EES4725
Digital Circuits and FPGA Design

Chua Dingjuan
elechuad@nus.edu.sg

Lecture 3

VERILOG FOR COMBINATIONAL LOGIC

DATAFLOW – ASSIGN

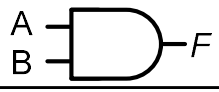
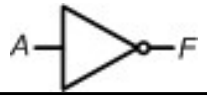
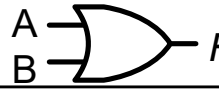
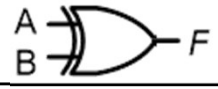
BEHAVIOURAL – ALWAYS @ (*)

STRUCTURAL - INSTANTIATION

Recall where we last left off….

```
module box(input  a, b,
           input  [1:0] c,
           output [3:0] y );
wire tmp;
reg [1:0] one = 3;
reg two;
integer three = 1;

assign y[3] = one[0];
assign y[2:1] = a + c;
assign y[0] = ( a > b );

endmodule
```

Net / Variable Name	Number of bits?	Value in dec / bin
a	1	1 / 1'b1
b	1	0 / 1'b0
c	2	2 / 2'b10
tmp	1	Z / 1'bZ
one	2	3 / 2'b11
two	1	X / 1'bX
three	32	1 / 32'h00000001
y	4	15 / 4'b1111

Gate	Symbol	Function (*F*)	Verilog Operator	Gate	Symbol	Function (*F*)	Verilog Operator
AND		$A \cdot B$	F = A & B	NOT		$\bar{A}$	F = ~A
OR		$A + B$	F = A | B	XOR		$A \oplus B$	F = A ^ B

assign F = ~w & ~x & z | ~w & x & z | w & y & z | x & y & z;

~ & ∧ |

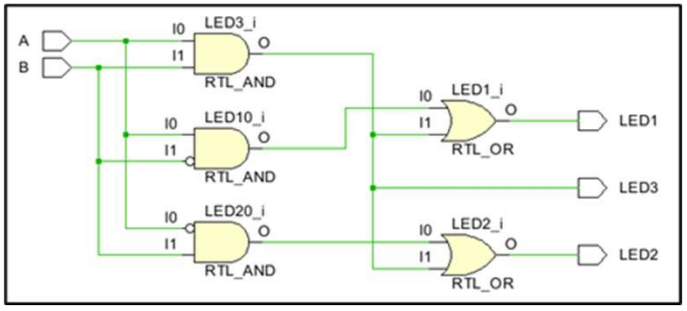
Design Workflow!

idea!

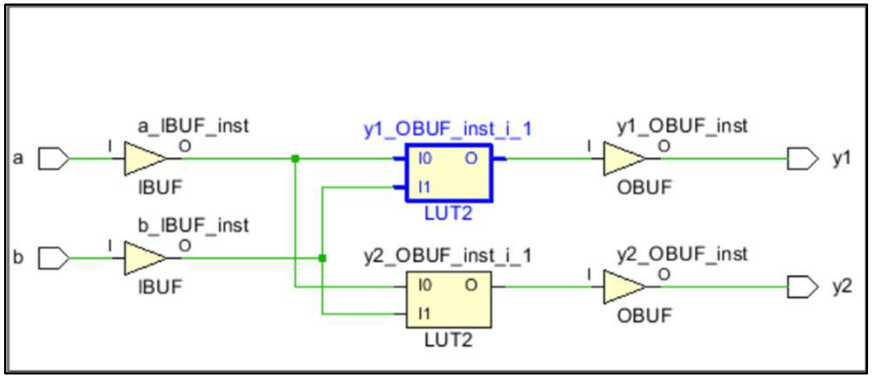
.v (verilog)

```
assign LED1 = (A & ~B) | (A & B);
```

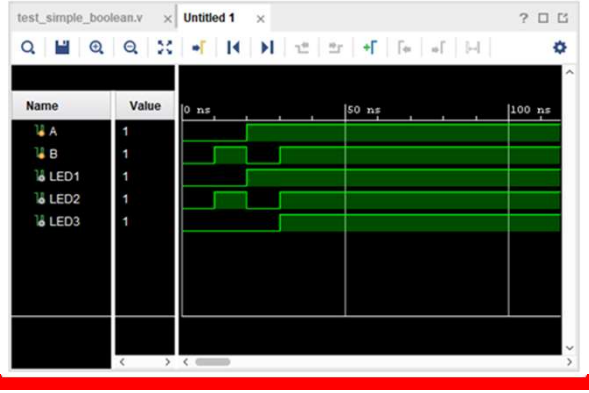
RTL Schematic



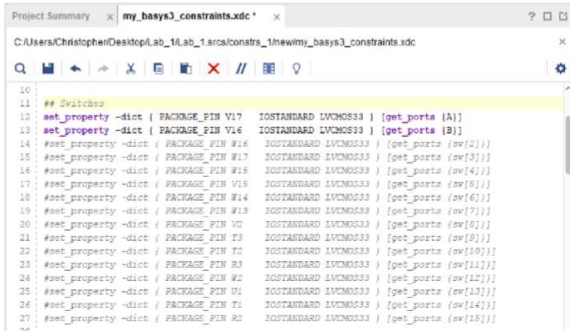
Synthesized Schematic



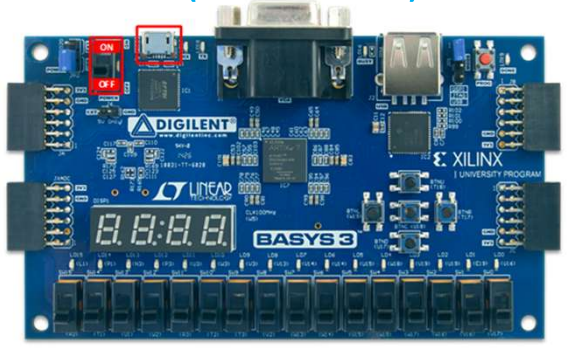
Simulation Results



.xdc (Constraints Mapping)



.bit (Bitstream)



Is this OK?

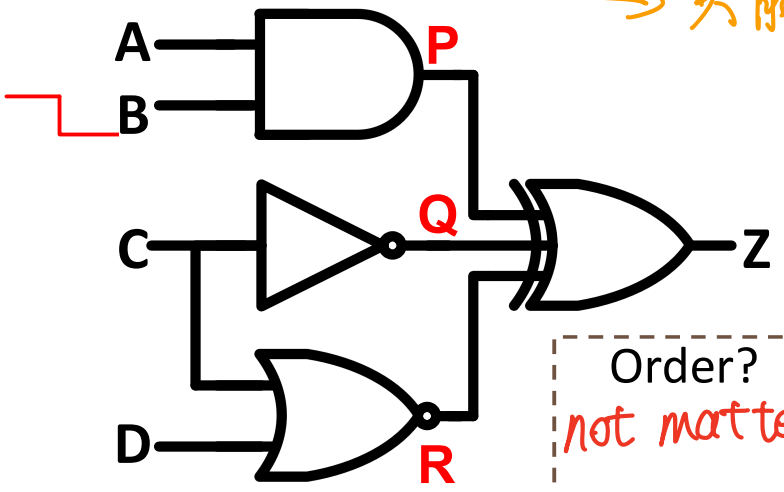
```
module notgood
(input a,b, output x, y);

   reg z;
   assign z = a | b;

endmodule
```

Continuous Assignment (Dataflow)

`assign` statements are often used to model combinational logic

只能 wire. 不能给 reg

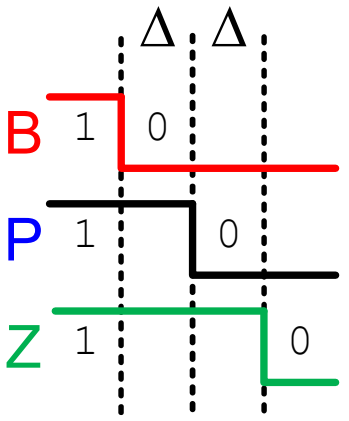
默认是 wire




```
module bigbox (input a, b, c, d,
               output z);

wire p, q, r;

assign z = p ^ q ^ r; 1
assign q = ~c;        2
assign p = a & b;     3
assign r = ~(c | d);  4

endmodule
```

internal connections

Order? not matter

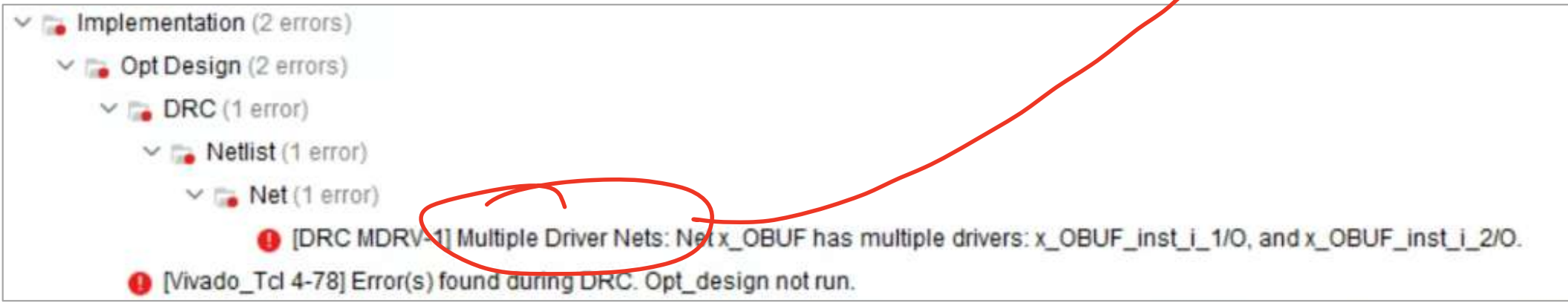


- Whenever there's an ***event*** on the RHS signal, expression is evaluated and assigned (Δ) ➔ *continuously monitored*
- Multiple statements can be executed in parallel (concurrently) 并行地
- **`wire`** is used to represent an internal connection

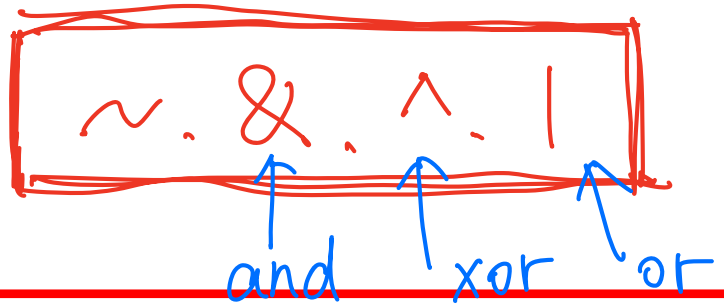
Is this OK?

```
module notgood(….);

    assign x = a | b;
    assign x = a + b;


endmodule
```

两个输入 → 可能短路



Useful Operators



- Boolean (bit-wise), logical, arithmetic, concatenation.
- Use brackets for readability, take note if *synthesizable*.

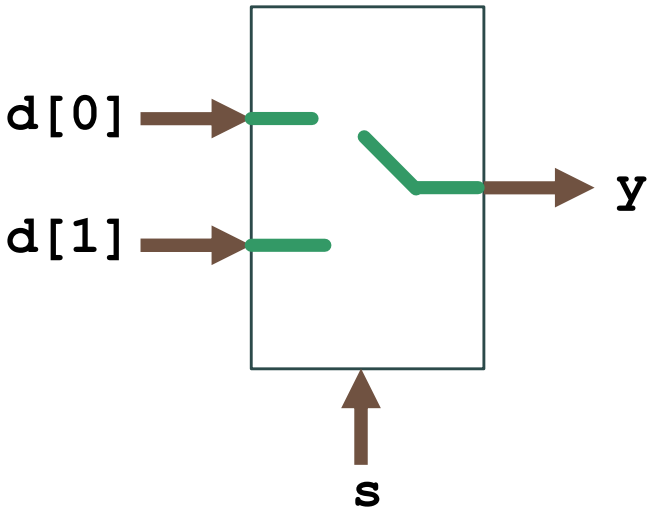
High ↑ Precedence ↓ Low

Operator	Description	Examples: a = 4'b1010, b=4'b0000
!, ~	Logical negation, Bit-wise NOT	!a = 0 , !b = 1 , ~a=4'b0101 , ~b=4'b1111
&, |, ^	Reduction (Outputs 1-bit)	&a = 0, |a=1, ^a = 0
{__,__}	Concatenation	{b, a} = 8'b00001010
{n{___}}	Replication	{2 {a} } = 8'b10101010
*, /, %,	Multiply, *Divide, *Modulus	3 % 2 = 1, 16 % 4 = 0
+, -	Binary addition, subtraction	a + b = 4'b1010
<< , >>	Shift Zeros in Left / Right	a << 1 = 4'b0100, a >> 2 = 4'b0010
<, <=, >, >=	Logical Relative (1-bit output)	(a > b) = 1
==, !=	Logical Equality (1-bit output)	(a == b)= 0 (a != b)= 1
&, ^, |	Bit-wise AND, XOR, OR	a&b = 4'b0000 a|b = 4'b1010
&&, ||	Logical AND, OR (1-bit output)	a&&b = 0 a||b = 1
?:	Conditional Operator	<out> = <condition> ? If_ONE : if_ZERO

自己从为1位开始，逐位运算 &a = 1&0&1&0

Conditional Operator…

The `?:` conditional operator allows us to select the output from a set of inputs based on a condition.



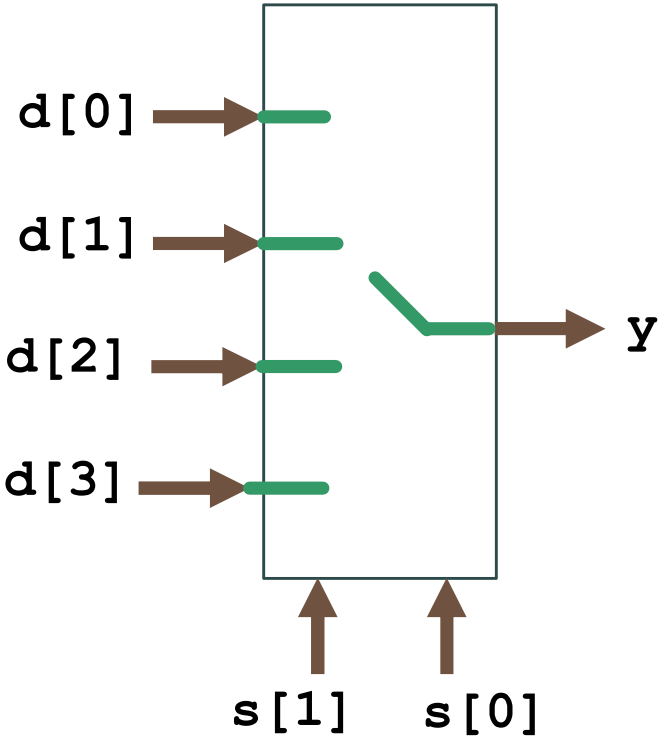

```
module mystery ( input s, input [1:0] d,
                 output y);

assign y = s ? d[1] : d[0];

endmodule
```

<output> = <condition> ? If_ONE : If_ZERO

s= 1 ➔ y = d[1], s= 0 ➔ y= d[0]

- This expression is evaluated whenever there is an event on any input.
- What is this block? 2-1 Multiplexer → MUX1

Write the code for 4-1 MUX

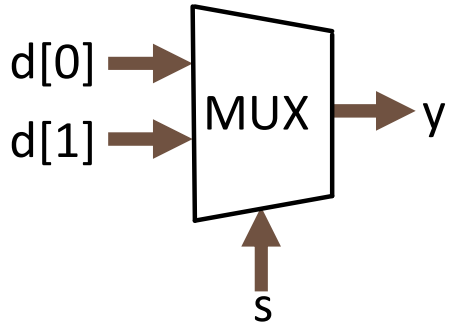


Lecture 3 Exercise Q1

Procedural Assignment : always

Behavioral, higher-level description of logic.

2 assignment types : **Blocking** & **Non-Blocking**




```
module mux21( input  s, input  d[1:0],
              output reg y);

always @(s, d)

begin
   if (sel == 1'b0)
       y= d[0];
   else
       y= d[1];

end
endmodule
```

Anything assigned in an **always** block must be declared as type **reg**

Conceptually, the **always** block runs *once* when any signal in *sensitivity list* (s,d) changes value.

Statements executed sequentially & evaluated instantaneously. ➔ Order matters!

begin and **end** behave like parentheses/brackets for conditional statements.

Some notes on: `always`

- **`always@(*)`** includes all signals that are read in statements.

- Statements within **`always`** block are executed ***sequentially***.
- Variables within sensitivity list are very important!
- **`if--else if--else`**, **`case`**, **`for`**, **`while`** can only be used in procedural assignments (always blocks)
- Multiple always blocks run in parallel, concurrently, watch out for multi-driven nets and race conditions. (next slide) 并存地
- No ~~**`assign`**~~ in always blocks!

Registers

- Anything assigned in an **`always`** block must be declared as type **`reg`**
- In Verilog, the term register (**`reg`**) simply means a variable that can hold a value. (cf. **`wire`**)

↓ 寄存器：1) 有无 posedge/negedge
2) 有无 clock
3) 有无记忆行为

Is this OK?

```
module notgood(….);
   always @ (*)
      y = y + 1;

   always @ (*)
      y = y + 3;

endmodule
```

Ref – Conditional Statements

If - else if - else

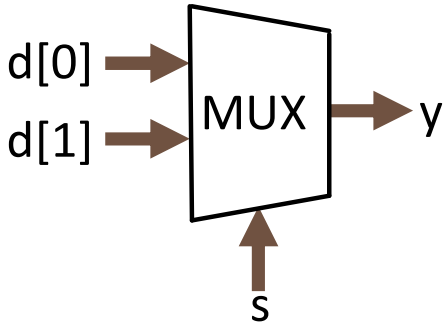
```
if ( expr )
   statement;


if ( expr )
   statement;
else
   statement;


if ( expr )
   statement;
else if ( expr )
   statement;
else if ( expr )
   statement;
else
   statement;
```

case

```
case ( expr )

   value1 : statement;
   value2 : statement;
   value3 : statement;
   ...
   default : statement;

endcase
```

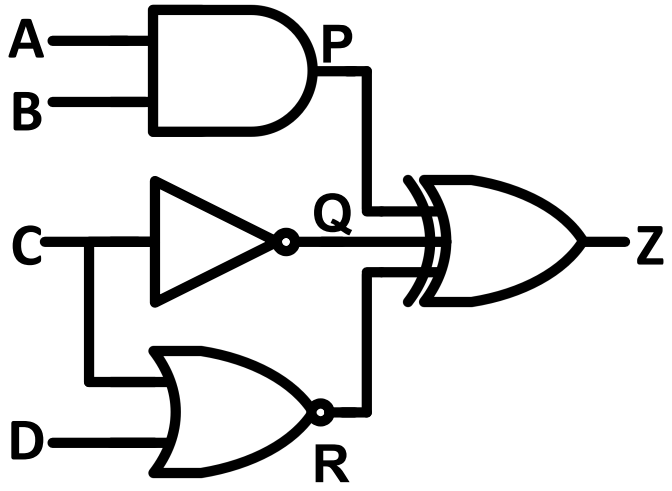
Equivalence

```
module mux21( input s, input [1:0] d, output reg y);

always @ (s, d)
begin
    if (s == 1'b0)
        y= d[0];
    else
        y= d[1];
end
endmodule
```



```
module mux ( input s, input [1:0] d, output y);

assign y = s ? d[1] : d[0];

endmodule
```

Equivalence…




```
module bigbox
(input a,b,c,d, output z);

wire p, q, r;

assign q = ~c;
assign z = p ^ q ^ r;
assign p = a & b;
assign r = ~(c | d);

endmodule
```

```
module bigbox
(input a,b,c,d, output reg z);
reg p,q,r
always @ (a,b,c)d /*
    begin
        q = ~c;
        z = p ^ q ^ r;
        p = a & b;
        r = ~(c | d);
    end
endmodule
```

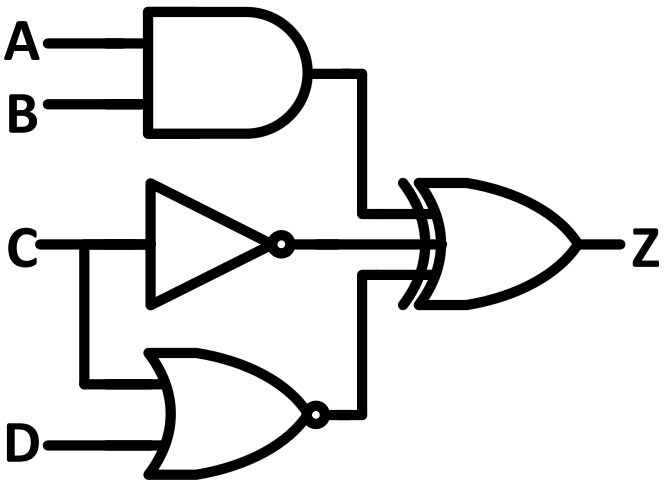
最后一行

Lecture 3 Exercise Q2

Structural Modeling – Primitives

Structural modeling can be used to connect multiple modules and gates.

Verilog provides a standard set of primitive such as basic logic gates.



Dataflow

```
module bigbox (input a,b,c,d, output z);

   assign z = (a & b) ^ ~c ^ ~(c | d) ;

endmodule
```

Structural (using primitives)

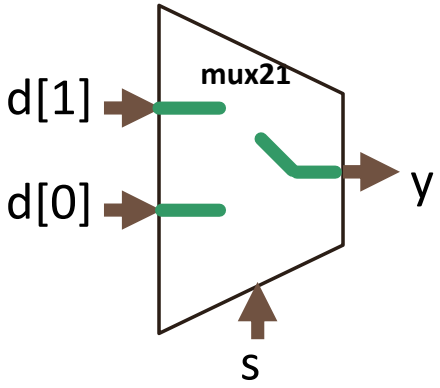
```
module bigbox (input a,b,c,d, output z);
wire p, q, r;

   and u1 (p, a, b); //output, then inputs
   not u2 (          );
   nor u3 (          );
   xor u4 (          );

endmodule
```

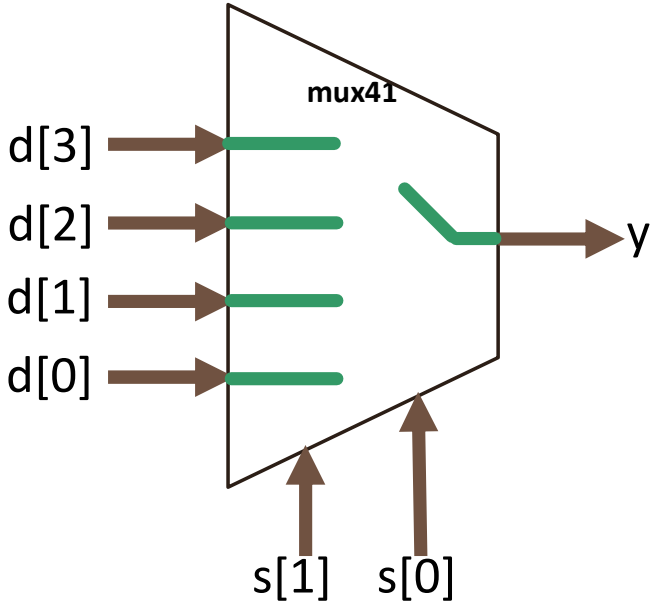
Structural Modeling

Structural modeling can be used to connect multiple modules via port connection by name and position.

```
module mux21( input s,
              input [1:0] d,
              output y);

  assign y = s ? d[1] : d[0];

endmodule
```

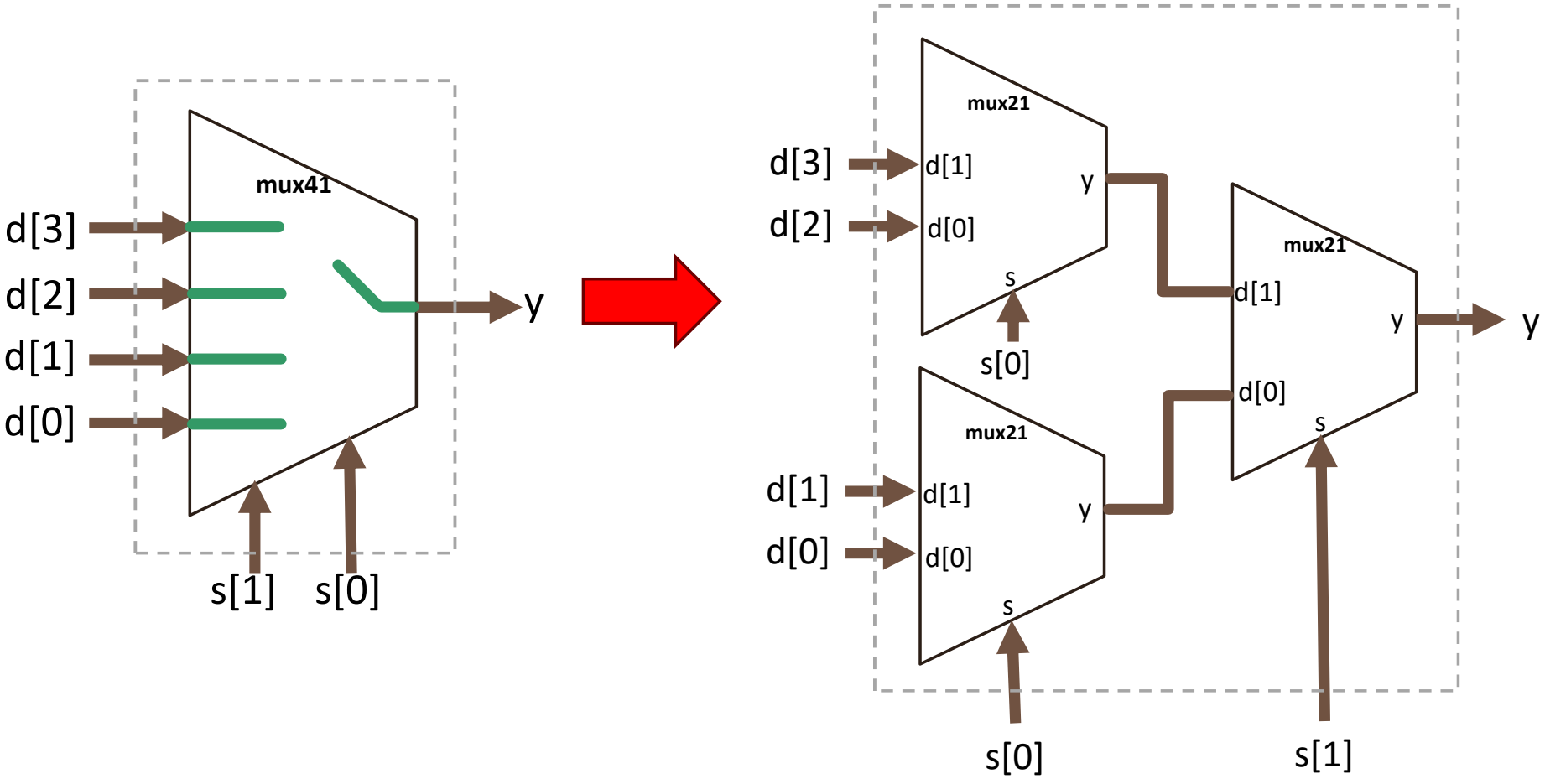
```
module mux41( input [1:0] s,
              input [3:0] d,
              output y);

… … … ?

endmodule
```





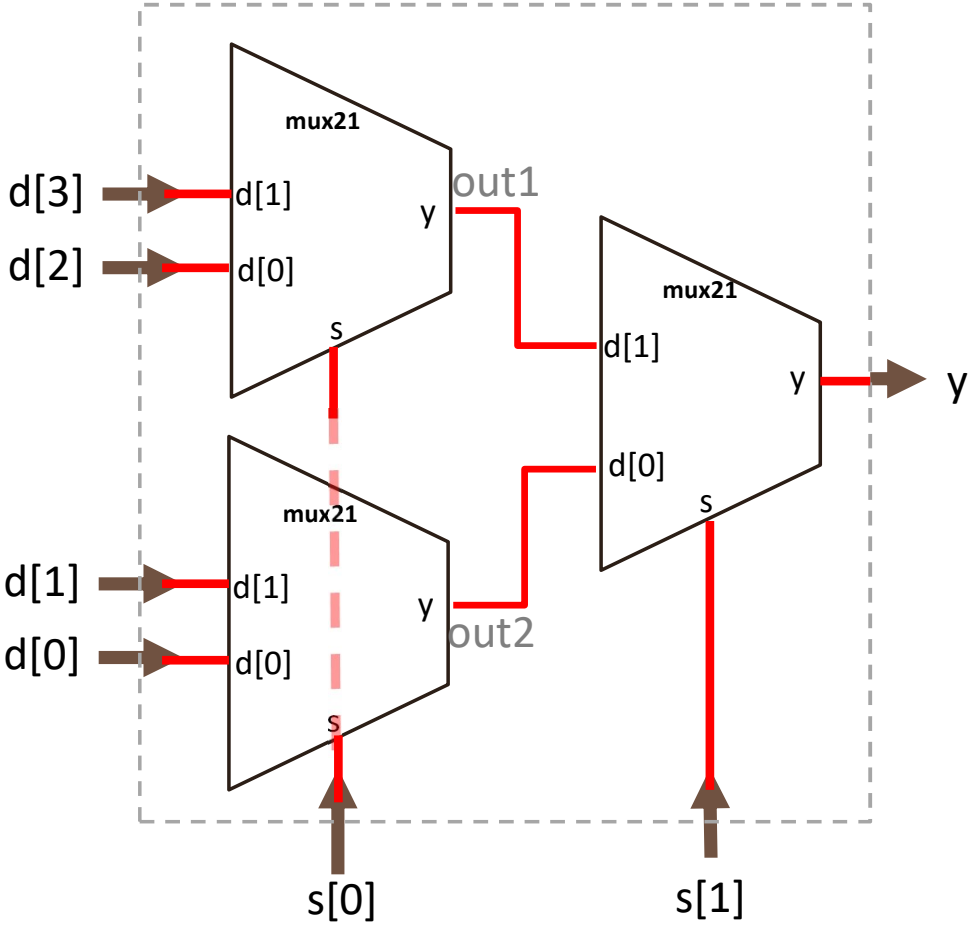
Structural Modeling

For example, a 4-to-1 multiplexer can also be implemented by combining several 2-to-1 multiplexers. How can we write this in Verilog?



Structural Modeling

For example, a 4-to-1 multiplexer can also be implemented by combining several 2-to-1 multiplexers.




```
module mux21( input s, input [1:0] d,
              output y);
   assign y = s ? d[1] : d[0];
endmodule
```

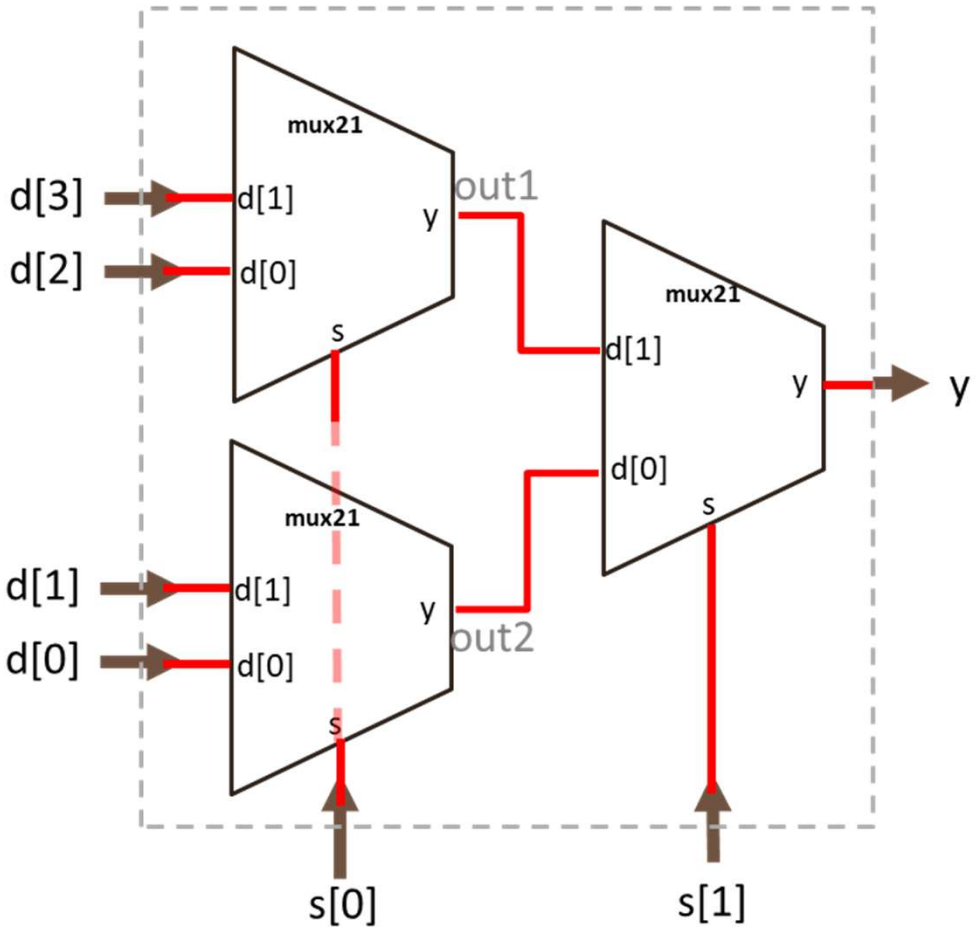
Port Connection by Position

```
module mux41( input [1:0] s,
              input [3:0] d,
              output y);

wire out1, out2;

//check for port order!
mux21 u1 (s[0], d[3:2], out1);

mux21 u2 (s[0], d[1:0], out2 );

mux21 u3 (s[1],          , y  );

endmodule
```

{out1, out2}

Structural Modeling

For example, a 4-to-1 multiplexer can also be implemented by combining several 2-to-1 multiplexers.



Port Connection by Name

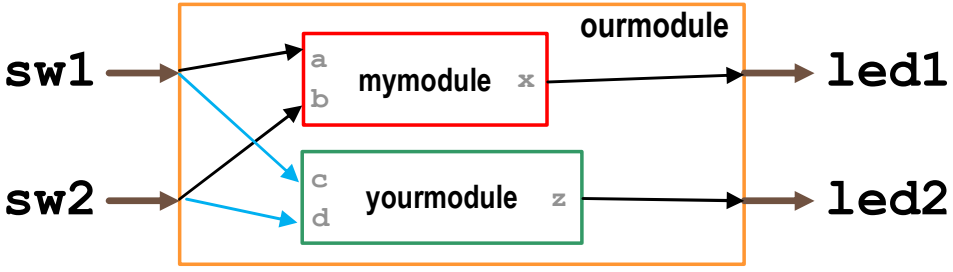
```
module mux41( input [1:0] s,
              input [3:0] d,
              output y);

wire out1, out2;

mux21 u1 (.s (s[0]),
          .d (d[3:2]),
          .y (out1)      );

mux21 u2 (.s
          .d
          .y             );

mux21 u3 (.s (s[1]),
          .d (out1,out2),
          .y (y)         );

endmodule
```

Structural Modeling

- For modular designs, the top design is often specified as interconnected blocks.
- Two examples below demonstrate port connection by position / name.

```
module mymodule (input a, b, output x);
...
endmodule
```

```
module yourmodule (input c, d, output z);
…
endmodule
```



Port Connection by Position

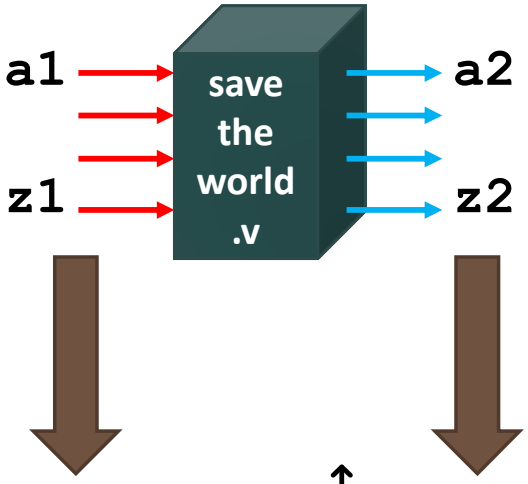
```
module ourmodule (input sw1, sw2,
                  output led1, led2);

mymodule M1 (sw1, sw2, led1);
yourmodule M2 (sw1, sw2, led2);

endmodule
```

Port Connection by Name

```
module ourmodule (input sw1, sw2,
                  output led1, led2);

mymodule M1 (  .a (sw1),
               .b (sw2),
               .x (led1) );

yourmodule M2( .z(led2),
               .c (sw1),
               .d(sw2) );

endmodule
```

Recall Simulation?

```
module savetheworld (input a1, … z1,
                     output a2, …,z2);
……. ……. ……. ……. …….
……. ……. ……. ……. …….
……. ……. ……. ……. …….
endmodule
```



How do we know our design actually works?

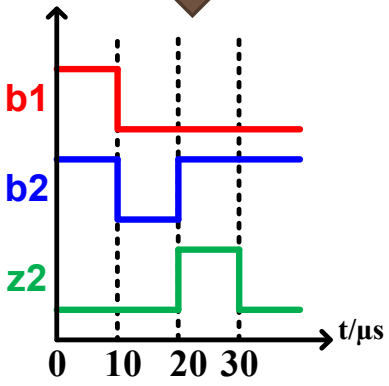
- Functional Simulation (Xilinx)

Verilog Code

```
module
……. …….
……. …….
endmodule
```

Test Bench

```
call module
a1=9;
b1=1;
//wait for 10u
#10
b1=0;
```



Method

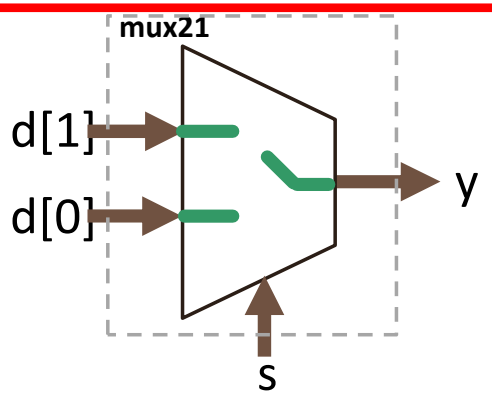
- Designer applies input values to the code
- Simulator produces corresponding outputs in truth tables / timing diagrams
- Simulators usually assume negligible propagation gate delays.

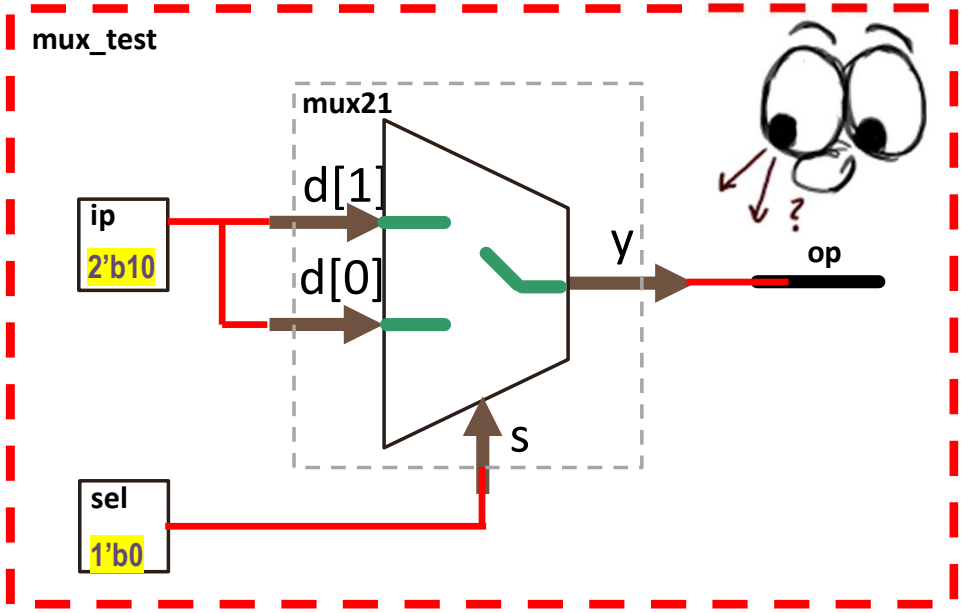
可忽略的

Simulation Testbench Example

```
module mux21( input s,
              input [1:0] d,
              output y);
  assign y = s ? d[1] : d[0];
endmodule
```

```
module mux_test();

reg [1:0] ip = 0;
reg sel = 0;
wire op;

mux21 dut (sel, ip, op);

initial begin
    ip = 2'b10;
    sel = 1'b0;
    #10; //wait 10 time units
end

endmodule
```





Pre-Lab Preparation Q3