



Sharif University of Technology  
Department of Computer Engineering

# Digital System Design

## Gate-level modelling

Siavash Bayat-Sarmadi

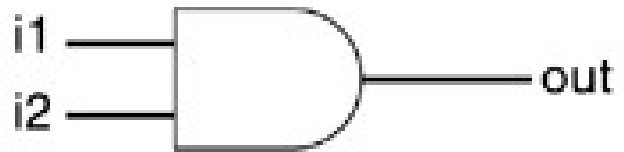
# Outline

2

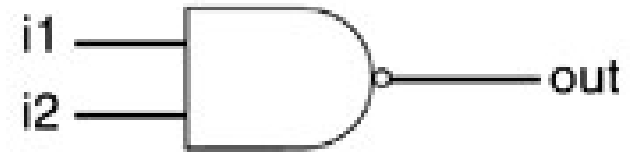
- Gate types
  - ▣ Basic gates
  - ▣ Three-state
  - ▣ Structural design using primitive gates
- Gate delays
  - ▣ Gate-level delay types
  - ▣ Gate-level delay syntax

# Basic Gates

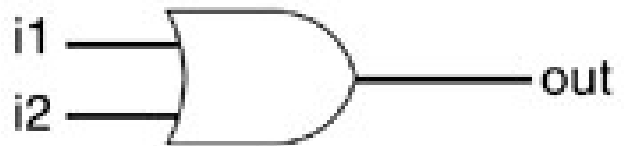
3



and



nand



or



nor



xor



xnor

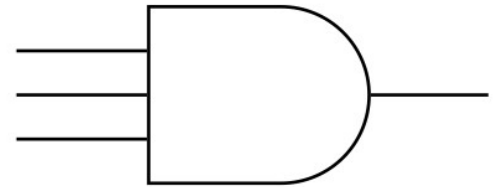
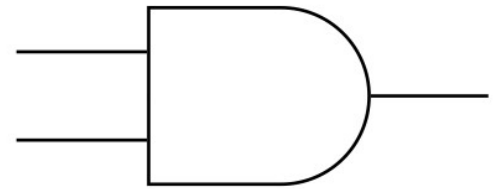
# Instantiation Syntax

4

- Similar to instantiating modules
- Name optional for these primitives
- Example

```
wire OUT, IN1, IN2, IN3;  
and a2input(OUT, IN1, IN2);
```

```
and (OUT, IN1, IN2, IN3);
```



# 4-Value Truth Table

5

## □ Example

and	0	1	X	Z	xnor	0	1	X	Z
0	0	0	0	0	0	1	0	X	X
1	0	1	X	X	1	0	1	X	X
X	0	X	X	X	X	X	X	X	X
Z	0	X	X	X	Z	X	X	X	X

# Buf and Not Gates

6

- Input

- ▣ Last port in terminal list

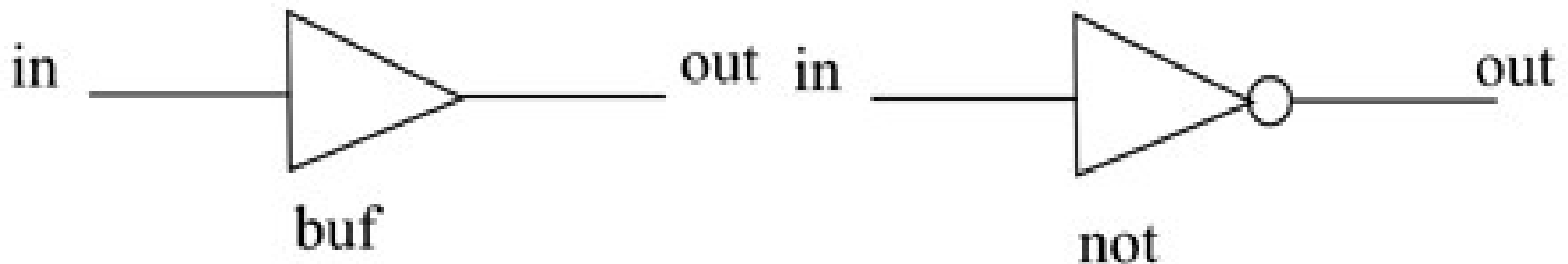
- Outputs

- ▣ All other ports

- Example

- ▣ `buf b1_2out (OUT1, OUT2, IN);`

- ▣ `not (OUT1, IN); //Legal`



# 4-Value Truth Table

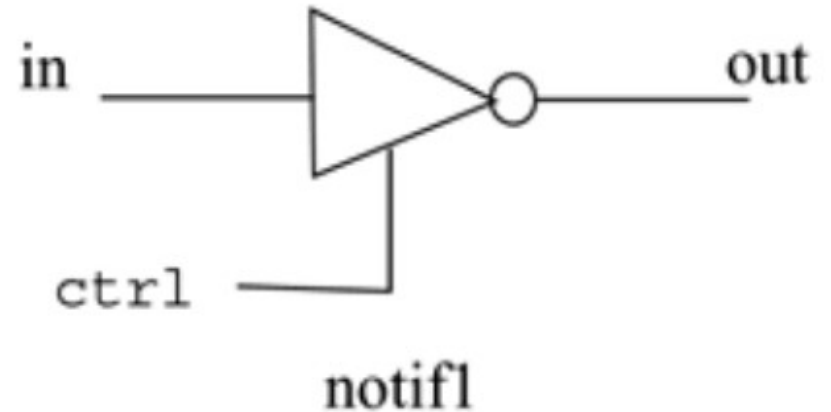
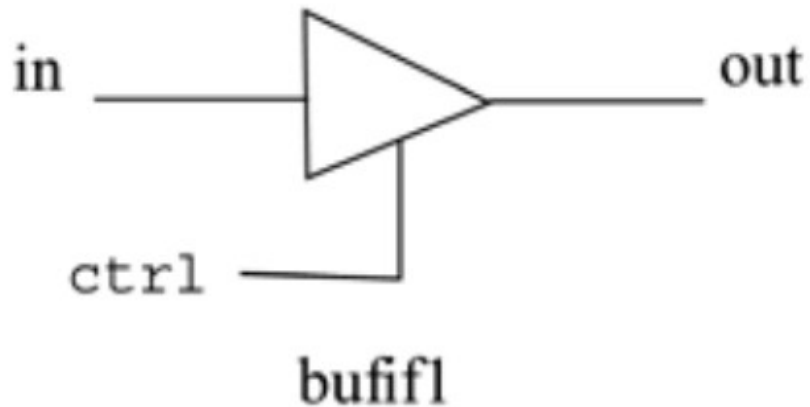
7

buf	in	out
	0	0
	1	1
	X	X
	Z	X

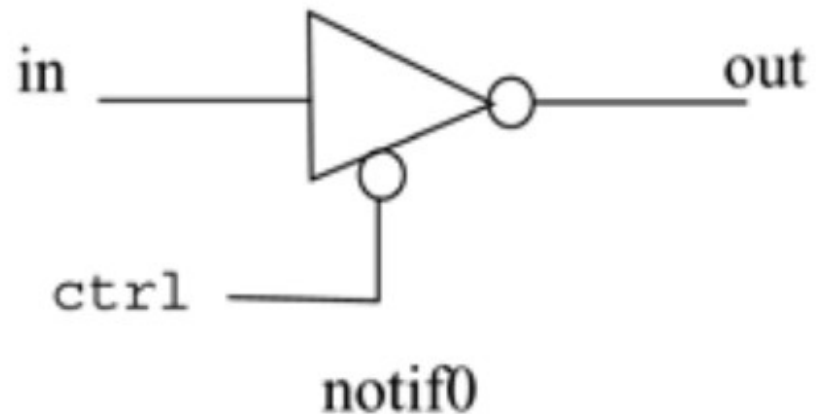
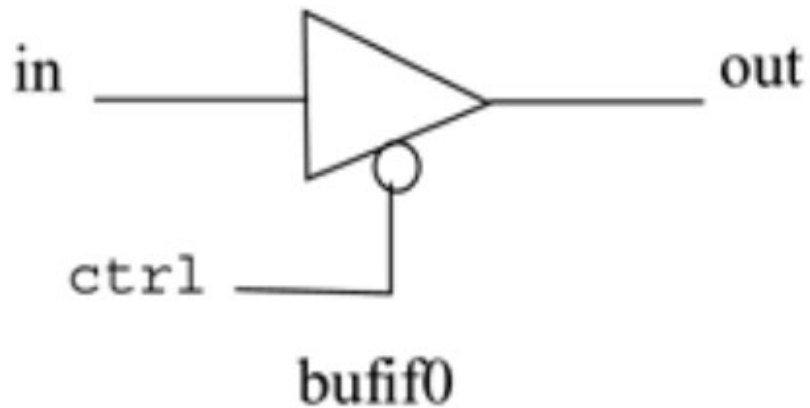
not	in	out
	0	1
	1	0
	X	X
	Z	X

# Three-state Logic Gates

8



```
bufif1 b1 (out, in, ctrl);    notif1 n1 (out, in, ctrl);
```



```
bufif0 b1 (out, in, ctrl);    notif0 n1 (out, in, ctrl);
```



# Array of Instances

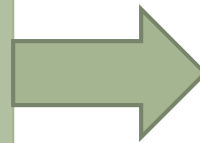
9

```
wire [2:0] OUT, IN1,  
        IN2;
```

```
nand n0 (OUT[0],  
        IN1[0], IN2[0]);
```

```
nand n1 (OUT[1],  
        IN1[1], IN2[1]);
```

```
nand n2 (OUT[2],  
        IN1[2], IN2[2]);
```



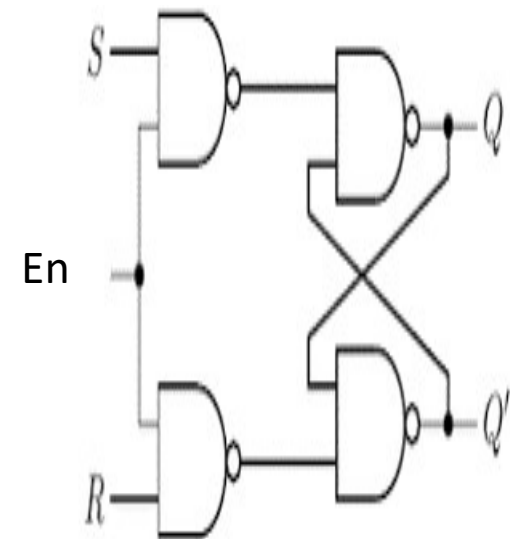
```
wire [2:0] OUT,  
        IN1, IN2;
```

```
nand n [2:0] (OUT,  
        IN1, IN2);
```

# Example: SR Latch

10

```
module SR_latch(Q,  
Qbar, S,  $\overline{R}$ , En);  
  
output Q, Qbar;  
input S, R, En;  
wire Sbar, Rbar;  
nand n1(Sbar, S, En),  
      n2(Rbar, R, En),  
      n3(Q, Sbar, Qbar),  
      n4(Qbar, Rbar, Q);  
endmodule
```



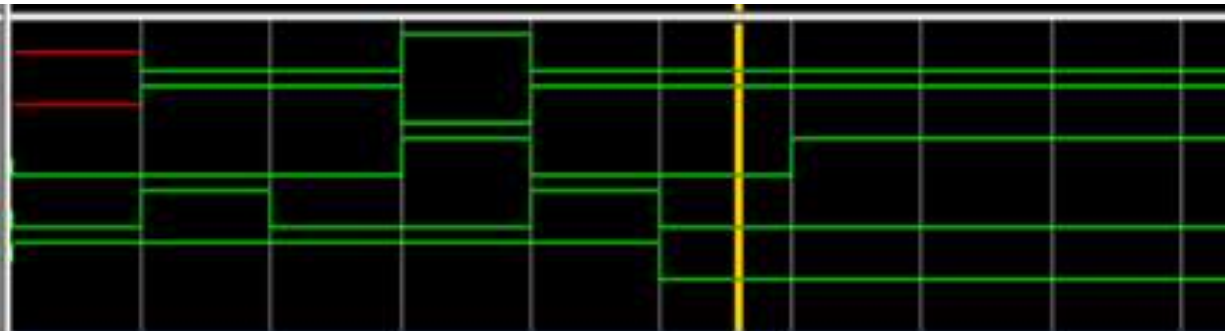
# Example

11

```
module Top;
  wire q, qbar;
  reg set , reset , en;
  SR_latch l1(q, qbar, set, reset, en);
  initial
  begin
    $monitor($time, "set = %b, reset= %b, en=%b,
q= %b\n", set, reset, en, q);

    set = 0; reset = 0 ; en = 1;
    #10 reset = 1;
    #10 reset = 0;      #10 set = 1;
    #10 set = 0; reset = 1;
    #10 set = 0; reset = 0 ; en= 0;
    #10 set = 1; reset = 0;
  end
endmodule
```

◆ /Top/q	St0
◆ /Top/qbar	St1
◆ /Top/set	0
◆ /Top/reset	0
◆ /Top/control	0



Transcript

File Edit View Bookmarks Window Help

Transcript



add wave -position insertpoint sim:/Top/\*

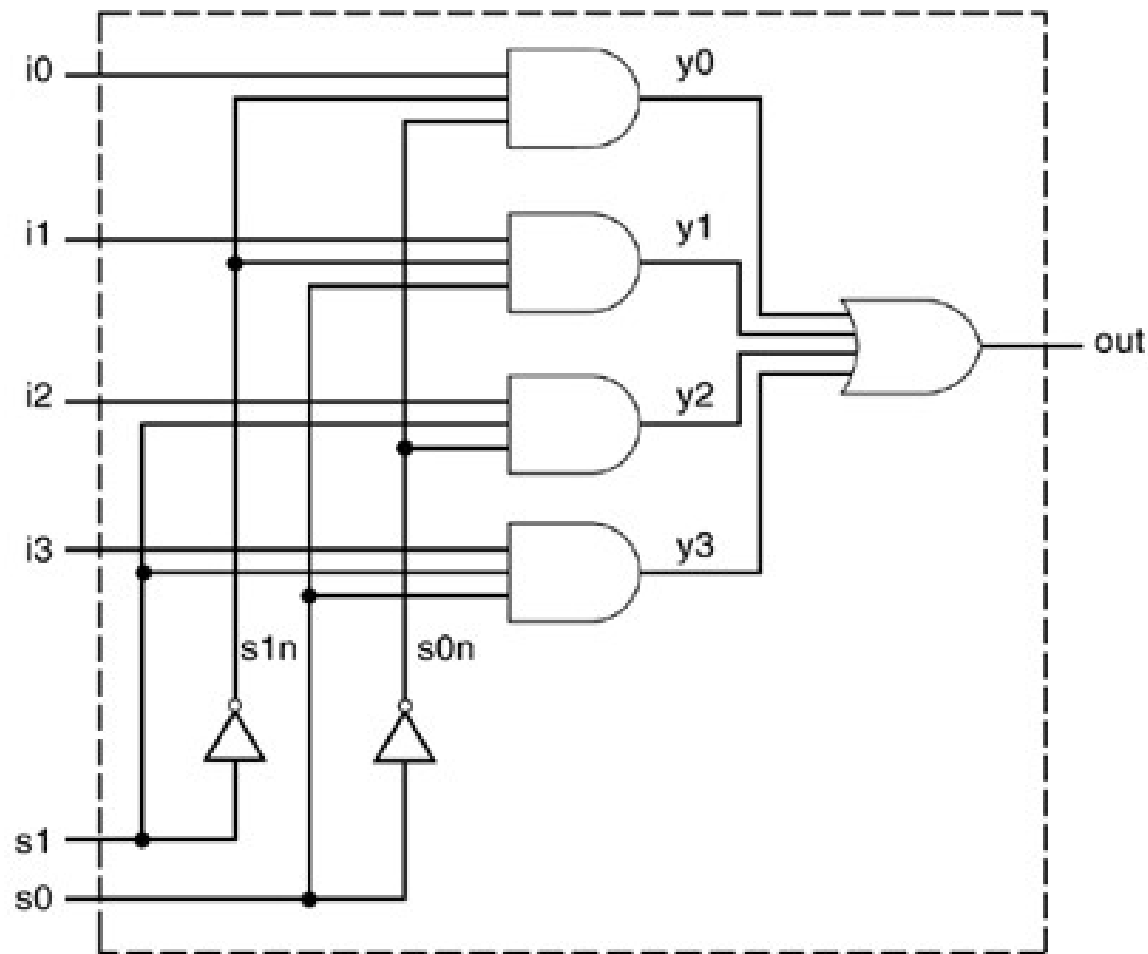
VSIM 3> run

```
#           0 set = 0, reset= 0, control=1, q= x
#
#           10 set = 0, reset= 1, control=1, q= 0
#
#           20 set = 0, reset= 0, control=1, q= 0
#
#           30 set = 1, reset= 0, control=1, q= 1
#
#           40 set = 0, reset= 1, control=1, q= 0
#
#           50 set = 0, reset= 0, control=0, q= 0
#
#           60 set = 1, reset= 0, control=0, q= 0
#
run
```

# Example: Multiplexer

13

## □ 4-to-1 Multiplexer



# Example (cont'd.)

14

```
module mux4_to_1 (output out, input i0, i1, i2, i3, s1, s0);  
    wire s1n, s0n;  
    wire y0, y1, y2, y3;  
    not (s1n, s1);  
    not (s0n, s0);  
    and (y0, i0, s1n, s0n);  
    and (y1, i1, s1n, s0);  
    and (y2, i2, s1, s0n);  
    and (y3, i3, s1, s0);  
    or (out, y0, y1, y2, y3);  
endmodule
```

# Example (cont'd.)

15

```
module stimulus;
reg i0, i1, i2, i3, s1, s0;
wire out;
mux4_to_1 my_mux(out,i0,i1,i2,i3,s1,s0);
initial begin i0 = 0;i1 = 1;i2 = 0; i3 = 1;
#1 $display("i0=%b, i1=%b, i2=%b, i3=%b\n",
,i0,i1,i2,i3);
s1 = 0; s0 = 0;
#1 $display("s1=%b, s0=%b, out=%b\n",
s1,s0,out);
```

# Example

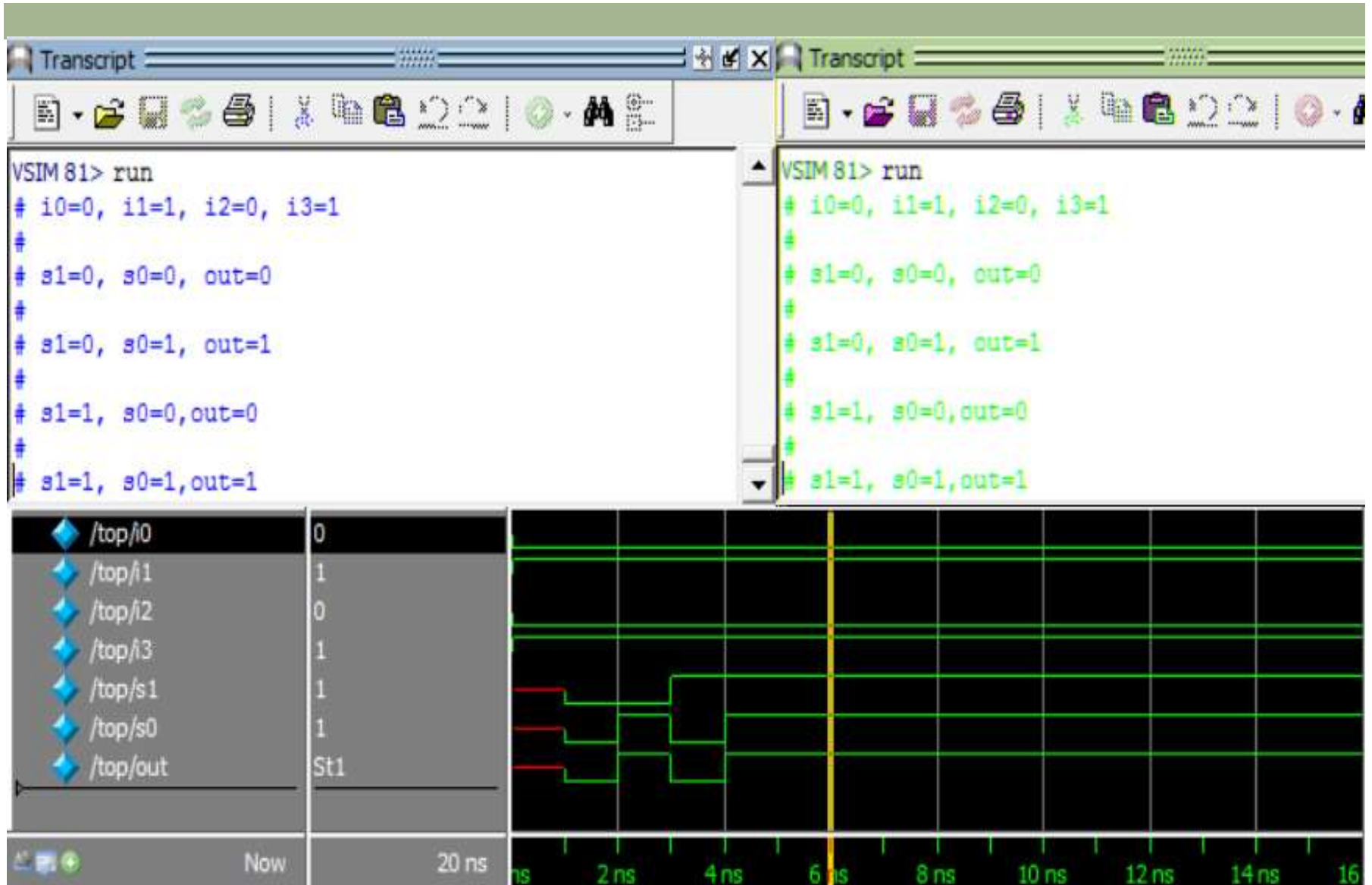
16

```
s1 = 0; s0 = 1;
#1 $display("s1=%b, s0=%b, out=%b\n"
,s1,s0,out);
s1 = 1; s0 = 0;
#1 $display("s1=%b, s0=%b, out=%b\n"
,s1,s0,out);
s1 = 1; s0 = 1;
#1 $display("s1=%b, s0=%b, out=%b\n"
,s1,s0,out);
end
endmodule
```



# Example

17



# Gate Delays

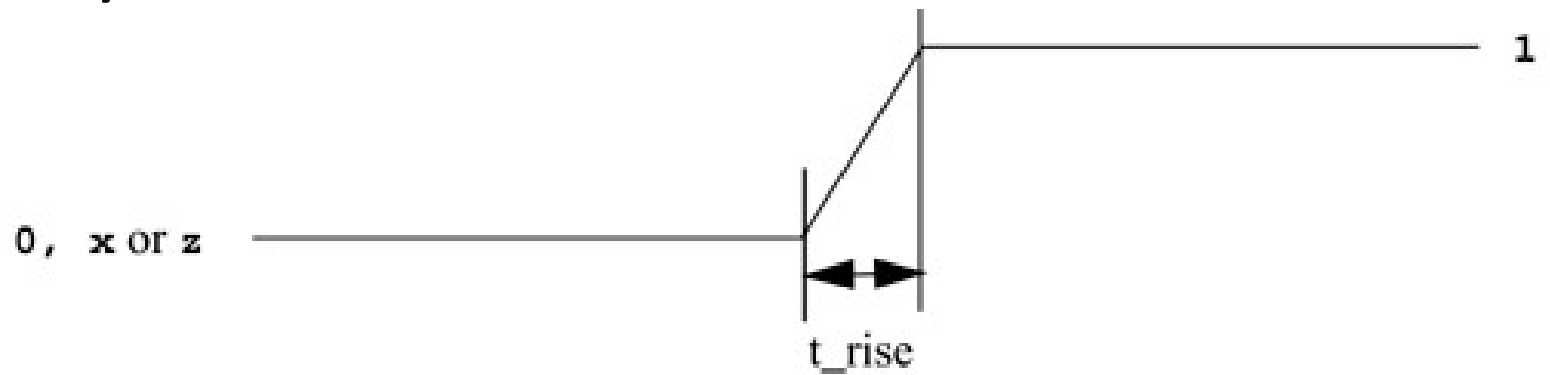
18

- Types of delay
- Gate-level delay
- Gate-level syntax

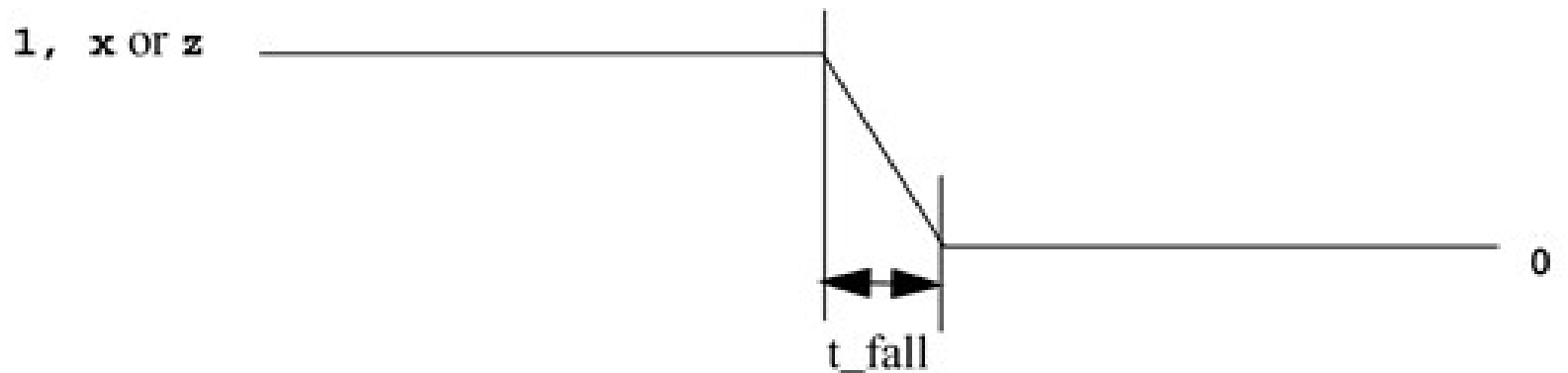
# Types of delay

19

## □ Rise delay



## □ Fall delay



# Types of delay (cont'd.)

20

- Turn-off delay
  - ▣ Gate output transition to **z** from other values
- If one delay specified
  - ▣ Same delay for all transitions
- If two
  - ▣ Assigned to rise and fall
  - ▣ turn-off = min (rise, fall)
- Minimum of the three is considered for a transition to **x**

# Example

21

- One delay specified
  - ▣ and `#(delay_time) a1(out, i1, i2);`
  
- Two delays specified
  - ▣ and `#(rise_val, fall_val) a2(out, i1, i2);`
  
- Three delays specified
  - ▣ `bufif0 #(rise_val, fall_val, turnoff_val) b1(out, in, control);`

# Min, Typ and Max Values

22

- Each delay type can have 3 values
  - ▣ Minimum
  - ▣ Typical
  - ▣ Maximum
- Only one value used during a simulation
  - ▣ Chosen at the start
  - ▣ Typical used if not specified

# Example

23

```
/*  
one delay,  
if mindelays chosen -> delay = 4  
if typdelays chosen -> delay = 5  
if maxdelays chosen -> delay = 6  
*/  
bufif1 # (4:5:6) b1 (t1, in1, ctrl1);
```

# Example

24

```
/*  
two delays  
if mindelays chosen -> rise=4,fall=3,turn-  
off=3  
if typdelays chosen -> rise=5,fall=4,turn-  
off=4  
if maxdelays chosen -> rise=6,fall=5,turn-  
off=5  
*/  
bufif1 # ( 4:5:6,3:4:5) b2(t2,in2,ctrl2);
```



# Example

25

```
/*  
three delays  
  if mindelays chosen -> rise=1,fall=2,turn-  
off=1  
  if typdelays chosen -> rise=2,fall=3,turn-  
off=2  
  if maxdelays chosen -> rise=3,fall=4,turn-  
off=3  
*/  
bufif1 # (1:2:3, 2:3:4, 3:4:5 ) b3  
(t3, In3, ctrl3);
```

# Example

26

The screenshot displays the ModelSim interface. The top section contains a 'Transcript' window with the following commands:

```
ModelSim> vsim -novopt work.buif +typdelay
# vsim +typdelay -novopt work.buif
# Loading work.buif
add wave -position insertpoint sim:/buif/*
```

Below the transcript is a code editor showing the Verilog code for a module named 'buif':

```
1 module buif;
2   wire out;
3   reg in,ctrl;
4   buif1 #(1:2:3, 2:3:4, 3:4:5 ) b (out,in,ctrl);
5   initial begin
6     in = 1 ; ctrl = 1;
7   end
8   endmodule
```

The bottom section shows a timing diagram. On the left, a list of signals is provided: /buif/out, /buif/in, and /buif/ctrl. The main area is a grid representing time in nanoseconds (ns), with major ticks every 2 ns from 0 to 14 ns. A vertical yellow cursor is positioned at 2 ns. The signal /buif/out is shown as a red line that transitions from low to high at 2 ns. The signals /buif/in and /buif/ctrl are shown as green lines that remain at a constant high level throughout the simulation.

Signal	Value
/buif/out	St1
/buif/in	1
/buif/ctrl	1

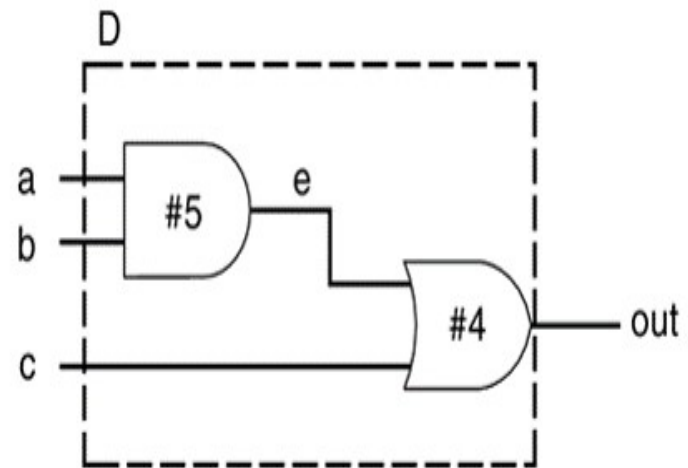
Timing Diagram Data:

Time (ns)	/buif/out	/buif/in	/buif/ctrl
0	Low	High	High
2	High	High	High
4	High	High	High
6	High	High	High
8	High	High	High
10	High	High	High
12	High	High	High
14	High	High	High

# Delay example

27

```
module D (out, a, b, c);  
  output out;  
  input a, b, c;  
  
  wire e;  
  
  and # (5)  a1 (e, a, b);  
  or  # (4)  o1 (out, e, c);  
  
endmodule
```



# Delay example (cont'd.)

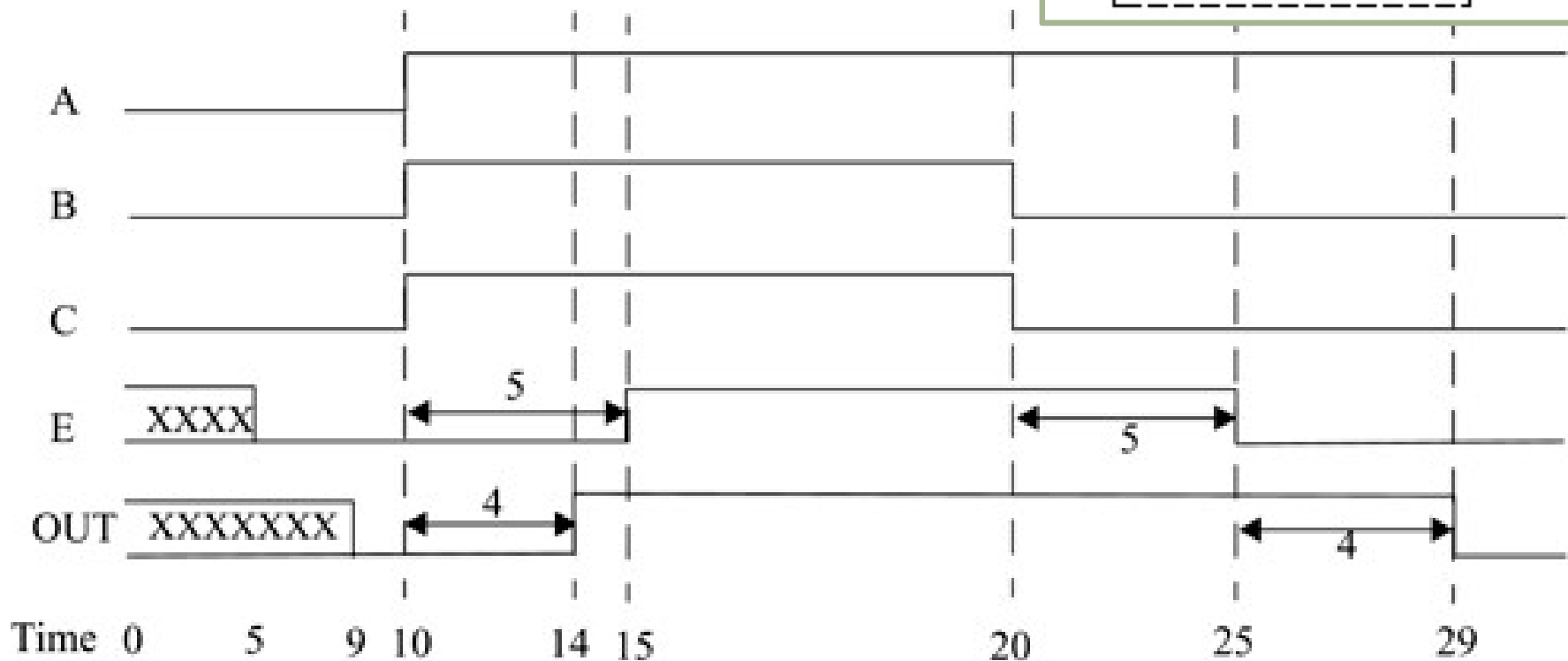
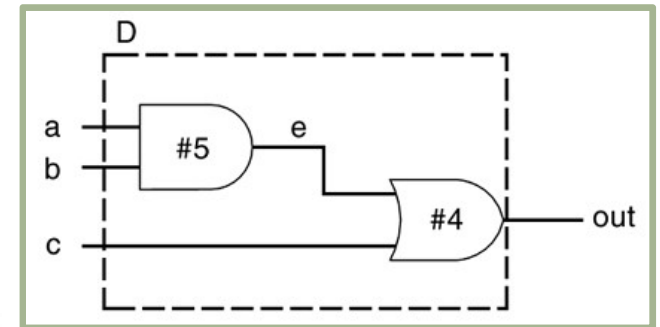
28

```
module stimulus;
reg A,B,C;
wire OUT;
D d1 (OUT,A,B,C);
initial begin
    A = 1'b0; B=1'b0; C = 1'b0;
    #10 A = 1'b1; B = 1'b1; C = 1'b1;
    #10 A = 1'b1; B = 1'b0; C = 1'b0;
    #20 $finish;
end
endmodule
```

# Delay example (cont'd.)

29

## □ Delay simulation waveform



# Generate Blocks

30

- Dynamic generation
  - ▣ At elaboration time
  - ▣ Before the simulation begins
- Keywords
  - ▣ generate
  - ▣ endgenerate

# Generate Blocks (Cont'd.)

31

- What can be instantiated
  - ▣ **Variable declarations**
  - ▣ **Modules**
  - ▣ **Gate primitives**
  - ▣ ...

# Generate Blocks (Cont'd.)

32

- Allowed in the generate scope
  - ▣ Variable declaration
  - ▣ Parameter redefinition
  - ▣ ...
- Not permitted in a generate statement
  - ▣ parameters, local parameters declarations
  - ▣ input, output, inout declarations
  - ▣ ...



# Generate Blocks (Cont'd.)

33

- Particularly convenient when
  - ▣ The same operation or module instance is repeated
    - generate loop
  - ▣ Certain code is conditionally included based on parameter definitions
    - generate conditional
    - generate case

# Generate loop

34

```
genvar i;  
generate  
    for  
    begin  
        //statement  
    end  
endgenerate
```

- **genvar** can be defined only in a generate loop
- Generate loops can be nested

# Example 1

35

```
module bitwise_xor (out, i0, i1);  
parameter N = 32;  
output [N-1:0] out;  
input [N-1:0] i0, i1;  
genvar j;  
generate for (j=0;j<N;j=j+1) begin:xor_loop  
    xor g1 (out[j], i0[j], i1[j]);  
end  
endgenerate  
endmodule
```

## Example 2 (Cont'd.)

36

```
module ripple_adder(co, sum, a0, a1, ci);  
parameter N = 4; //4-bit bus by default  
//port declarations  
output [N-1:0] sum;  
output co;  
input [N-1:0] a0 ,a1;  
input ci;  
wire [N-1:0] carry; //Local wire declaration  
//Assign 0th bit of carry equal to carry  
//input  
buf b1 (carry[0], ci);
```

## Example 2 (Cont'd.)

37

```
genvar i;
generate for (i=0; i<N; i=i+1) begin:r_loop
    wire t1, t2, t3;
    xor g1 (t1, a0[i], a1[i]);
    xor g2 (sum[i], t1, carry[i]);
    and g3 (t2, a0[i], a1[i]);
    and g4 (t3, t1, carry[i]);
    or  g5 (carry[i+1], t2, t3);
end//end of the for loop
endgenerate//end of the generate block
buf b2 (co, carry[N]);
endmodule
```

## Example 2 (Cont'd.)

38

### □ genvar i

- ▣ A temporary loop variable
- ▣ Used only in the evaluation of the generate block
- ▣ Does not exist during the simulation
  - generate loops are unrolled before simulation

# Example 2 Hierarchical instance names

39

Xor	And	Or
r_loop[0].g1	r_loop[0].g3	r_loop[0].g5
r_loop[1].g1	r_loop[1].g3	r_loop[1].g5
r_loop[2].g1	r_loop[2].g3	r_loop[2].g5
r_loop[3].g1	r_loop[3].g3	r_loop[3].g5
r_loop[0].g2	r_loop[0].g4	
r_loop[1].g2	r_loop[1].g4	
r_loop[2].g2	r_loop[2].g4	
r_loop[3].g2	r_loop[3].g4	

## Example 2 Hierarchical instance names (Cont'd.)

40

- Generated instances are connected with the following generated nets

<code>r_loop[0].t1</code>	<code>r_loop[0].t2</code>	<code>r_loop[0].t3</code>
<code>r_loop[1].t1</code>	<code>r_loop[1].t2</code>	<code>r_loop[1].t3</code>
<code>r_loop[2].t1</code>	<code>r_loop[2].t2</code>	<code>r_loop[2].t3</code>
<code>r_loop[3].t1</code>	<code>r_loop[3].t2</code>	<code>r_loop[3].t3</code>



# Generate Conditional

41

```
module multiplier (product, a0, a1);  
parameter a0_width = 8;  
parameter a1_width = 8;  
localparam product_width =  
a0_width + a1_width;  
  
output [product_width-1:0] product;  
input [a0_width-1:0] a0;  
input [a1_width-1:0] a1;
```

# Generate Conditional

42

**generate**

```
if ((a0_width < 8) || (a1_width < 8))  
    cla_multiplier #(a0_width,a1_width)  
        m0 (product, a0, a1);
```

**else**

```
    tree_multiplier #(a0_width,a1_width)  
        m0 (product, a0, a1);
```

**endgenerate**

**endmodule**

Part-3

# 16-bit RISC Processor

Structural ALU

# Outline

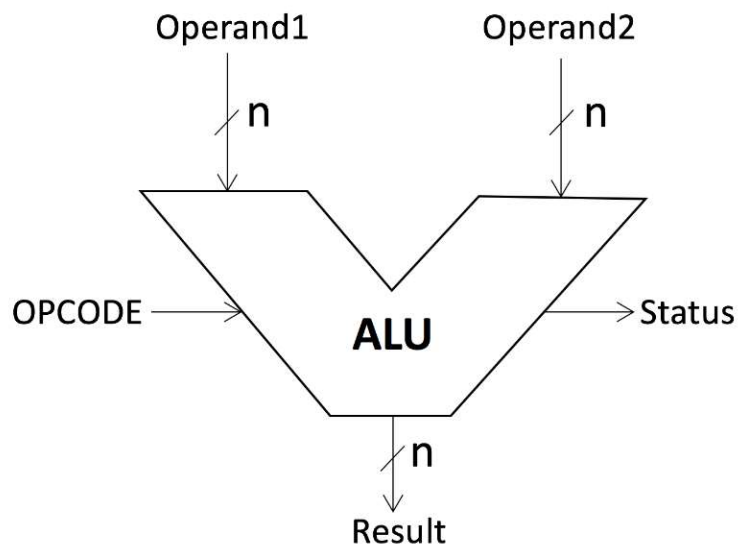
44

- What is an ALU?
- Design simple ALU with 8 instruction in structural level

# ALU

45

- What is an ALU?
  - ▣ **A**rithmetic **L**ogic **U**nit (ALU)
  - ▣ digital circuit used to perform arithmetic and logic operations

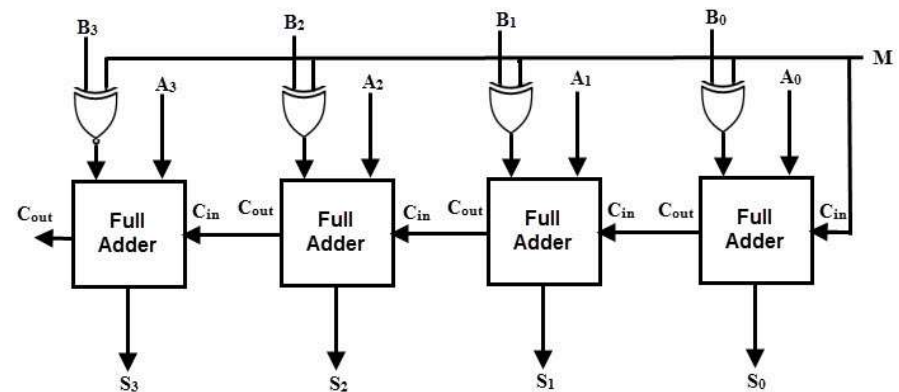


# ALU (cont.)

46

## □ ALU instructions in this project:

- 1) Subtractor
- 2) Adder
- 3) Compare
- 4) Inverter
- 5) Shift Left
- 6) Shift Right
- 7) OR
- 8) AND



4-bit Comparator

