UCI University of California, Irvine

EECS 31L: INTRODUCTION TO DIGITAL DESIGN LAB LECTURE 3

Salma Elmalaki
salma.elmalaki@uci.edu

The Henry Samueli School of Engineering
Electrical Engineering and Computer Science
University of California, Irvine

LECTURE 3: LOGISTICS

- **All questions should be posted on canvas**
- Lab 3 deadline is Monday 2/1/2021 at 11:00 pm on canvas
  - No late submission will be considered
- For Mac users, install VMs for emergencies
- Copying files from the server you can use:

```
scp username@hostname:/path/to/remote/file /path/to/local/file
```

LECTURE 3: OVERVIEW

- Typical mistakes in Verilog
- Data flow description
- Concurrent signal assignment + examples
- Assignment with delay + examples
- Concurrent code Verilog

LECTURE 3: OVERVIEW

- Typical mistakes in Verilog
- Data flow description
- Concurrent signal assignment + examples
- Assignment with delay + examples
- Concurrent code Verilog

UCI University of California, Irvine

TYPICAL MISTAKES IN VERILOG

```
module mult_comb (a, b, P)
  input [1:0] A, b;
  output (3:0) P;
  P[0] = a[0] and b[0];
  Assign p[0]= S[0] | a[0];
endmodule;
```

- `module mult_comb (a, b, P)`: Semicolon (;) is missing at the end of the statement
- `input [1:0] A, b;`: 'A' is not defined; it should be lowercase
- `output (3:0) P;`: Brackets [3:0] should be used instead of parenthesis
- `P[0] = a[0] and b[0];`: The word 'assign' is missing; The word 'and' cannot be used in verilog. The logical operator "&" should be used
- `Assign p[0]= S[0] | a[0];`: S[0] is a vector and not declared; 'assign' is a lowercase
- `endmodule;`: No semicolon (;)at the end of 'endmodule'

TYPICAL MISTAKES IN VERILOG

```
reg out;
always @(a,b) begin
 out = a & b;
end


reg out2;
always @(a,b) begin
 ~~assign~~ out2 = a ^ b;
end
```

There is no ‘assign’ keyword in always blocks!

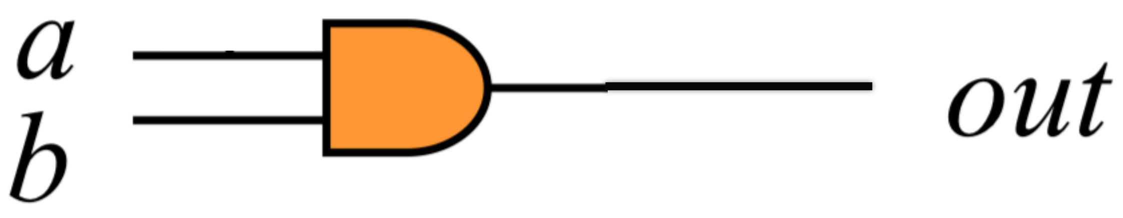
TYPICAL MISTAKES IN VERILOG

```
module …
.
.
.
 always @(a,b) begin
   x = a^b
 end
 always @(reset) begin
  x = 1’b0;
 end
endmodule
```

<u>Setting the same `reg` in multiple `always` blocks</u>

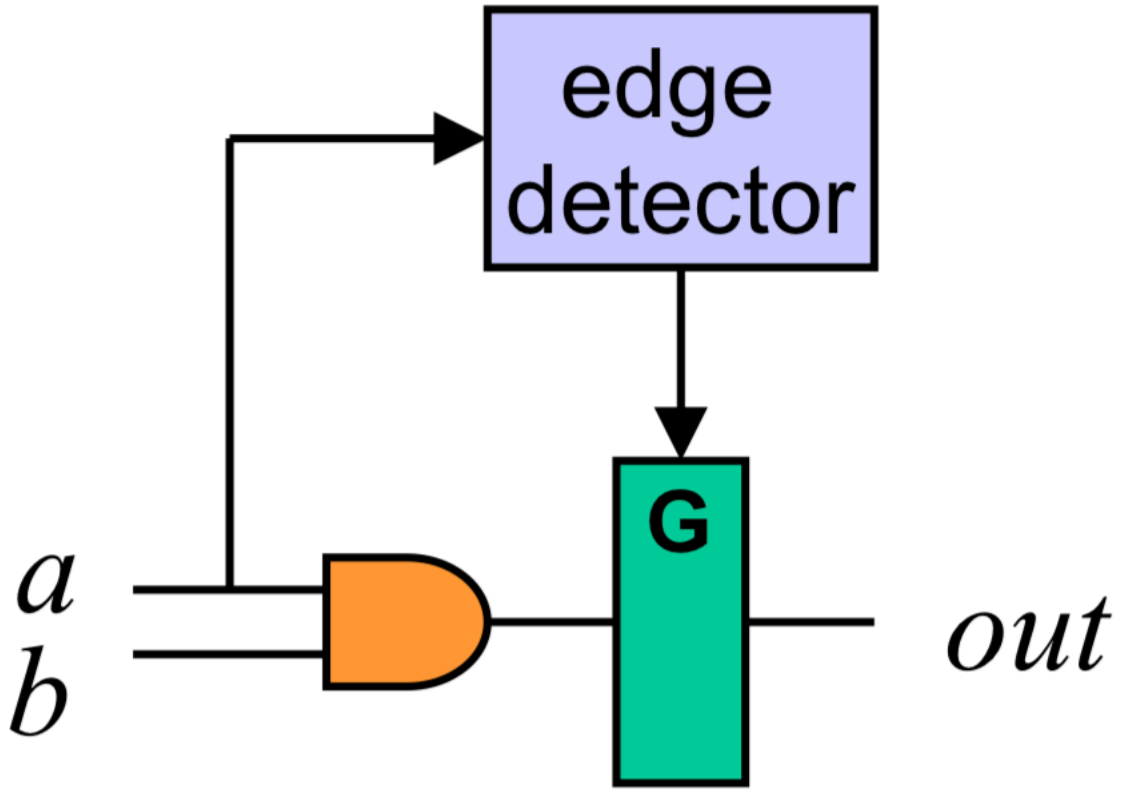
- Simulators will typically do what you tell them to do
- Synthesis tools will typically give a warning
- Never do this in Hardware verilog
- You might be able to get away with it in testing verilog but normally don’t do it

TYPICAL MISTAKES IN VERILOG



```
always @(a) begin // missing b
  out = a & b;
end
```

Result



Incomplete Sensitivity List

- `out` updates when `a` changes as expected
- `out` does not update when be `b` changes!
  - Put another way, `b` can change all it wants but `out` will not update - this requires a memory element to remember the last value of `out`
- Synthesis tool will give you a warning
- Using the construct:

  `always @(*)`

  eliminates this type of bug

TYPICAL MISTAKES IN VERILOG

```
wire a;
wire [3:0] b;
```

Why are the assignments below illegal?

```
assign b(4) = a;
```

Wrong Indexing

- Order (ascending or descending) is reversed
- Incorrect use of parenthesis
- Index values fall outside the actual range

LECTURE 3: OVERVIEW

- Typical mistakes in Verilog
- Data flow description
- Concurrent signal assignment + examples
- Assignment with delay + examples
- Concurrent code Verilog

DATA FLOW DESCRIPTION

- How data flows from input to output
- Circuit is designed with **concurrent** statements
  - The order of the statements is ***not*** important
  - The execution of the statements is event-based
  - Useful for combinational circuits modeling
- More realistic circuit modeling if either of the followings are available:
  - The Boolean equation
  - Truth table

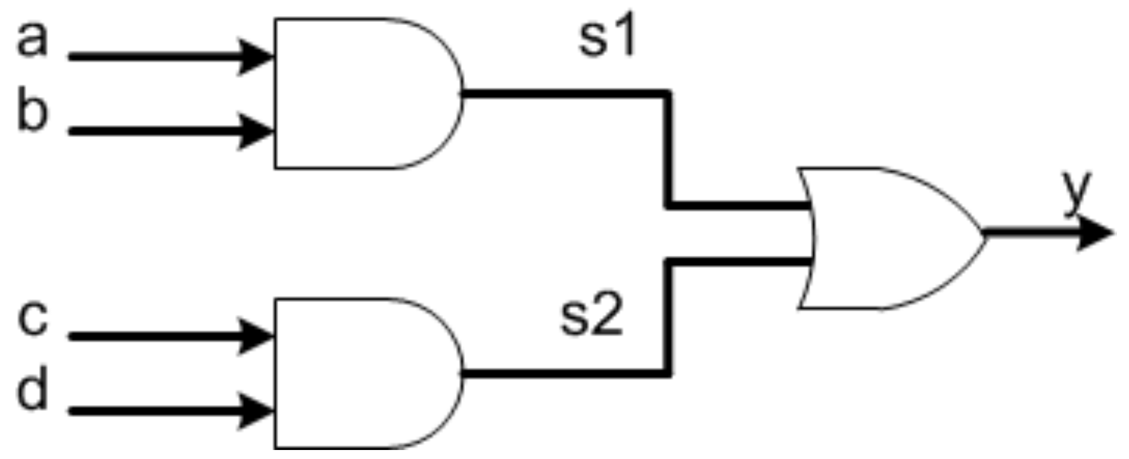
DATA FLOW DESCRIPTION

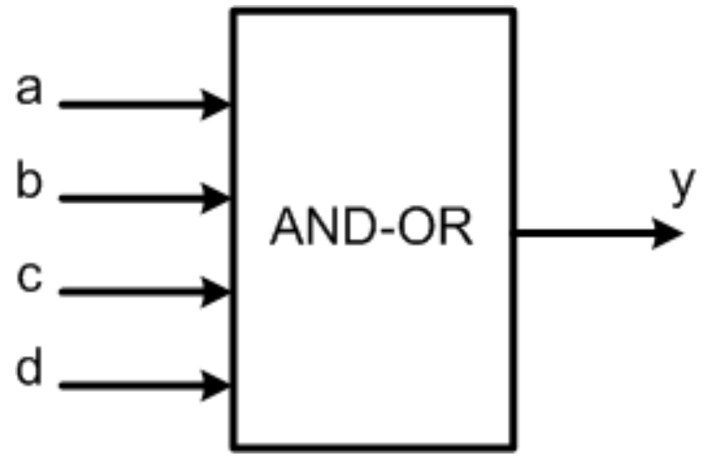
- All ports are defined as **wires** by default
    - output ports defined as reg to be used inside **sequential** blocks
- In concurrent assignment, **assign to a wire**

```
input a, b;
reg tmp;  // tmp should be defined as wire
assign temp = a & b; // syntax error
```

DATA FLOW DESCRIPTION - EXAMPLE 1

- AND-OR block
  - The circuit of the design





Input				Output
a	b	c	d	Y
1	1	-	-	1
-	-	1	1	1
1	1	1	1	1

  - A Boolean function of the output

➡`y = (ab) + (cd)`

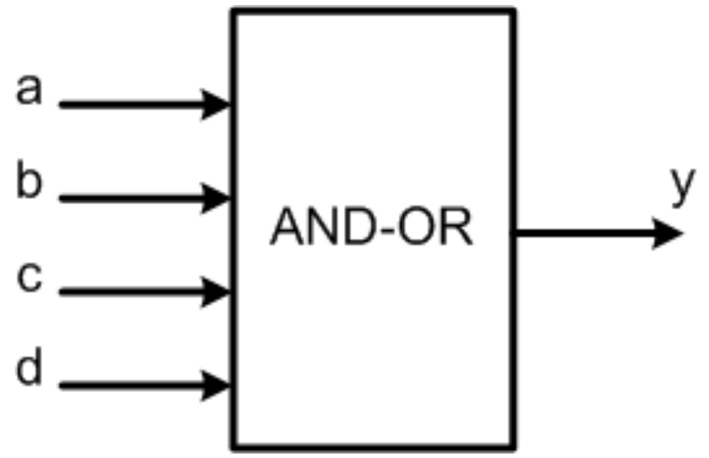
VERILOG CODE FOR AND-OR BLOCK

```
module and_or(a,b,c,d,y);
   input a,b,c,d;
   output y;

 wire s1,s2;
 assign s1 = a & b;
 assign s2 = c & d;
 assign y  = s1 | s2;
 endmodule
```

Wire statement here is not necessarily needed since s1 and s2 are single bit

Runs concurrently



Input				Output
a	b	c	d	Y
1	1	-	-	1
-	-	1	1	1
1	1	1	1	1

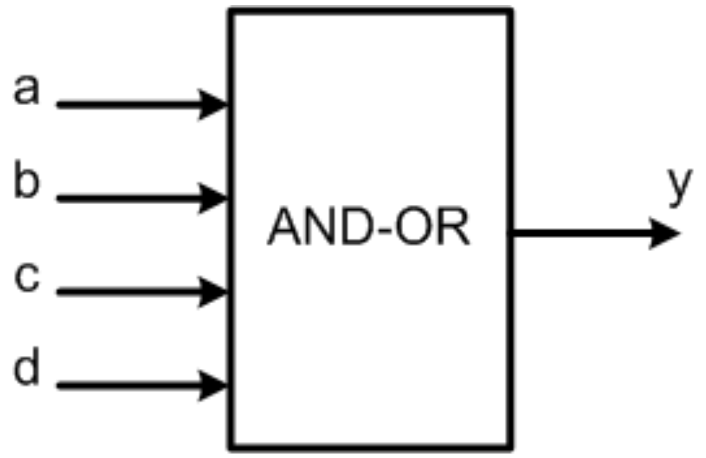
UCI University of California, Irvine

VERILOG CODE FOR AND-OR BLOCK

```
module and_or(a,b,c,d,y);
   input a,b,c,d;
   output y;

 wire s1,s2;
 assign s1 = a & b;
 assign s2 = c & d;
 assign y  = s1 | s2;
 endmodule
```

Runs concurrently



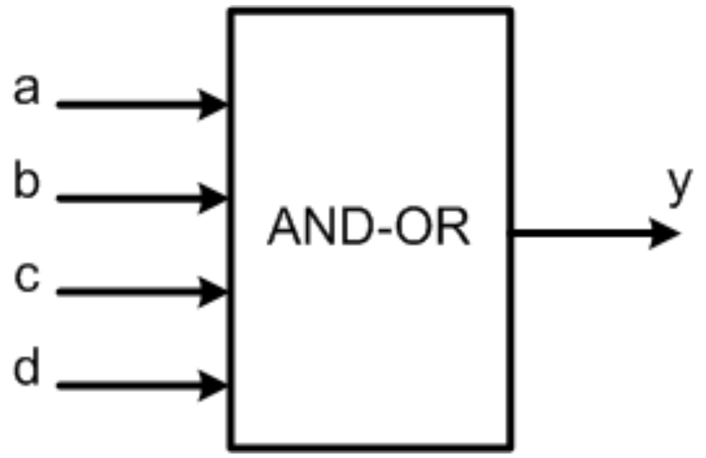
Input				Output
a	b	c	d	Y
1	1	-	-	1
-	-	1	1	1
1	1	1	1	1

VERILOG CODE FOR AND-OR BLOCK

```
module and_or(a,b,c,d,y);
   input a,b,c,d;
   output y;

 wire s1,s2;
 assign y  = s1 | s2;
 assign s2 = c & d;
 assign s1 = a & b;
 endmodule
```

Runs concurrently



Input				Output
a	b	c	d	Y
1	1	-	-	1
-	-	1	1	1
1	1	1	1	1

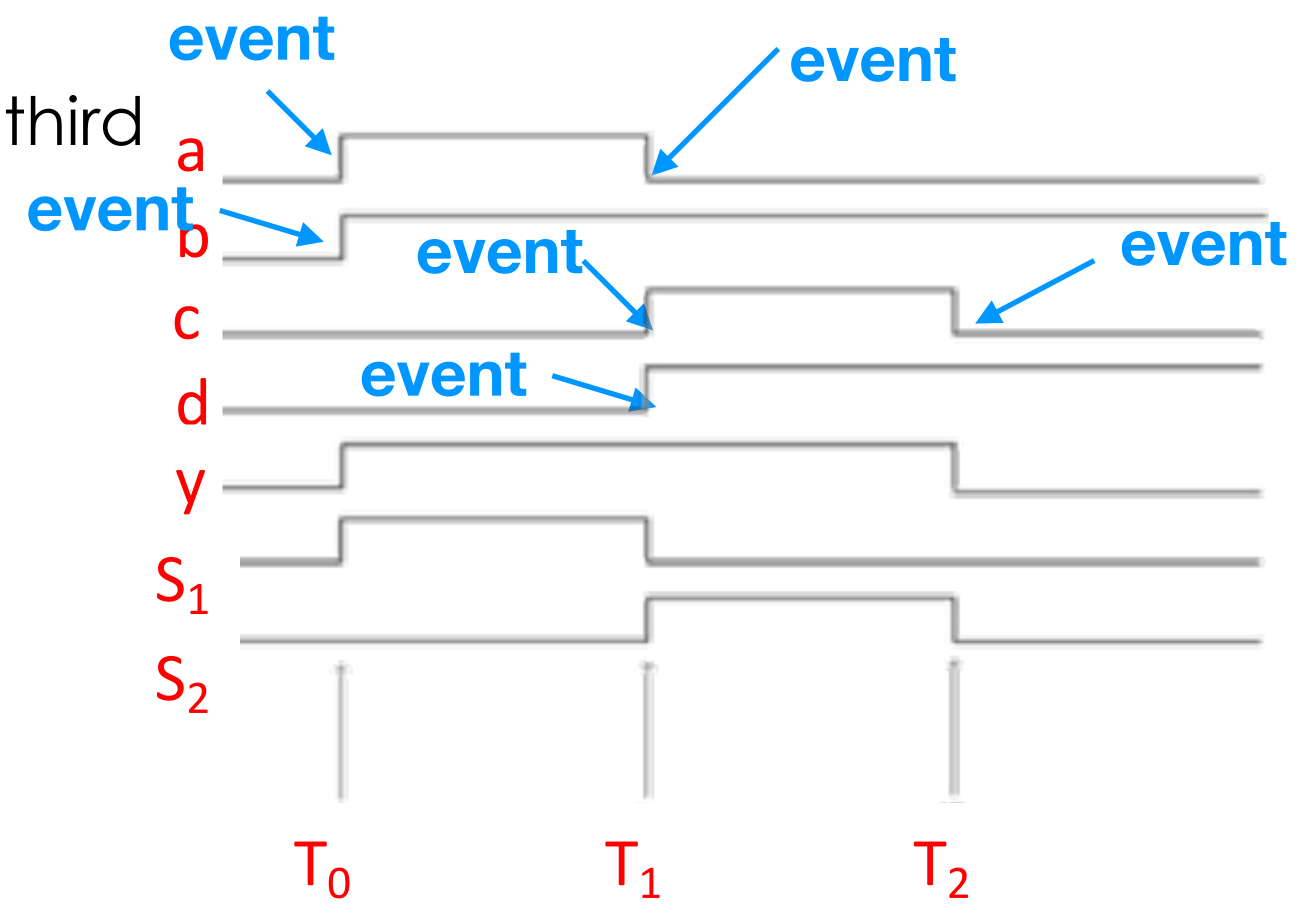
CONCURRENT SIGNAL ASSIGNMENT (CONT'D)

- Signal assignment execution
  - Different from language such as C
  - An event should occur on the right side
    - Event is a change of value of right side
      - For example, from 0 to 1 or even 0 to Z
    - The statement that receives an event is executed first
    - More than one statement can have event at the same time
      - executed concurrently
- Signal statement phases
  - Calculation
    - At the time of event happens
  - Assignment
    - After calculation is done
- ***assign*** is the keyword

SIMULATION OF AND-OR

- Event at T_0
  - On a and b; statement 1 is executed.
  - After executing first statement the output s1 is updated
    - No delay; there is an event on the third statement at T_0
    - So y also changes at T0
- Event at T_1
  - On a, c and d
  - Statement 1 and 2 executed first
  - No delay; statement 3 executed
  - S_1 and S_2 are updated
- Event at T_2
  - On c; statement 2 executed

```
assign s1 = a & b;
assign s2 = c & d;
assign y  = s1 | s2;
```



LECTURE 3: OVERVIEW

- Typical mistakes in Verilog
- Data flow description
- Concurrent signal assignment + examples
- Assignment with delay + examples
- Concurrent code Verilog

CONSTANT DECLARATION

- Has same usage as other languages
  - Declared by its type such as integer or time
  - Use = to assign it a value
  - the unit is simulation screen unit

```
time period =100; //verilog
parameter period = 100;
`define period  100;
```

- More data types:
  - **integer** - 4-state Verilog data type, 32-bit signed integer
  - **time** - 4-state Verilog data type, 64-bit unsigned integer

- Delay assignment
  - Use #

```
assign #10 S1 = a & b;
```

AND-OR BLOCK WITH DELAY

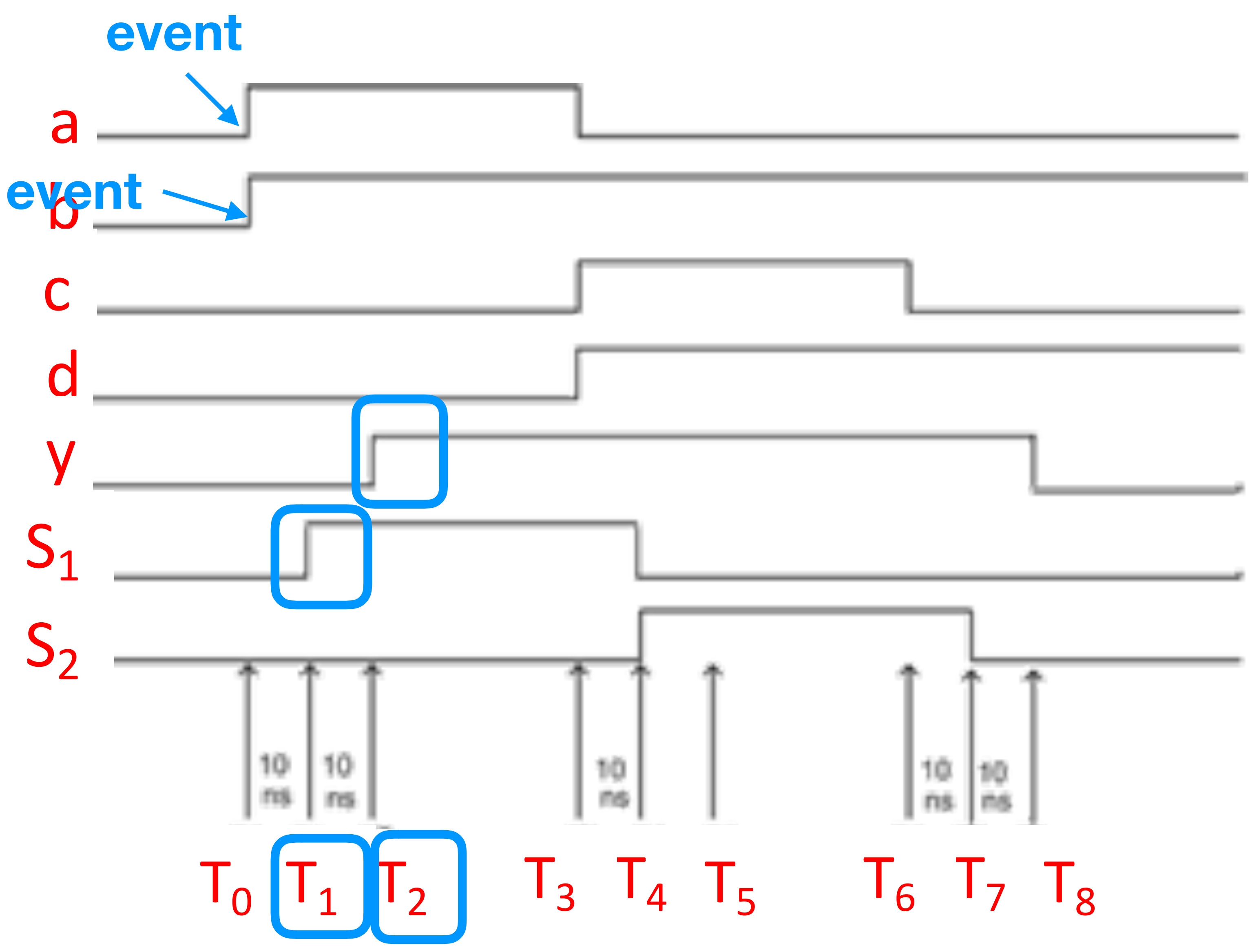
```
module and_or(a,b,c,d,y);
   input a,b,c,d;
   output y;

 wire s1,s2;
 time dly = 10;
 assign #dly s1 = a & b;
 assign #dly s2 = c & d;
 assign #dly y  = s1 | s2;
 endmodule
```

SIMULATION OF AND-OR BLOCK WITH DELAY

- Event at T_0
  - On a and b; statement 1 is executed
  - After dly (at T_1) S_1 is updated;
  - Statement 3 is executed
  - After dly (at T_2) y is updated

```
assign #dly s1 = a & b;
assign #dly s2 = c & d;
assign #dly y  = s1 | s2
```

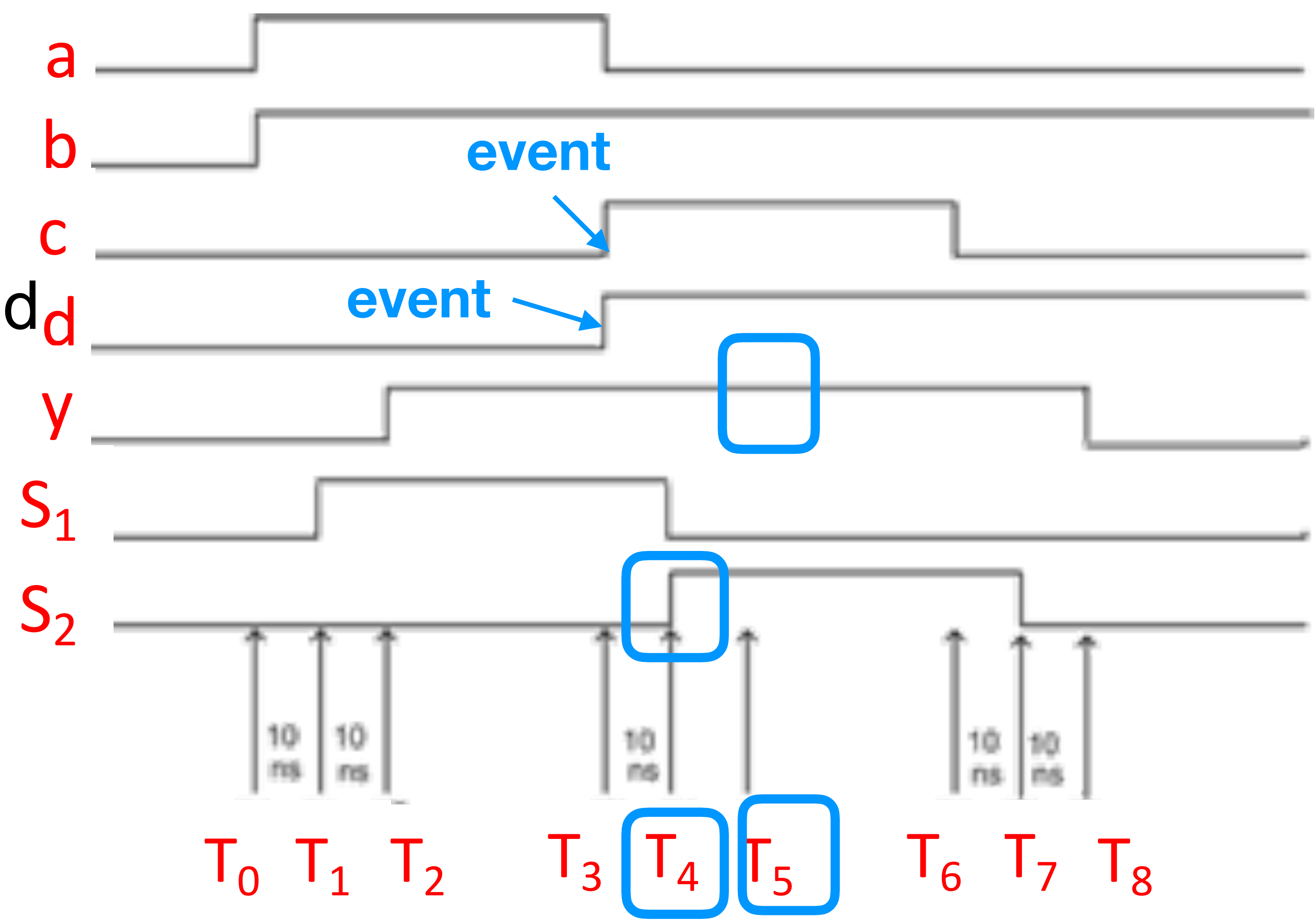


SIMULATION OF AND-OR BLOCK WITH DELAY

- Event at T_0
  - On a and b; statement 1 is executed
  - After dly (at T_1) S_1 is updated;
  - Statement 3 is executed
  - After dly (at T_2) y is updated

```
assign #dly s1 = a & b;
assign #dly s2 = c & d;
assign #dly y  = s1 | s2
```

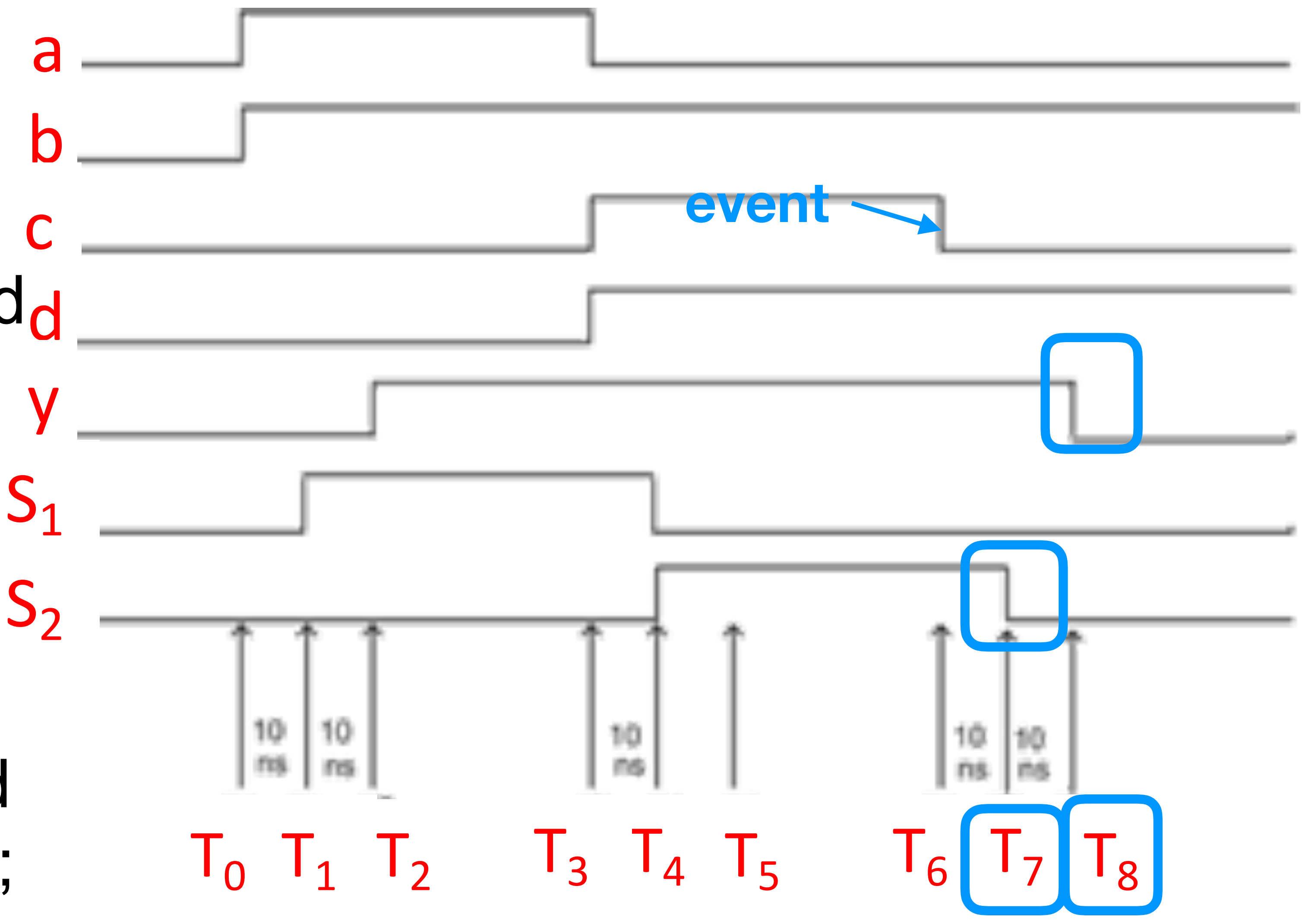
- Event at T_3
  - On a, c and d;
  - Statement 1 & 2 are executed
  - At T_4, S_1 & S_2 are updated
  - Statement 3 is executed
  - At T_5, y is updated



SIMULATION OF AND-OR BLOCK WITH DELAY

```
assign #dly s1 = a & b;
assign #dly s2 = c & d;
assign #dly y  = s1 | s2
```

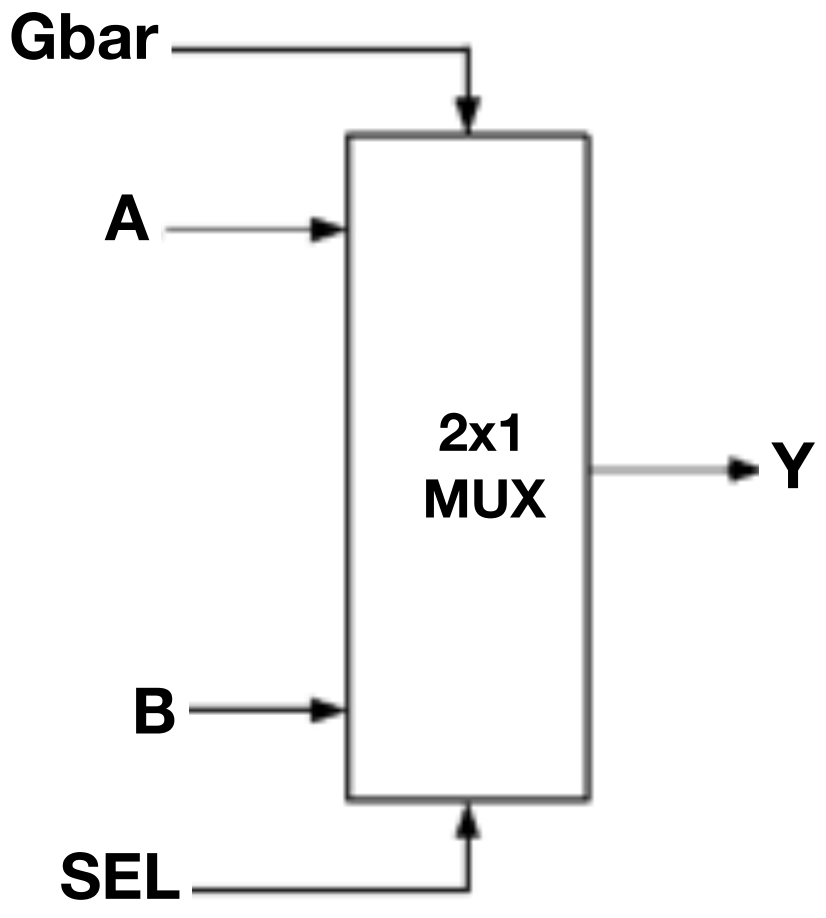
- Event at T_0
  - On a and b; statement 1 is executed
  - After dly (at T_1) S_1 is updated;
  - Statement 3 is executed
  - After dly (at T_2) y is updated
- Event at T_3
  - On a, c and d;
  - Statement 1 & 2 are executed
  - At T_4, S_1 & S_2 are updated
  - Statement 3 is executed
  - At T_5, y is updated
- Event at T_6
  - on c; statement 2 is executed
  - after dly (at T_7) S_1 is updated;
  - statement 3 is executed
  - after dly (at T_8) y is updated

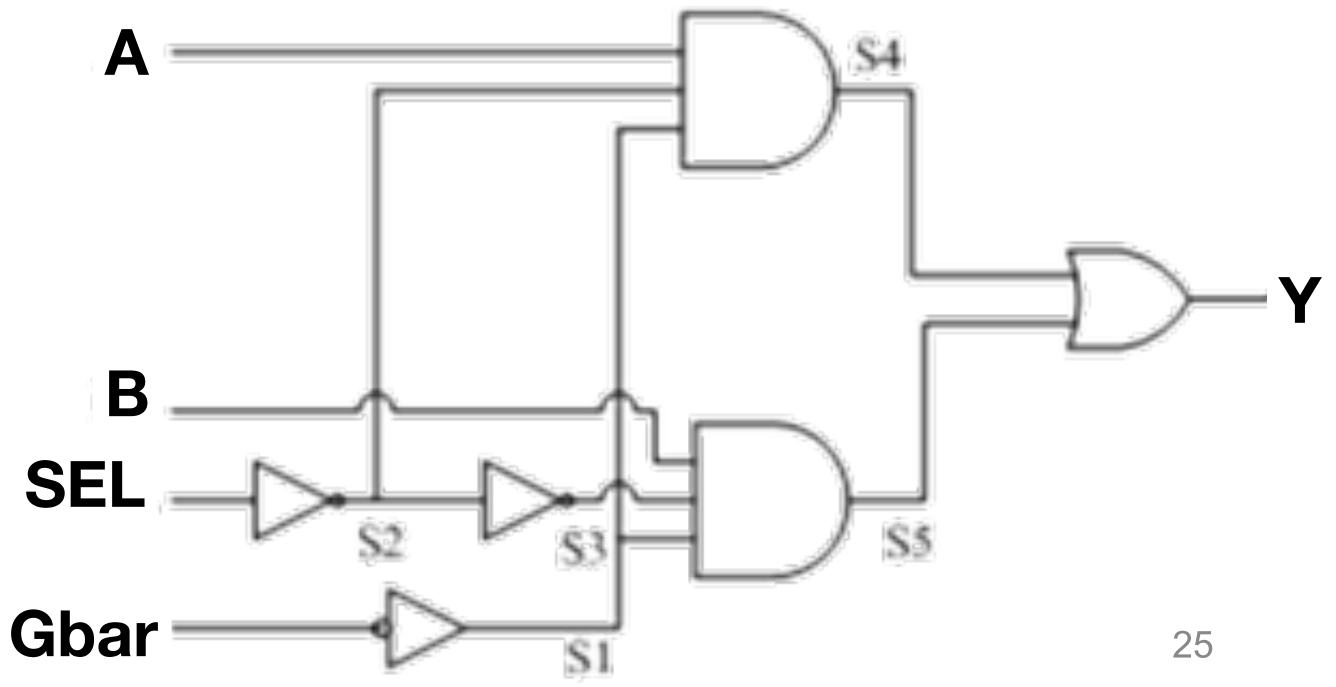


DATA FLOW DESCRIPTION OF MUX 2X1

- 2x1 multiplexer
  - with active low enable
  - with constant delay
    - All gate delays are 7 ns

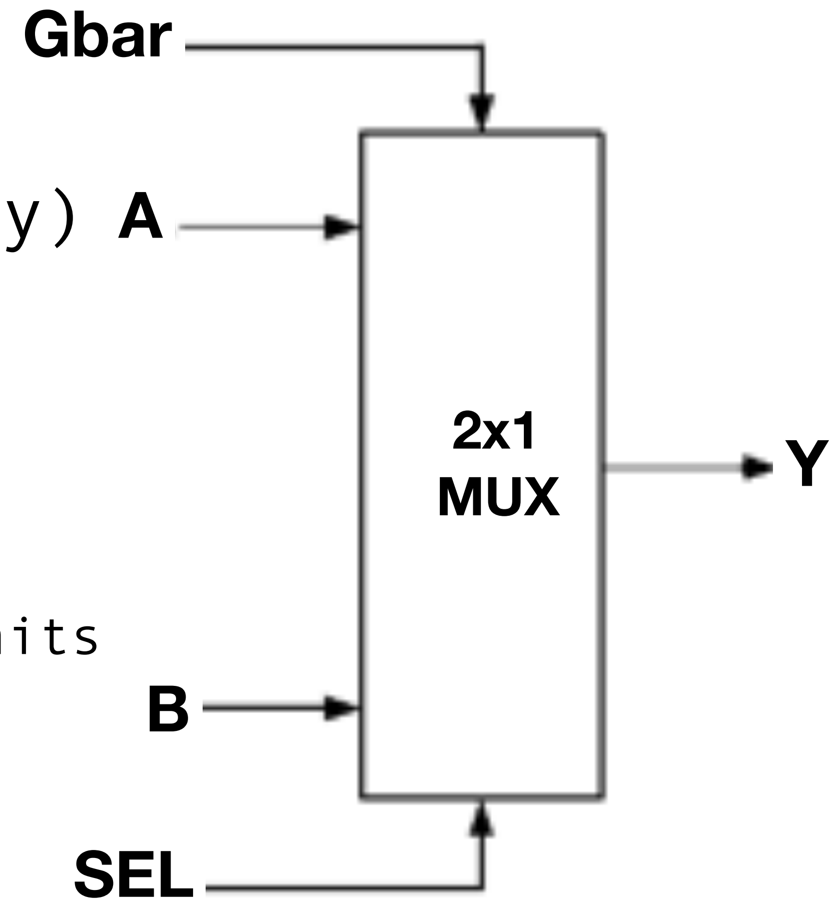
Input		Output
SEL	Gbar	Y
X	1	0
0	0	A
1	0	B

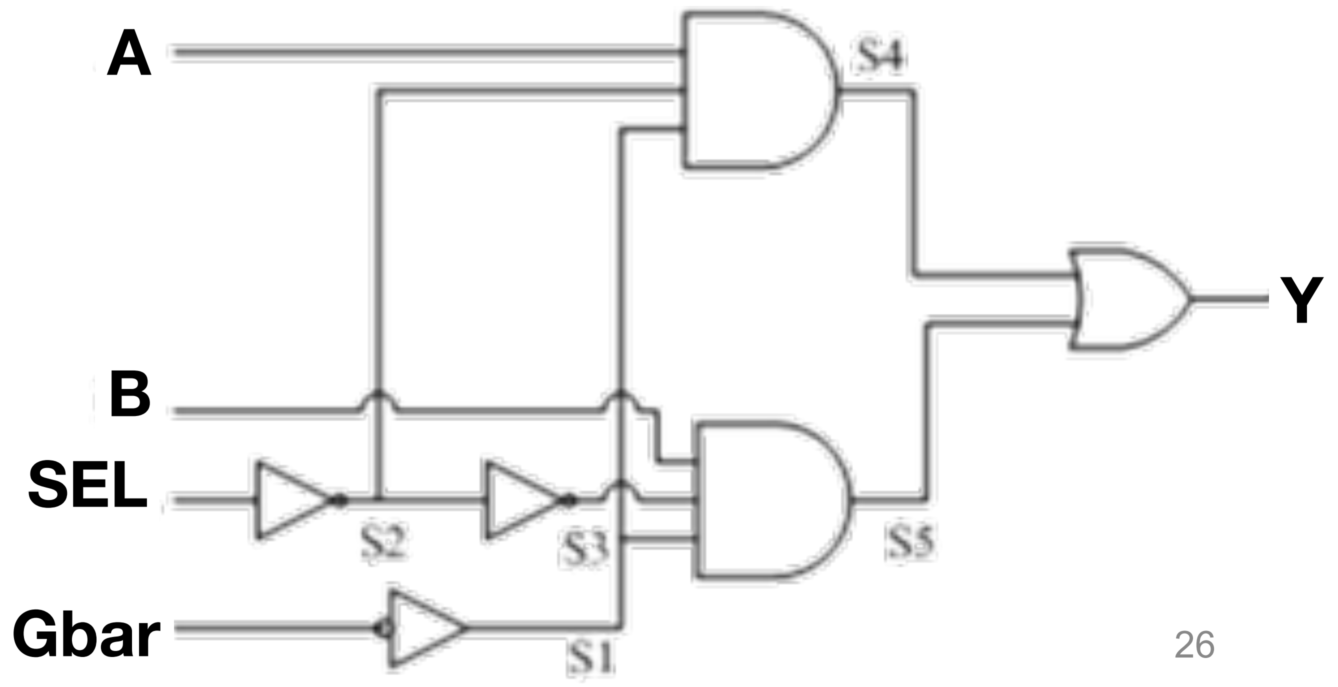




MUX 2X1- WITH DELAY CODE

```
module mux_enable(a,b,sel,gbar,y)
   input a,b,sel,gbar;
   output y;
 wire s1,s2,s3,s4,s5;
 time  dly = 7;//simulation screen units
 assign #dly  y=s4|s5;
 assign #dly  s4=a & s2 & s1;
 assign #dly  s5=b & s3 & s1;
 assign #dly  s2= ~sel;
 assign #dly  s3= ~s2;
 assign #dly  s1= ~gbar;
 endmodule
```

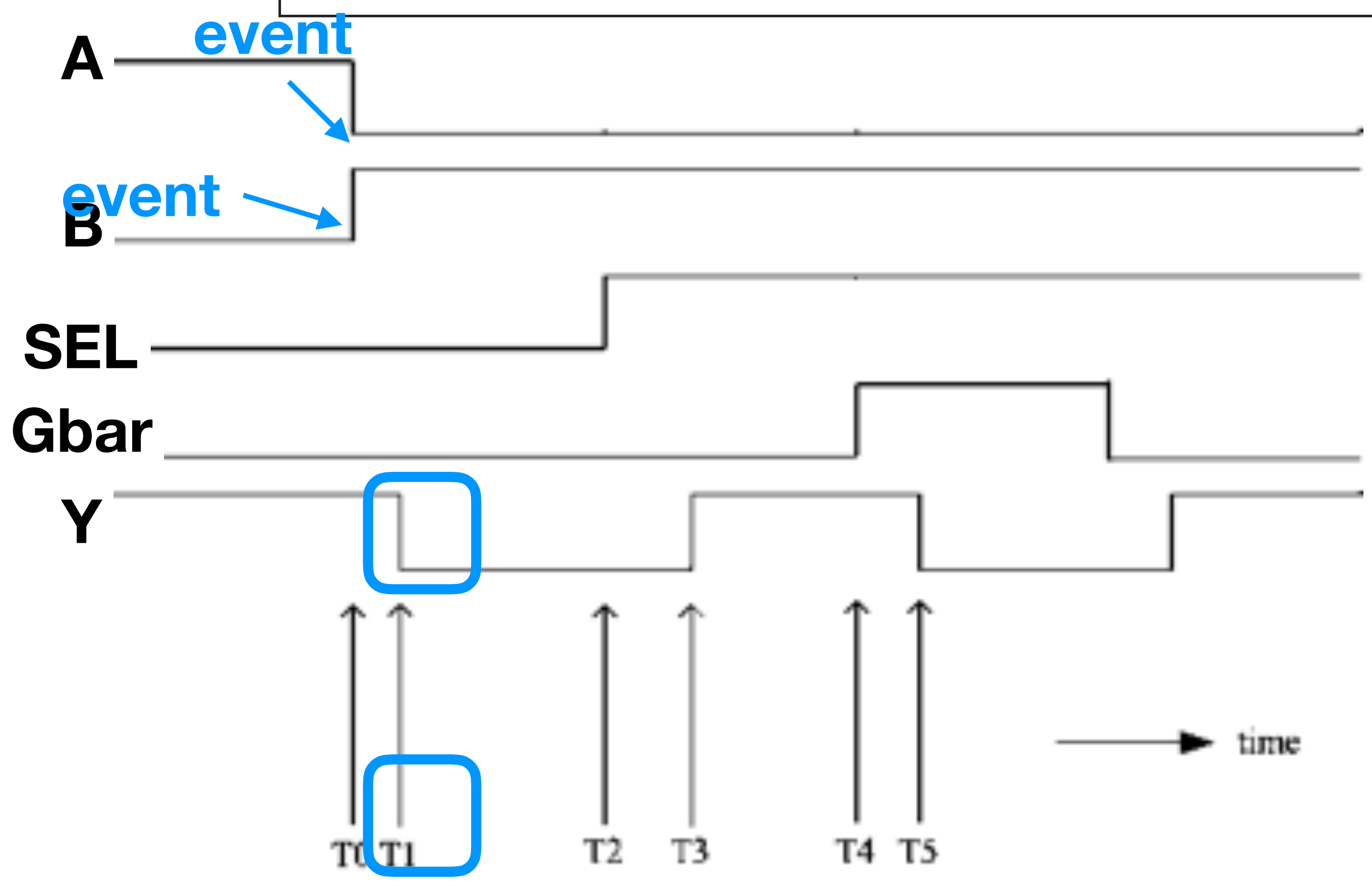




SIMULATION OF MUX 2X1

- If T_0 =100ns
  - T_1 = 114ns, why?

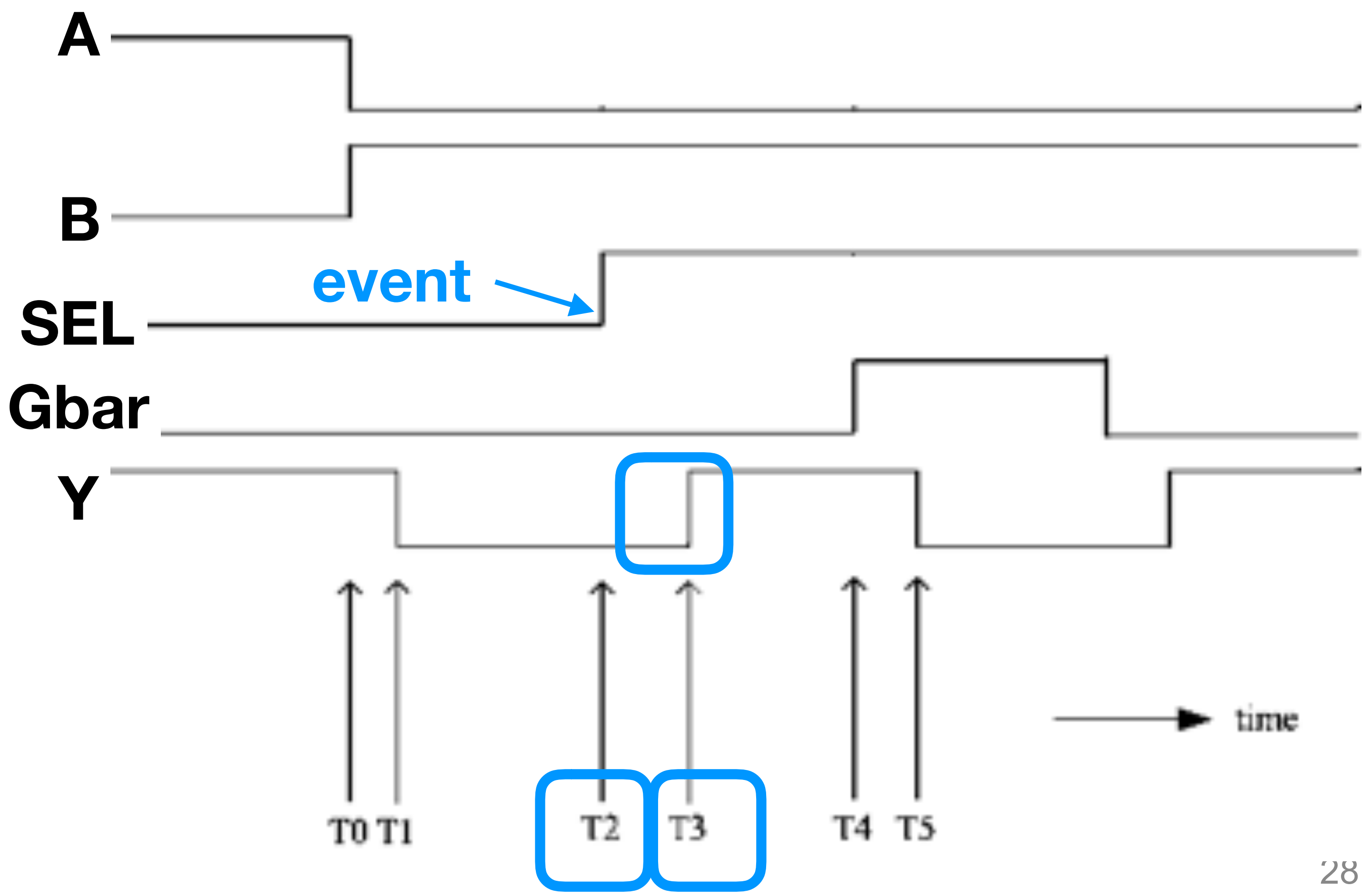
```
assign #dly  y=s4|s5;
assign #dly  s4=a & s2 & s1;
assign #dly  s5=b & s3 & s1;
assign #dly  s2= ~sel;
assign #dly  s3= ~s2;
assign #dly  s1= ~gbar;
```



SIMULATION OF MUX 2X1

- If T_0 =100ns
  - T_1 = 114ns, why?
- If T_2 =200ns
  - T_3 = 228ns, why?

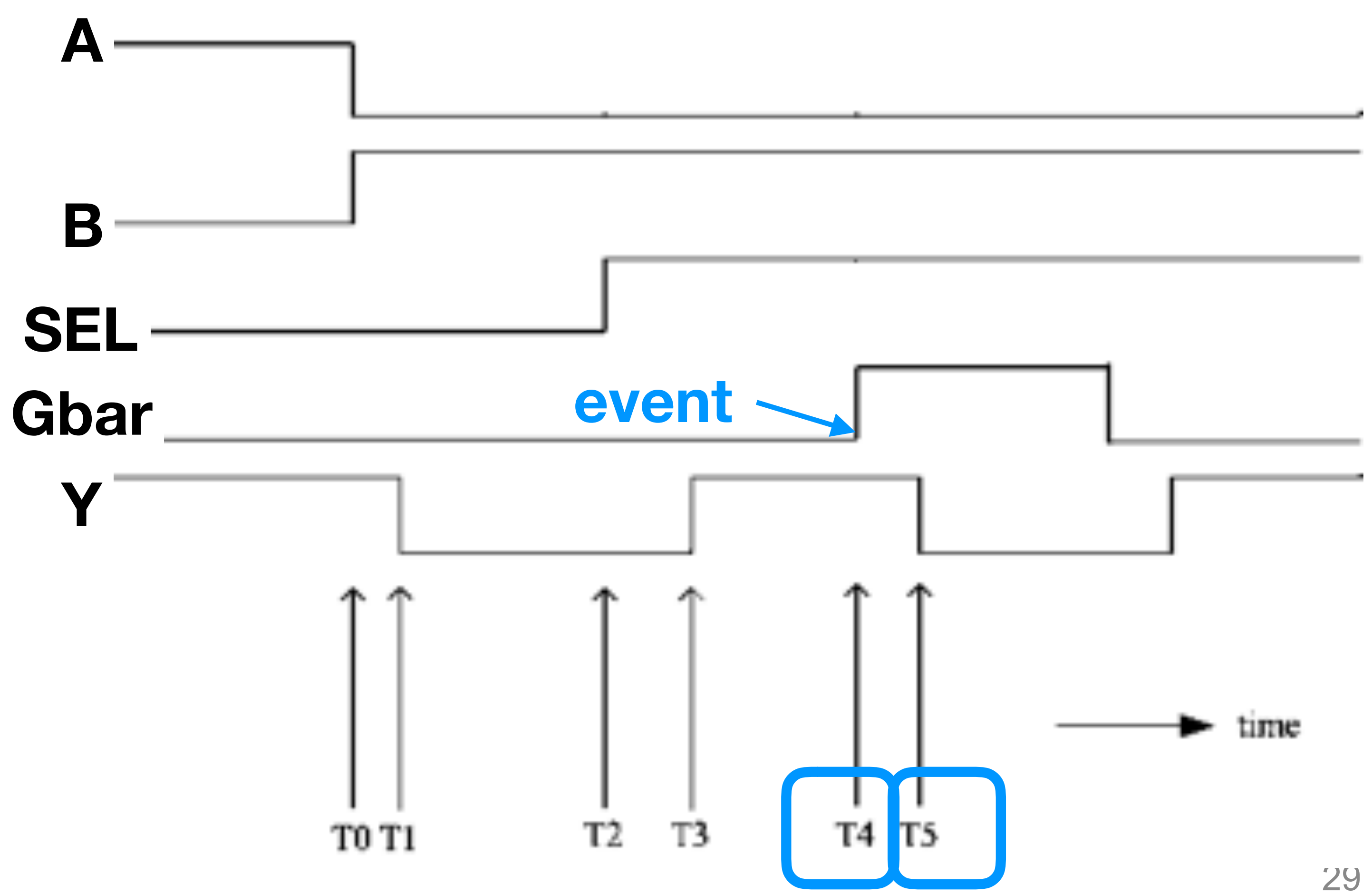
```
assign #dly  y=s4|s5;
assign #dly  s4=a & s2 & s1;
assign #dly  s5=b & s3 & s1;
assign #dly  s2= ~sel;
assign #dly  s3= ~s2;
assign #dly  s1= ~gbar;
```



SIMULATION OF MUX 2X1

- If T_0 =100ns
  - T_1 = 114ns, why?
- If T_2 =200ns
  - T_3 = 228ns, why?
- If T_4 =300ns
  - T_5 = 321ns, why?

```
assign #dly  y=s4|s5;
assign #dly  s4=a & s2 & s1;
assign #dly  s5=b & s3 & s1;
assign #dly  s2= ~sel;
assign #dly  s3= ~s2;
assign #dly  s1= ~gbar;
```



LECTURE 3: OVERVIEW

- Typical mistakes in Verilog
- Data flow description
- Concurrent signal assignment + examples
- Assignment with delay + examples
- Concurrent code Verilog

CONCURRENT IMPLEMENTATION

- Purely Concurrent Statement Syntax
  - assign
    - `assign out = in_a & in_b;`
  - ?:
    -
```
mux =  (addr==2'b00)   ?  i0 :
      ((addr == 2'b01) ?  i1 :
      ((addr == 2'b10) ?  i2 :
      ((addr == 2'b11) ?  i3 :
                          4'bz)));
```
  - generate

CONCURRENT IMPLEMENTATION: GENERATE

- **Syntax**

```
genvar i;
generate
for (i=0; i< MAX; i=i+1) begin
    [statements]
end
endgenerate
```

UCI University of California, Irvine

CONCURRENT IMPLEMENTATION: GENERATE

- Example on loop

```
input [7:0] x;
input [15:0] y;
output [7:0] z;

genvar i;
generate
  for (i=0; i<=7; i=i+1) begin
     assign z[i] = x[i] & y[i+8];
  end
endgenerate
```

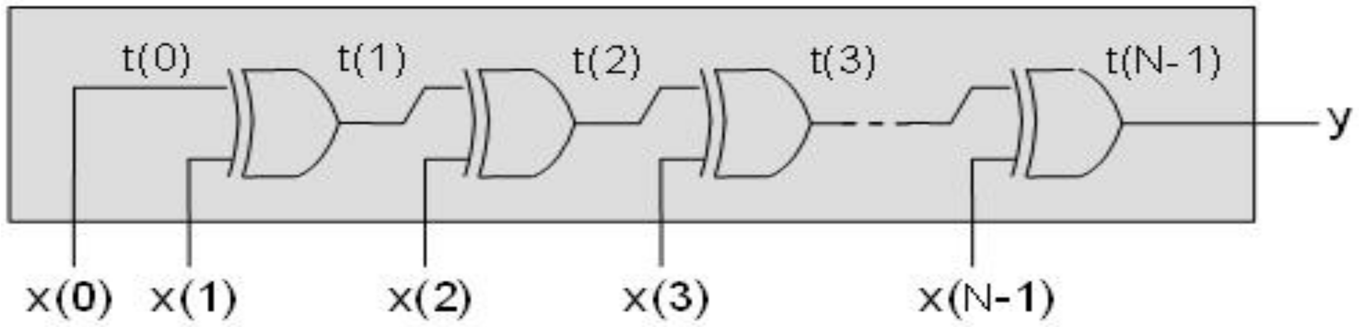
```
/*  Equivalent Example  */

assign z[0] = x[0] & y[0+8];
assign z[1] = x[1] & y[1+8];
assign z[2] = x[2] & y[2+8];
assign z[3] = x[3] & y[3+8];
assign z[4] = x[4] & y[4+8];
assign z[5] = x[5] & y[5+8];
assign z[6] = x[6] & y[6+8];
assign z[7] = x[7] & y[7+8];
```

UCI University of California, Irvine

CONCURRENT IMPLEMENTATION: PARITY DETECTOR

- XOR Tree

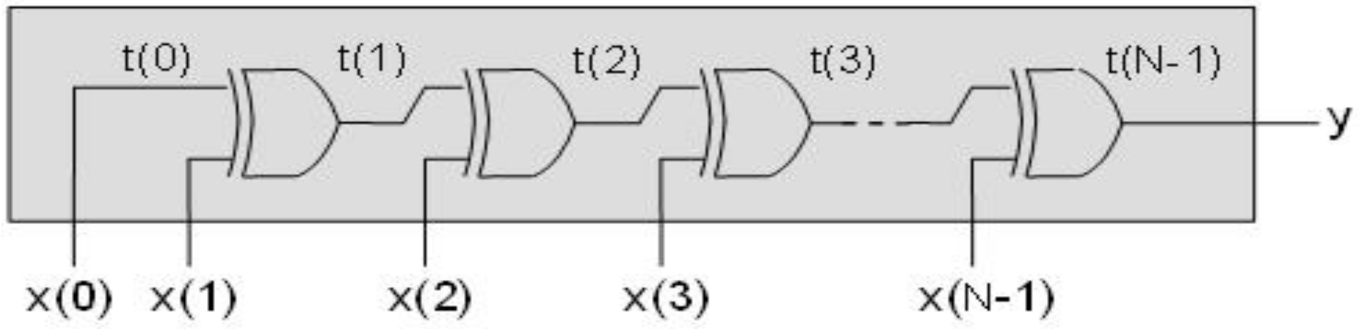


$$y = x(0) \oplus x(1) \oplus x(2) \oplus \ldots \oplus x(N-1)$$

```
module xor_tree( input [7:0]x, output y );
 wire [6:0] temp;
 assign temp[0] = x[0];
 assign temp[1] = temp[0] ^ x[1];
 assign temp[2] = temp[1] ^ x[2];
 assign temp[3] = temp[2] ^ x[3];
 assign temp[4] = temp[3] ^ x[4];
 assign temp[5] = temp[4] ^ x[5];
 assign temp[6] = temp[5] ^ x[6];
 assign y = temp[6] ^ x[7];
endmodule
```

CONCURRENT IMPLEMENTATION: PARITY DETECTOR

- **Does this work?**
- **NO, why?**

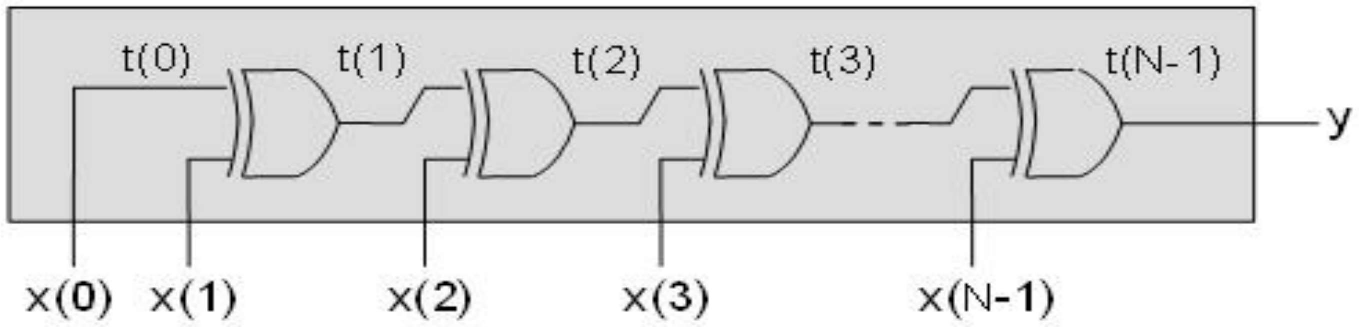


$y = x(0) \oplus x(1) \oplus x(2) \oplus \ldots \oplus x(N-1)$

```
module xor_tree( input [7:0]x, output y );
 wire [6:0] temp;
 genvar i;
 generate
   for (i=1; i<7; i=i+1) begin
      assign temp[i] = temp[i-1] ^ x[i];
   end
  endgenerate
  assign y = temp[7];
endmodule
```

CONCURRENT IMPLEMENTATION: PARITY DETECTOR

- **Does this work?**
- **YES**



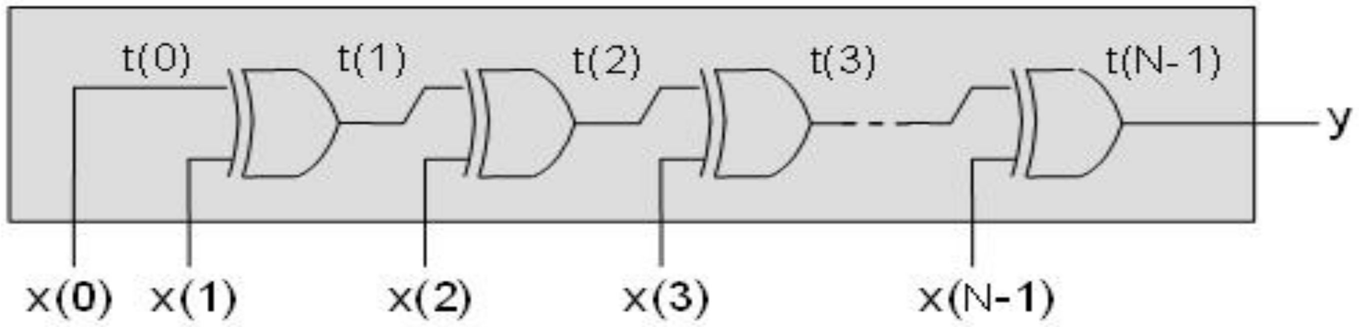
$$y = x(0) \oplus x(1) \oplus x(2) \oplus \ldots \oplus x(N-1)$$

```
module xor_tree( input [7:0]x , output y );
 wire [6:0] temp;
 temp[0] = x[0];
 genvar i;
 generate
   for (i=1; i<7; i=i+1) begin
      assign temp[i] = temp[i-1] ^ x[i];
   end
  endgenerate
  assign y = temp[6] ^ x[7];
endmodule
```

UCI University of California, Irvine

CONCURRENT IMPLEMENTATION: PARITY DETECTOR

- Number of 1's in a signal
  - Odd then PD = 1
  - Even then PD = 0



$$y = x(0) \oplus x(1) \oplus x(2) \oplus \ldots \oplus x(N-1)$$

- Usage
  - Error detection methods
    - Even parity
    - Odd parity