to individuals. experiment with different methods

register transfer level.

RTL-Level design.

for each function unit, what hardware looks like.
why? ① accurately estimate how fast/big of chip.

Gate-Level design how blocks should be constructed.

Why use Verilog?

can be used as an input to { synthesis: implement hardware.
simulation: what hardware will do before building

Verilog Modules NOT functions. like a chip with wires connected (I&O).

basic construct : module each hardware in one module

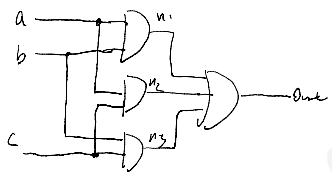
It has { declarations
input/output declaration.
Internal signal declaration.
Logic definition. { assign
case
module instantiation 模块实例化 }

Boolean Expression in Verilog.

Verilog use & , | , and ~ ^
and or not - XOR.

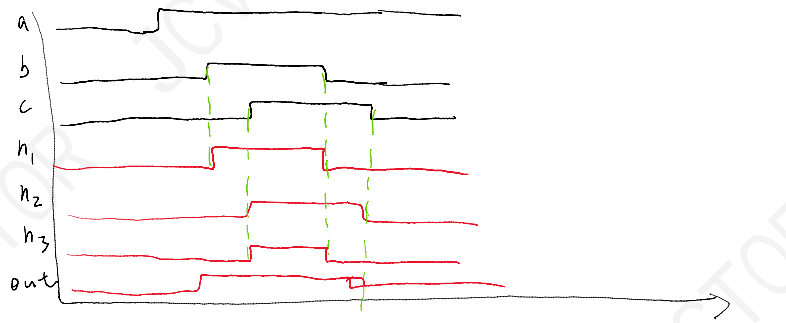
e.g. module declare ↗ I/O list ↙

```
module AND_GATE (a, b, c);
    input a, b;   ← Declare I/Os
    output c;
    assign c = a & b;   ← assign statement
endmodule
```




```
module Majority (a, b, c, out);
    input a, b, c;
    wire out;
    wire n1 = a & b;
    wire n2 = a & c;
    wire n3 = b & c;
    assign out = n1 | n2 | n3;
endmodule
```

or `assign out = (a&b)|(a&c)|(b&c);`



Verilog Literal Numbers 常量

number: <size>' [<signed>] <radix> value.

size: number of bits
radix: b — binary; d; o; h
value: digits.

eg. 8'b0001101
6'd29

e.g. 8'b0001[10]
8'd29

d
o
h

Buses:

wire [2:0] X.

3 element with indices 2 to 0.

[0:7] 8 element 0 to 7.

↓ bus name.

3 parallel wires, each element is single bit

↓ for i or output

① assign X = 8'b11100011;

② assign out_bus = in_bus;

③ access individual.

wire STATUS;
wire [15:0] Main_bus;
assign STATUS = Main_bus[15];

④

wire [31:0] Main_bus;
wire [3:0] Opcode;
assign Opcode = Main_bus[31:28]

{ wire [7:0] out_bus
  wire [15:8] in_bus
  assign out_bus = in_bus.

right to left

out_bus[0] = in_bus[8]
[1] [9]

{ wire[7:0] out_bus;
  wire[15:0] in_bus;
  assign out_bus = in_bus;

out_bus [0] = in_bus [0].

Verilog doesn't consider an error if use a signal without declaring. [7] = [7]
it will think it is a 1-bit wire

⑤

wire [3:0] Error_Code;
wire [7:0] Main_Bus;
assign Main_Bus = {4'b0000, 4'b1111};
... = {4'b0000, Error_Code};

⑥

wire [1:0]x = 2'b10;
wire [7:0]y = {4{x}}; y = 8'b10101010.

⑦ 2-D.

reg [3:0] data [7:0]. 8 elements 4 bit each

wire

e.g. data[7] = 4'b0011;

⑧ bitwise Boolean expression

each bit ↓ wire [3:0] result = A & B.

A[0] ┐
B[0] ┘— result[0]

//result[0] = A[0] & B[0];
result[1] = A[1] & B[1];

⑨

wire result = |A; result = (A[0] | A[1] | ...)

wire result = &A;

A — 4 ┐
B — 4 ┘— 4 — result.

A[0] ┐
A[1] ┘— result.

e.g. For thermostat

wire result = &A;

B 4

e.g. For thermostat


```
module Thermostat (presetTemp, currentTemp, fanOn);
    input [2:0] presetTemp, currentTemp; // 3 bits each
    output fanOn;
    assign fanOn = (currentTemp > presetTemp);
endmodule
```

Define.

`define Sa 2'b00 no ';'

mean Sa has value 2'b00. to use. `Sa

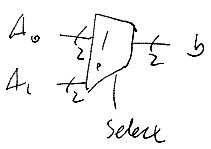
Conditional Operator

<cond_expr> ? <true_expr> : <false_expr>

if <cond_expr> is true, then value of entire expression is the value given by <true_expr>.

e.g. if in is 1, (in ? `Sb : `Sa) is `Sb

verilog of multiplexer:



combinational logic

wire [1:0]b = select ? A0 : A1

Always Block.

describe behaviour

always @ (<sensitivity_list>) begin

<sequence --- order matters>

end.

each always block describes one block of hardware

one hardware, one always.

when sensitivity_list changed, statements inside is evaluated.

↳ list of signals, separated by , or "or" or *

NOT input parameters.

NOT Pass value.

determines when the statements will be evaluated

In always block, no loops.

if, case, casex

use begin and end to group statements

signals, declared as reg instead of "wire"

No using assign statement.

block outputs using "=" or "<="

if (<condition expression>)

<statement>

;.

<statement>

[else <statement>]

case (<selector>)
{<label list>: <statement>}
[default: <statement>] → optional.
endcase

e.g. Days in Month Function

```
module DaysInMonth (month, days);
    input [3:0] month;
    output [4:0] days; // output reg [4:0] days;
    reg [4:0] days;
    always @(month) begin   // when month change, output changes
        case (month)
            2: days = 5'd28;
            4, 6, 9, 11: days = 5'd30;
            default: days = 5'd31;
        endcase
    end
endmodule
```

e.g. Writing truth table in Verilog.

```
module Prime (in, isprime);
    input [3:0] in;
    output reg isprime;
    always @(in) begin
        case (in)
            1, 2, 3, 5, 7, 11, 13: isprime = 1'b1;
            default:               isprime = 1'b0;
        endcase
    end
endmodule
```

0	0000	0
1	0001	1
2	0010	1
3	0011	1
4	0100	0
5	0101	1
6	0110	0
7	0111	1
8	1000	0
9	1001	0
10	1010	0
11	1011	1
12	1100	0
13	1101	1
14	1110	0
15	1111	0

always @(*)
↳ all signals in always block.

Combinational Logic Building

Verilog for 2:4 decoder.

```
module Dec24 (a, b);
    input [1:0] a;
    output [3:0] b;
```

```
wire [3:0] b = 1 << a ;
endmodule
```

`X << Y` means shift X to left by Y bit positions

Encoder:

a_3 a_2	a_1 a_0	b_1 b_0
0 0	0 1	0 0
0 0	1 0	0 1
0 1	0 0	1 0
1 0	0 0	1 1

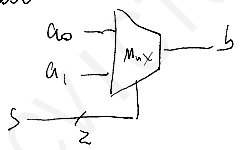
$b_0 = a_3 \vee a_1$

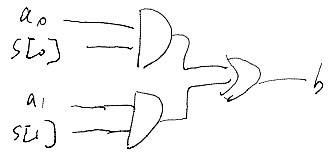
$b_1 = a_3 \vee a_2$

```
module Enc42 (a, b);
  input [3:0] a;
  output [1:0] b;
  assign b = {a[3]|a[2], a[3]|a[1]};
endmodule
```

1 bit, 2-input one-hot select mux

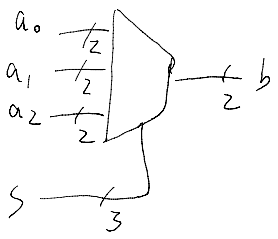
Symbol:






```
module Mux2a (a1, a0, s, b);
  input a0, a1;
  input [1:0] s;
  output b;
  assign b = (s[0]&a0)|(s[1]&a1);
endmodule
```

2 bit, 3-input Mux




```
module Mux3_2 (a2, a1, a0, s, b);
  input [1:0] a0, a1, a2;
  input [2:0] s;
  output [1:0] b;
  assign b = ({s[0], s[0]} & a0) |
             ({s[1], s[1]} & a1) |
             ({s[2], s[2]} & a2);
endmodule
```

```
always @(*) begin
  case (s)
    3'b001: b = a0;
    3'b010: b = a1;
    3'b100: b = a2;
    default: b = {8{1'bx}};
  endcase
```

don't care

```
module NAND_GATE (A, B, Z);
  :

module INV_GATE (A, Z);
  :
```

module ---

NAND_GATE U0 (In1, In2, X);
INV_GATE U1 (X, OUT1);
module output must be connected to wire

Named Port Association.

NAND_GATE U0 (IN1, IN2, X);
NAND_GATE U0 (.A(In1), .B(IN2), .Z(X));

Module Parameters.

module <module_name> (<port list>);
{ parameter <parameter name> [= <default value>];}

引用，用到的参数：<module name> # (<parameter list>) <instance name> (<port list>);

n:m decoder:

```
module dec (n, m);
    parameter n=2;
    parameter m=4;
    input [n-1:0] a;
    output [m-1:0] b;
    wire [m-1:0] b=1<<a;
endmodule.
```

if want 3->8 decoder:
Dec #(3, 8) U1 (a, b);

parameterized multiplexer (M)

```
module Mux3a (a1, a2, a3, s, b);
    parameter k=1;
    input [k-1:0] a0, a1, a2;
    input [2:0] s;
    output reg [k-1:0] b;
    always @(*) begin
    case (s)
        3'b001 : b=a0;
        3'b010 : b=a1;
        3'b100 : b=a2;
    default : b={k{1'bx}};
```

Always block Synthesis Rules - for combinational logic

Write Synthesizable Verilog. if not { tools not able to create hardware / error, or even no message.

Rules: (to ensure synthesizable)

① all outputs depend on current inputs (purely combinational).
{ 1. every input should in sensitivity list or use @(*)
2. Every output should be assigned a value - for every possible inputs

if combinational Loop no!

Y=Y;

every always block must match one of the rules.

Test:

not synthesizable only for testbench.

Initial Block. not Synthesizable. used only for test bench

initial begin

only for simulation.

order-matters, allow delays, e.g. #10

initial begin

order-makers, allow delay, e.g. #10 (time unit)

end.

only for simulation.

ModelSim: simulation.

Quartus: implementation.

eg. module test_maj // no I/O. (top level module)

```
reg [2:0] in;
wire out;
Majority m (in[0], in[1], in[2], out);   // DUT
initial begin
    in = 3'b000;
    repeat (8) begin                     // 8 times.
        #100;
        $display ("in = %b, out = %b", in, out);
        in = in + 3'b001;
    end
end
endmodule
```

generate all 8 patterns

```
module Majority (a, b, c, out);
    input a, b, c;
    output out;
    assign out = (a&b) | (a&c) | (b&c);
endmodule
```

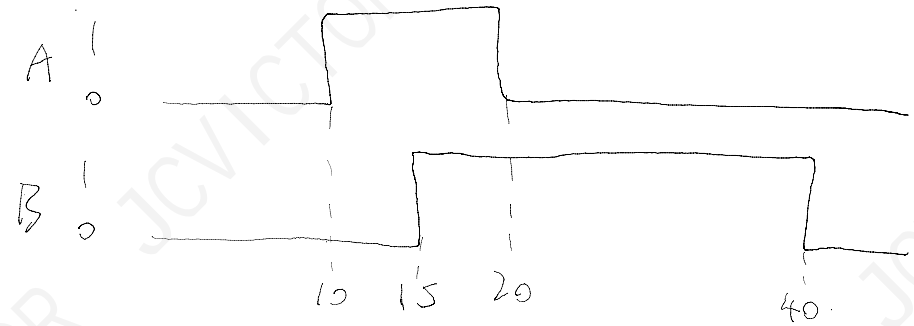
Testbench: no input/output.

How to write?

① Think of waveform I want. (change state times) 0→1 or 1→0

② Write script

e.g.




```
module AB_test;
    reg A, B;
    wire out;
    Foo DUT (A, B, out);   // instantiate design.
    initial begin          generate input patterns
        A = 1'b0;          // t=0
```

```
initial begin    generate input patterns
    A = 1'b0;    // t=0
    B = 1'b0;
    #10;
    A = 1'b1;    // t=10.
    #5;
    B = 1'b1;    // t=15
    #5;
    A = 1'b0;
    #20;
    B = 1'b0;
    #10;
    $stop(0);    stop simulating.
    end
endmodule.
```

At minimum, each line of synthesizable Verilog should be tested at least once

① Test one block.
② Test combinations of blocks.
③ Test what should happen does happen.
shouldn't happen doesn't happen.

Debugging

Simulation (Design) Errors

① Use ModelSim waveform viewer to see error
② Find the block of signal's driver.
③ check input to block in waveform viewer is correct.
④ If all inputs are OK, problem is the Verilog in the block.

If input is not OK. Trace source block.

Top Level Module

Top Level Module

specify "top level module": vsim <top_level_name>

when starting simulation in ModelSim transcript window

In Quartus, when setting up project.

In Combinational Modules.

Sensitivity list for case statements include all inputs