ECE 4514
Digital Design II
Spring 2007

Lecture 4: Gate Level Modeling

Patrick Schaumont

Gate-level Modeling

- In this lecture we focus on modeling and simulation of gate networks
- We will use structural modeling techniques
- We will use delay modeling to estimate the delay of a circuit
- We will discuss several examples:
  - bit comparator
  - byte comparator
  - latch
  - master/slave flip-flop

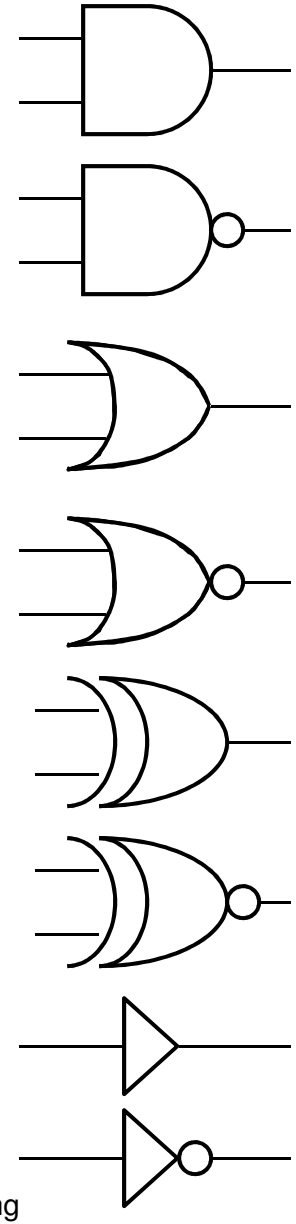
Outline

- Logic Primitive Gates
- Instantiation, Fanout, Fanin, Arrays of gates
- Truth Tables (X, Z)
- Delay Models
  - Signal propagation in gates
  - Delay modeling
- Example Models:
  - Bit-comparator, Word-comparator
  - Latches and Flip-flops

Primitive Gates

- A *primitive* is a module which you do not have to describe as the simulator already knows it
- There are 14 different primitive gates in Verilog
  - 8 of them are for logic functions (focus of this class)
  - 4 of them are for modeling tri-state signals
  - 2 of them are enance signal strength
- New primitive gates can be added
  - User Defined Primitives (Chapter 12 Palnitkar)

Standard Logic Functions



and(out, in1, in2)

nand(out, in1, in2)

or(out, in1, in2)

nor(out, in1, in2)

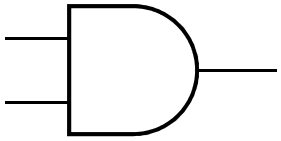
xor(out, in1, in2)

xnor(out, in1, in2)

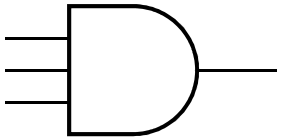
buf(out, in)

not(out, in)

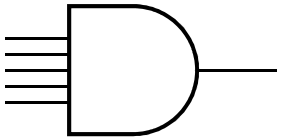
More than two inputs, still a primitive



and(out, in1, in2)



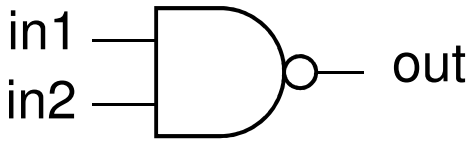
and(out, in1, in2, in3)



and(out, in1, in2, in3, in4, in5)

NAND Truth Table

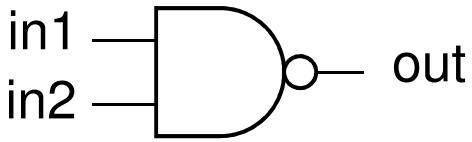
- Must deal with 'Z' and 'X' input (4-valued logic)
- Z is undefined (floating), X is unknown



in1 \ in2	0	1	X	Z
0	1	1		
1	1	0		
X				
Z				

NAND Truth Table

- Must deal with 'Z' and 'X' input (4-valued logic)
- Z is undefined (floating), X is unknown

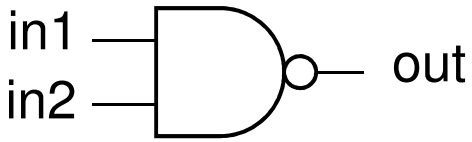


in1 \ in2	0	1	X	Z
0	1	1	**1**	**1**
1	1	0		
X	**1**			
Z	**1**			

With any input '0', ouput MUST be '1'.

NAND Truth Table

- Must deal with 'Z' and 'X' input (4-valued logic)
- Z is undefined (floating), X is unknown

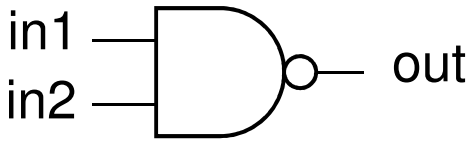


in1 \ in2	0	1	X	Z
0	1	1	1	1
1	1	0	**X**	**X**
X	1	**X**		
Z	1	**X**		

With an input '1', the ouput may be either '1' or '0',
(we don't know unless we know the value of the other input)

NAND Truth Table

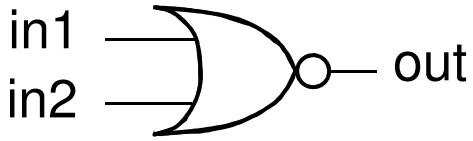
- Must deal with 'Z' and 'X' input (4-valued logic)
- Z is undefined (floating), X is unknown



in1 \ in2	0	1	X	Z
0	1	1	1	1
1	1	0	X	X
X	1	X	**X**	**X**
Z	1	X	**X**	**X**

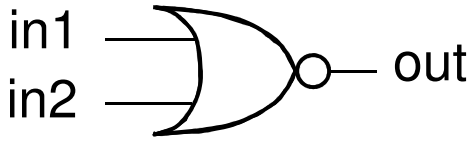
With an input unknown, we have no clue what the output may be (even if we know the other input)

NOR Truth Table



in1 \ in2	0	1	X	Z
0	1	0		
1	0	0		
X				
Z				

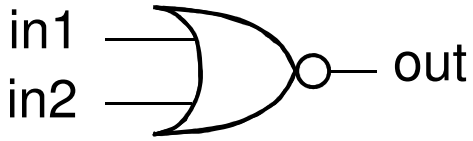
NOR Truth Table



in1 \ in2	0	1	X	Z
0	1	0		
1	0	0	**0**	**0**
X		**0**		
Z		**0**		

If an input is '1', the output MUST be 0.

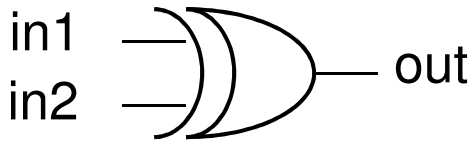
NOR Truth Table



in1 \ in2	0	1	X	Z
0	1	0	X	X
1	0	0	0	0
X	X	0	X	X
Z	X	0	X	X

In all other cases, we don't know.

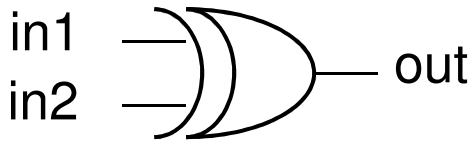
XOR Truth Table



in1 \ in2	0	1	X	Z
0	0	1	X	X
1	1	0	X	X
X	X	X		X
Z	X	X	X	

What should go here ? 0 or X ?

XOR Truth Table

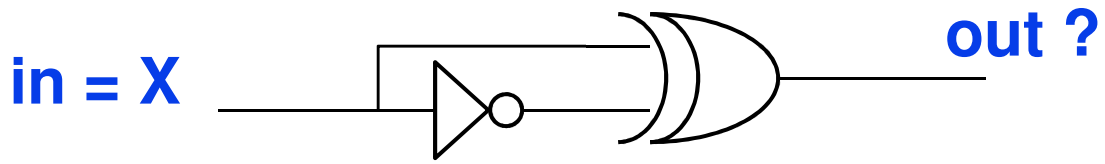


in1 \ in2	0	1	X	Z
0	0	1	X	X
1	1	0	X	X
X	X	X	X	X
Z	X	X	X	X

The simulator says 'X'. Shouldn't this be '0'?
After all, (1 XOR 1) = 0 and (0 XOR 0) = 0 ..

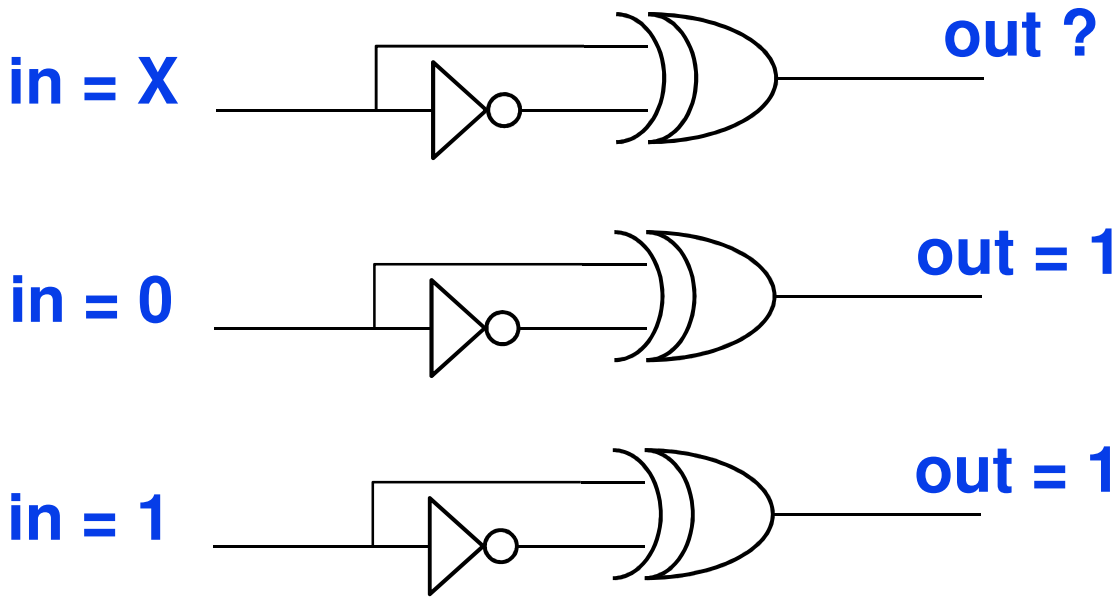
What would Verilog do?

- If you answered 'it should be X', then what is the output of the following circuit ?



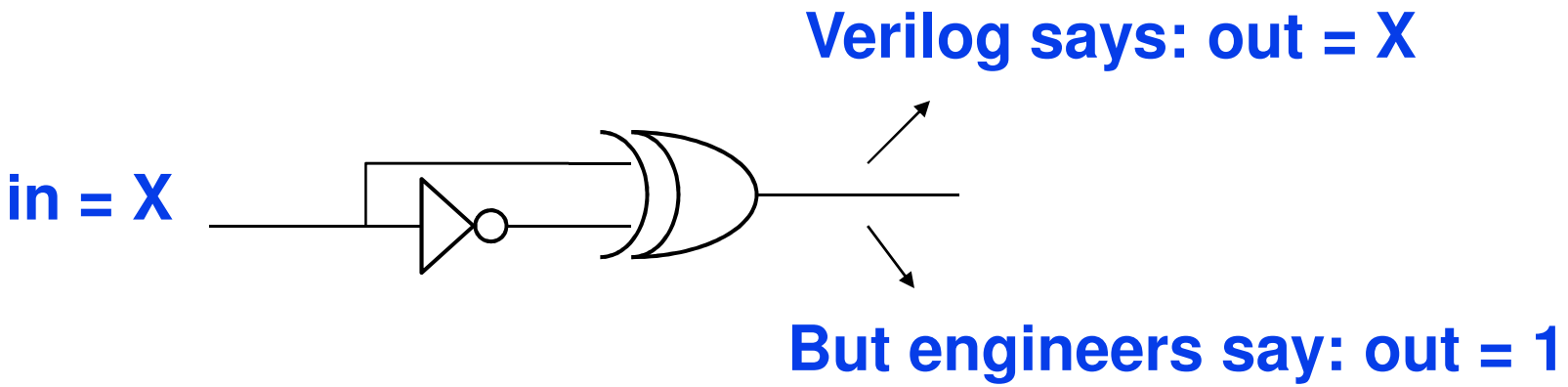
What would Verilog do?

- If you answered 'it should be X', then what is the output of the following circuit ?



What would Verilog do?

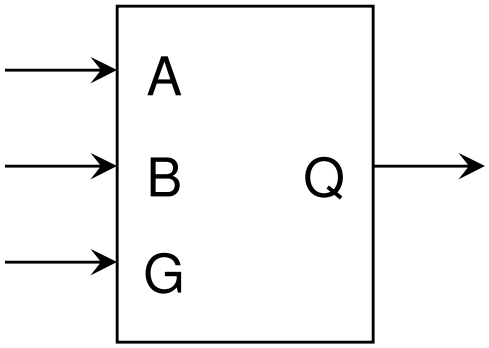
- If you answered 'it should be X', then what is the output of the following circuit ?



4-valued logic is an abstraction, sometimes it may be more abstract then what you had intended ...

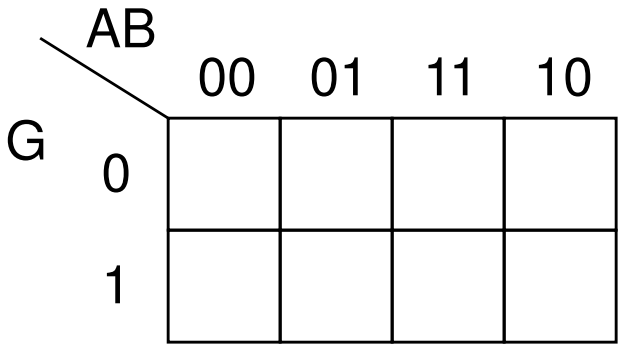
Gate Instantiation: Bit Comparator Example

- Bit comparator: Q indicates if A > B
  G is a cascading input



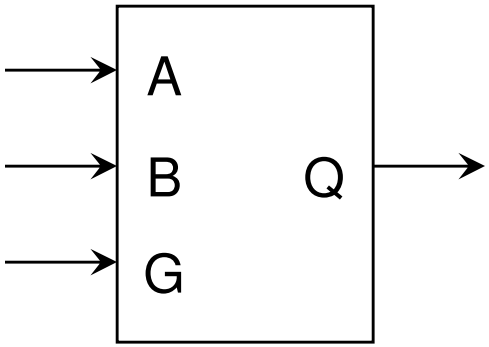
Pseudocode:

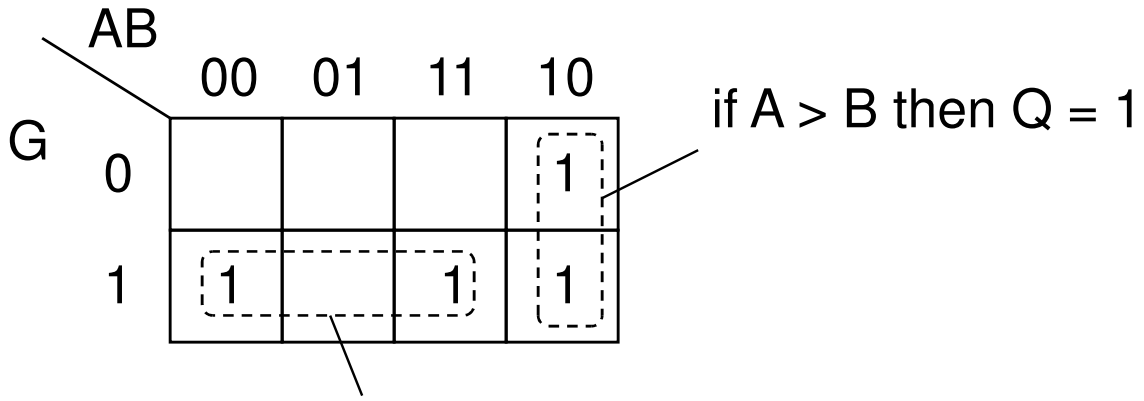
```
if (A = 1) and (B = 0)
    then Q is 1
else if (A=B) and (G = 1)
    then Q is 1
else Q is 0
```



Gate Instantiation: Bit Comparator Example

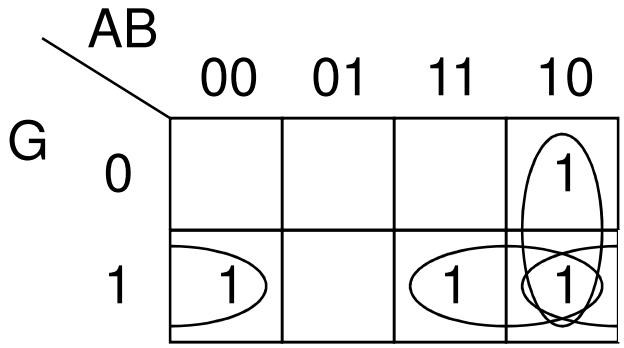
- Bit comparator: Q indicates if A > B
  G is a cascading input



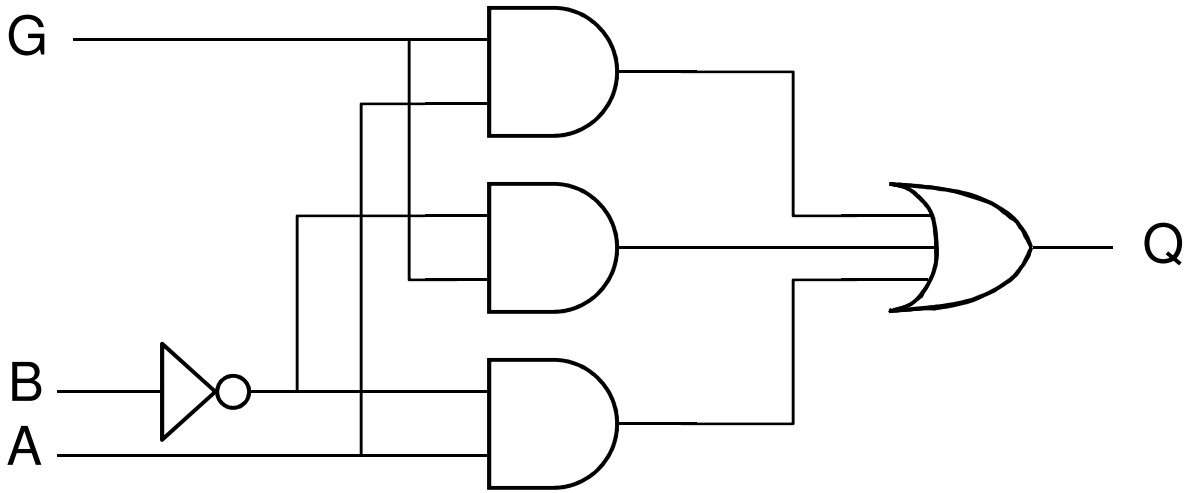


Gate Instantiation: Bit Comparator Example

- Bit comparator: Q indicates if A > B
  G is a cascading input

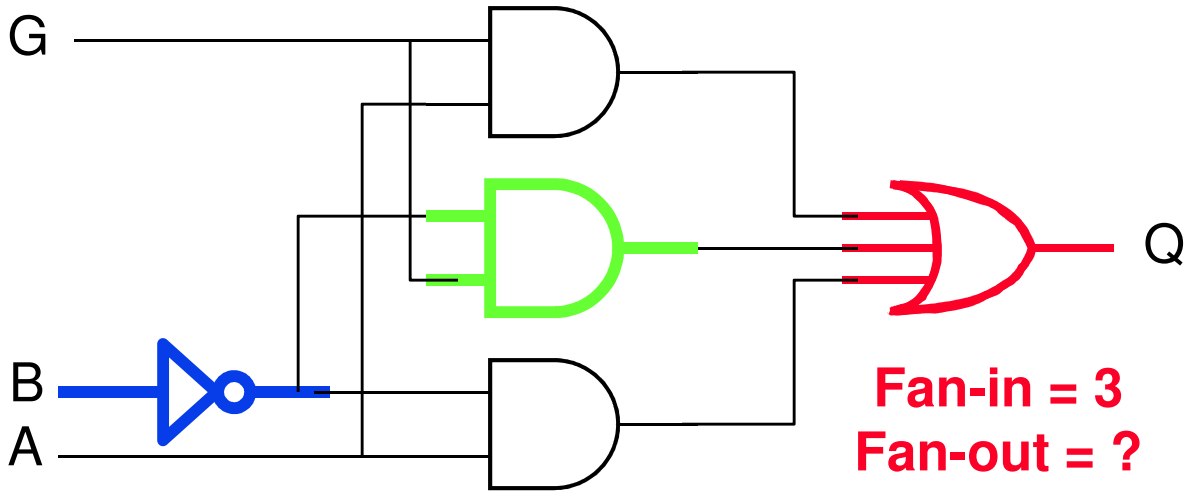


$$Q = A.B' + G.B' + G.A$$



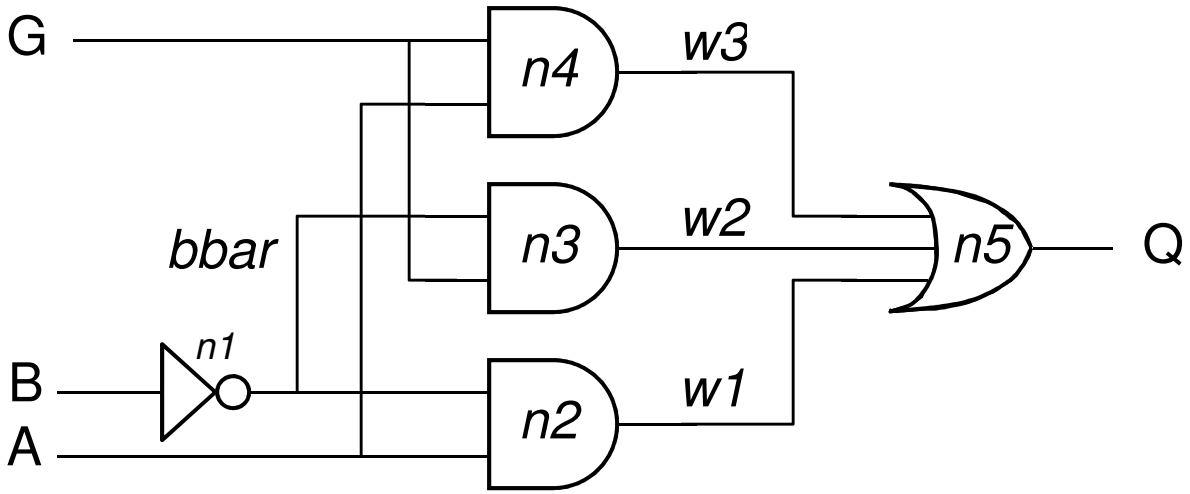
On the side - Fan-in, Fan-out

- Fan-in = number of inputs on a logic gate
- Fan-out = max number of gates that can be driven by an output



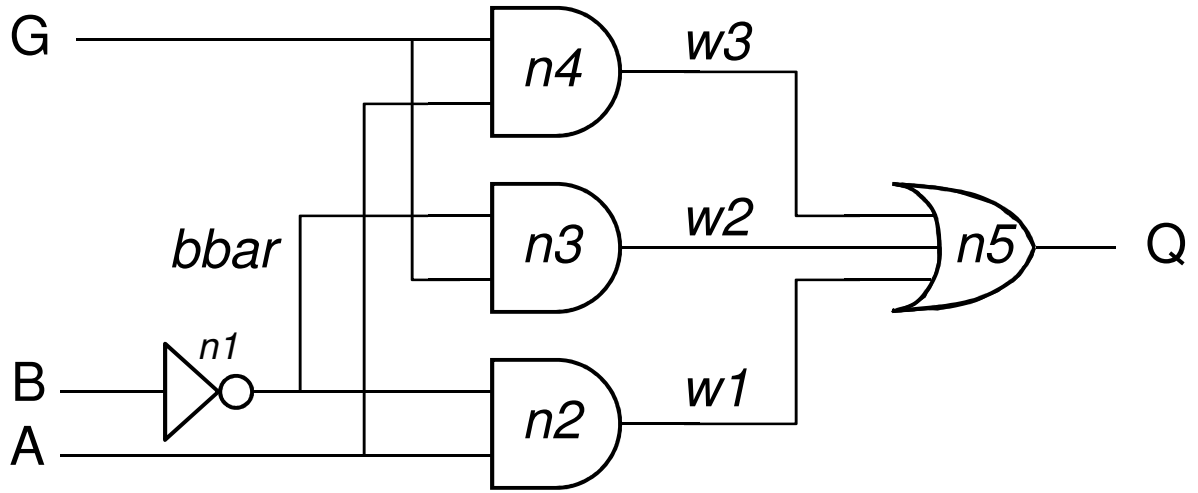
Gate Instantiation: Bit Comparator Example

- To convert to Structural Verilog
  - Label all nets. Create `wire`s for intermediate signals.
  - Label all gate instances
  - Write out netlist in text format



Gate Instantiation: Bit Comparator Example

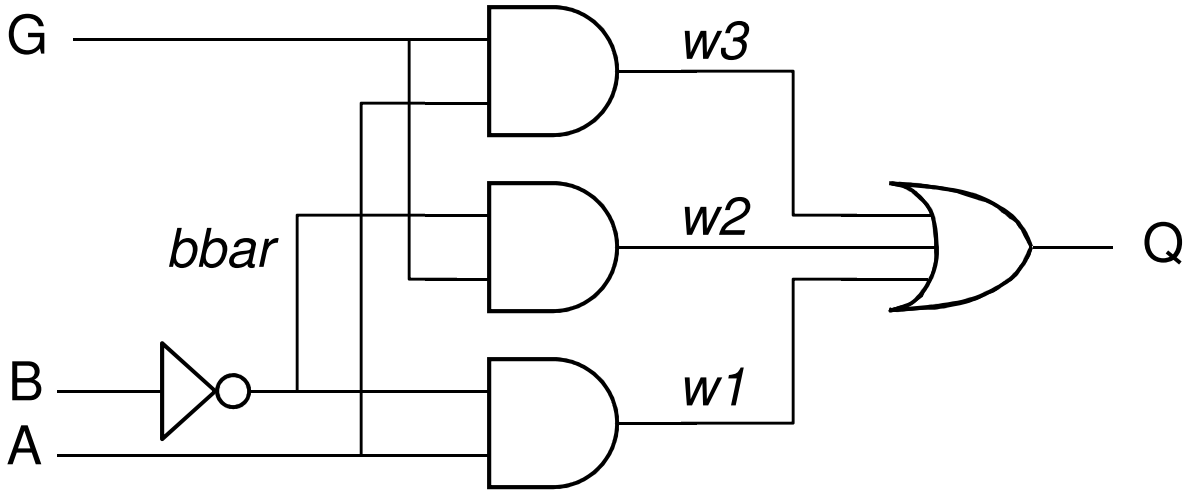
```
module bitcomp(q, a, b, g);
  output q;
  input a, b, g;
  wire bbar, w1, w2, w3;
  not n1(bbar, b);
  and n2(w1, bbar, a);
  and n3(w2, bbar, g);
  and n4(w3, a, g);
  or  n5(q, w1, w2, w3);
endmodule
```



Gate Instantiation: Bit Comparator Example

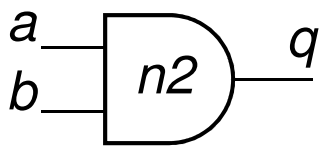
```
module bitcomp(q, a, b, g);
  output q;
  input a, b, g;
  wire bbar, w1, w2, w3;
  not (bbar, b);
  and (w1, bbar, a);
  and (w2, bbar, g);
  and (w3, a, g);
  or  (q, w1, w2, w3);
endmodule
```

Gate instance name is optional

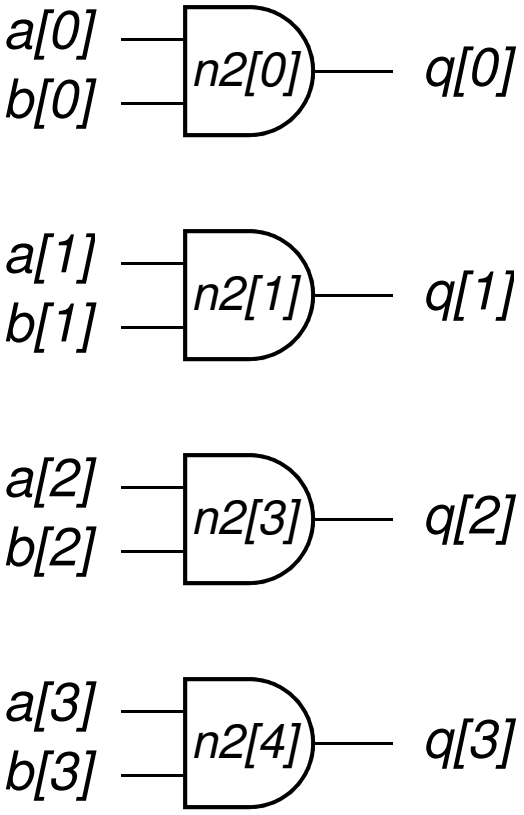


Gate Array Instantiation

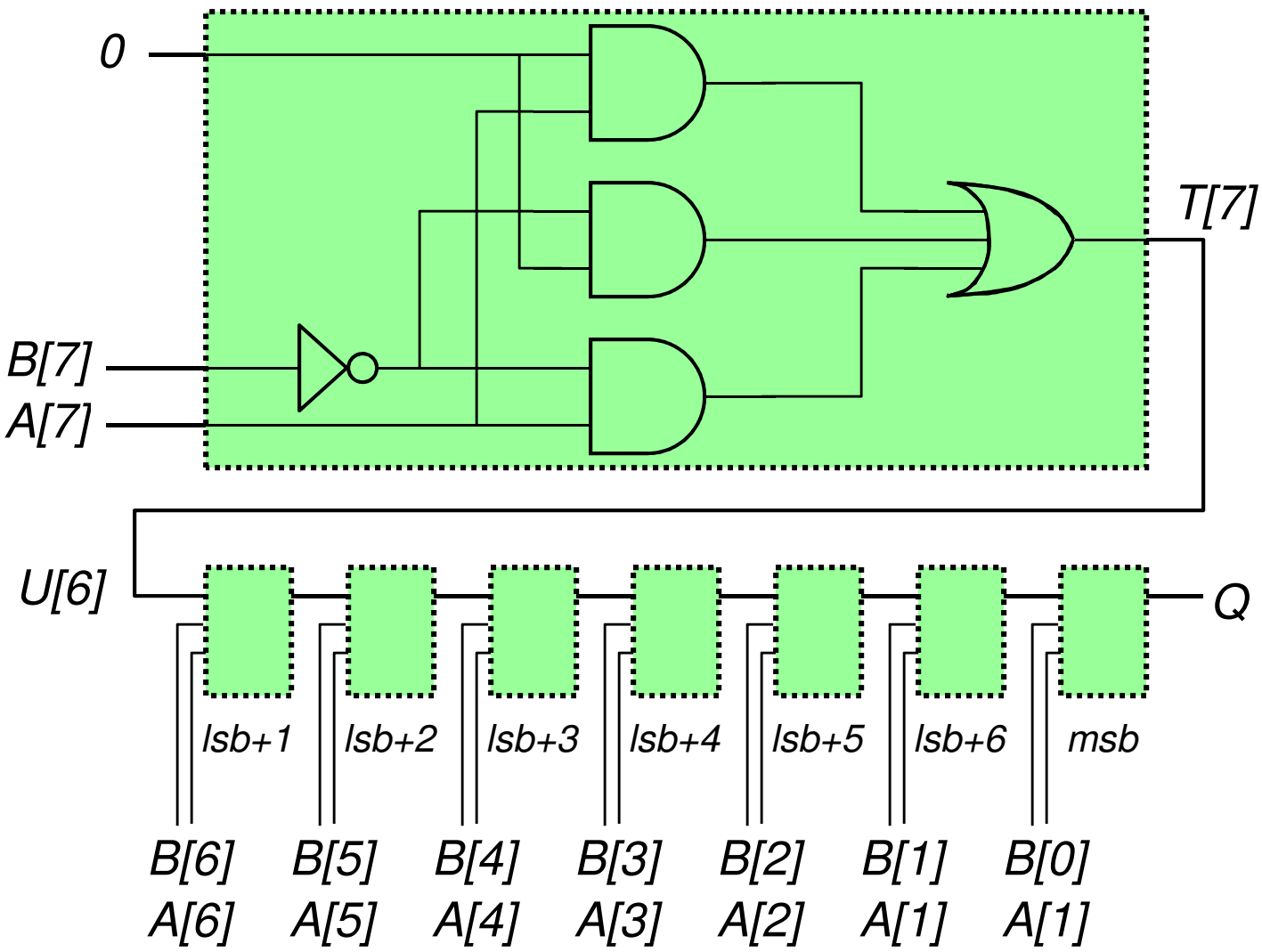
```
wire q, a, b;
and (q, a, b);
```




```
wire [0:3] q, a, b;
and [0:3] (q, a, b);
```



Gate Array Instantiation: Byte Comparator



Gate Instantiation: Bit Comparator Example

```
module bytecomp(q, a, b);
  output q;
  input [0:7] a, b;
  wire  [0:7] t, u, bbar, w1, w2, w3;

  not n1[0:7] (bbar, b);        // 8 not gates
  and n2[0:7] (w1, bbar, a);    // 8 and gates
  and n3[0:7] (w2, bbar, u);    // ...
  and n4[0:7] (w3, a, u);
  or  n5[0:7] (t, w1, w2, w3);

  assign u[7]   = 0;            // wire-to-wire
  assign u[0:6] = t[1:7];       //    connection
  assign q      = t[0];
endmodule
```

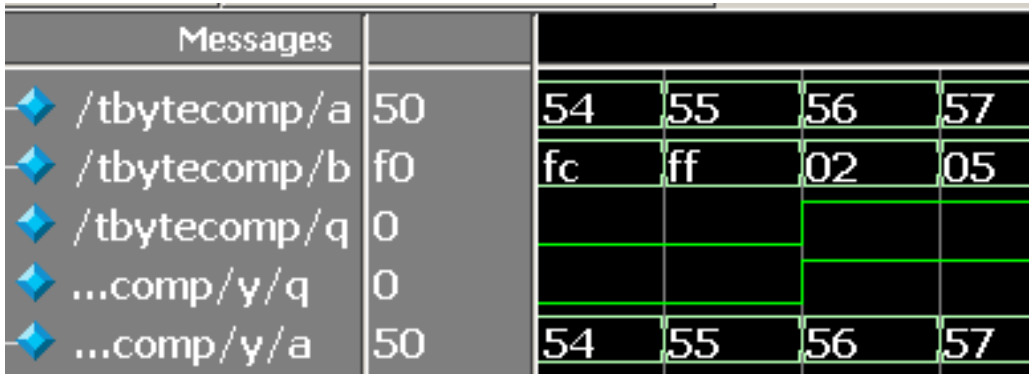
Alternate Description of Byte Comparator

```
module bytecomp(q, a, b);
  output q;
  input [0:7] a, b;
  wire  [0:7] t, u;

  bitcomp n1[0:7] (t, a, b, u);

  assign u[7]   = 0;          // wire-to-wire
  assign u[0:6] = t[1:7];     //    connection
  assign q      = t[0];
endmodule
```

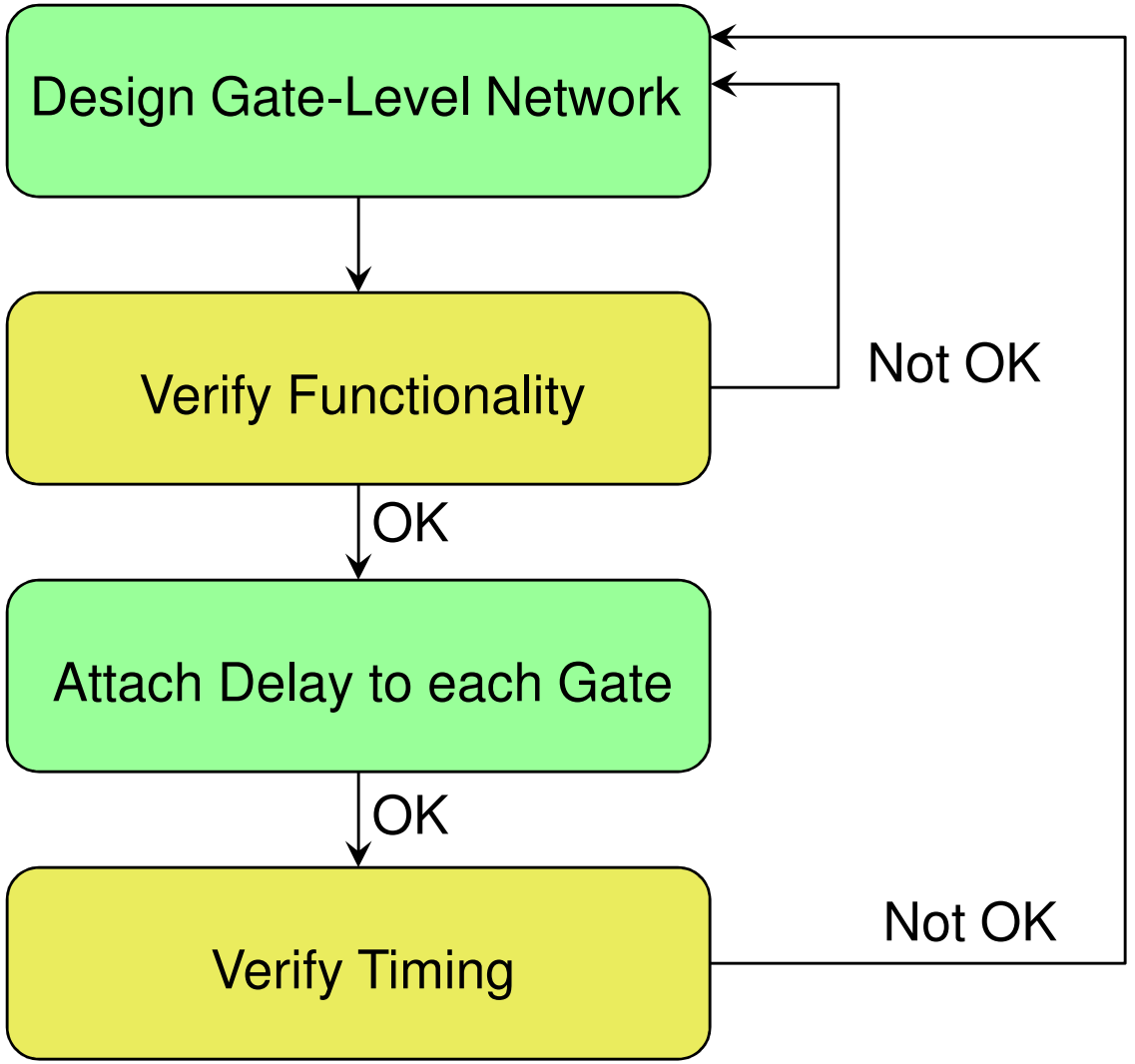
- In the above description there are 3 levels of hierarchy.
- In the description on the previous slide, there are only two levels of hierarchy.

Bytecomp Testbench Example

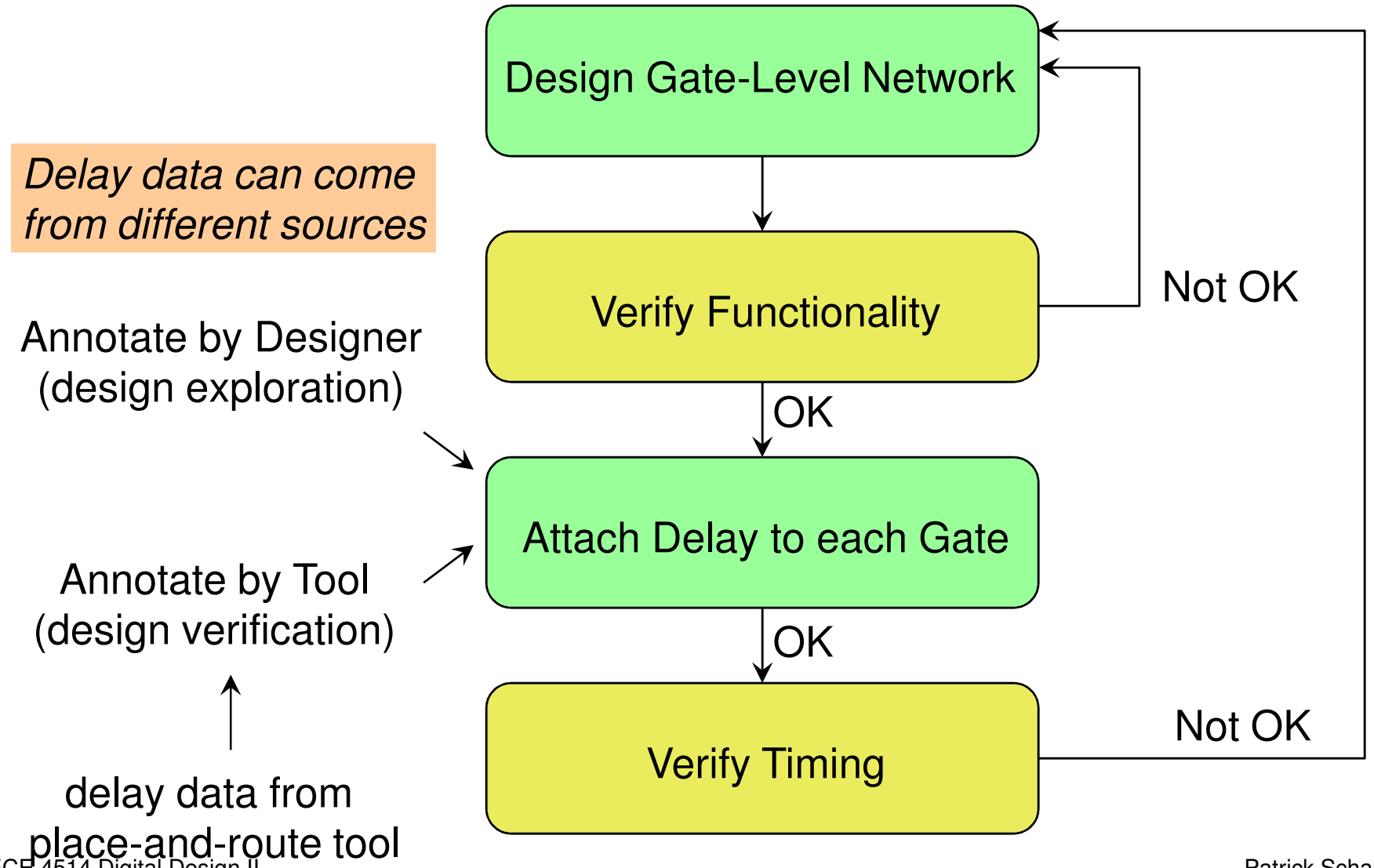
```
module tbytecomp;
     reg [0:7] a, b;          // reg at input
     wire  q;                 // only wire at output
     bytecomp y(q, a, b);

     initial begin
     a = 0;
     b = 0;
     end

     always begin
          #10
          a = a + 1;
          b = b + 3;
     end
endmodule
```



Gate Level Timing - The Big Idea

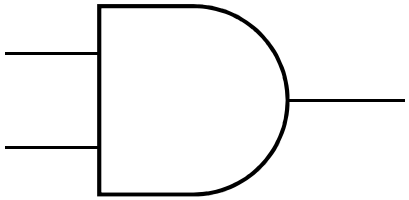


Gate Level Timing - The Big Idea

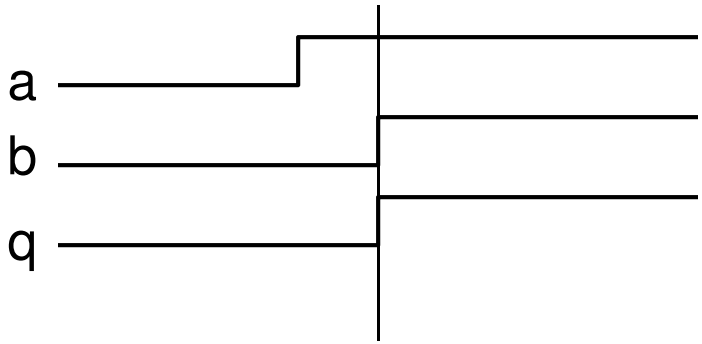


Gate-level timing

- Default model for gates does not have delay

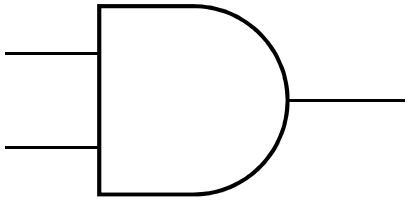


```
and n1(q, a, b);
```

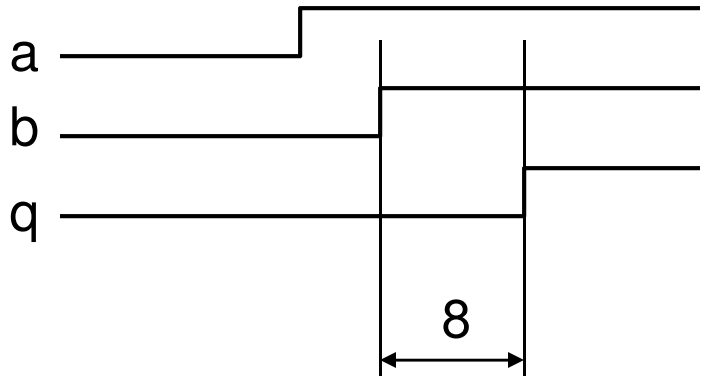


Gate-level timing

- Standard delay model specifies propagation delay



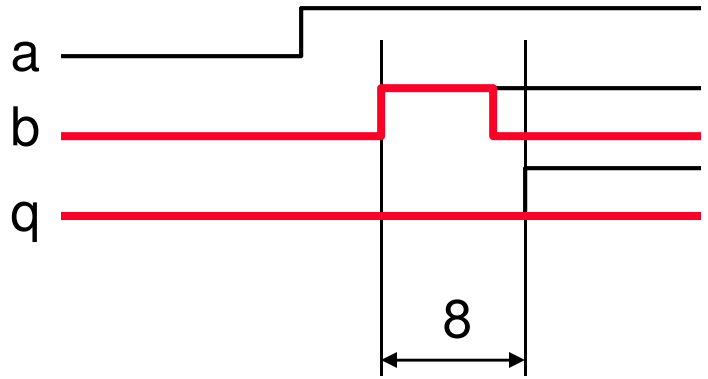
```
and #(8) n1(q, a, b);
```



Gate-level timing

- Propagation Delay for gates really specifies two things:
  - Delay from input to output
  - Minimum width of pulses to affect the output

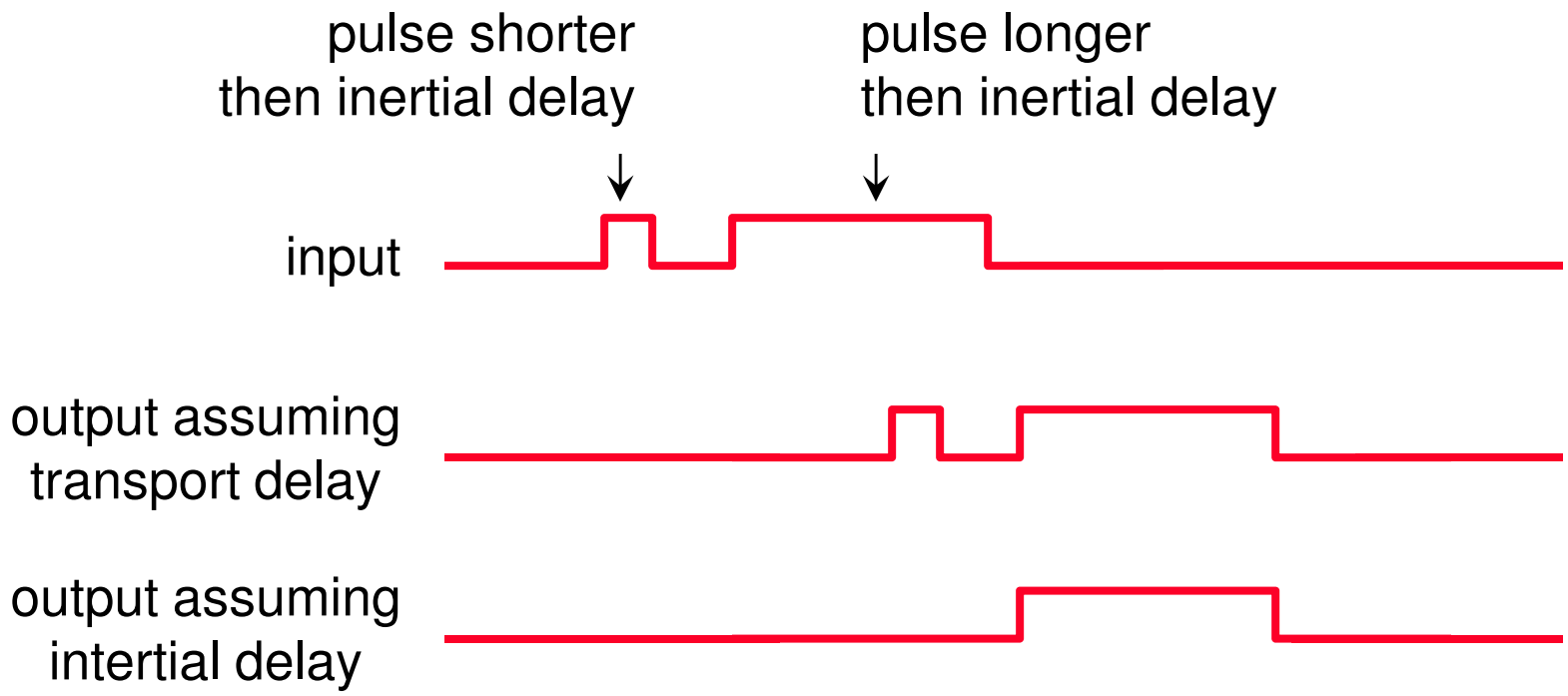
```
and #(8) n1(q, a, b);
```



Pulses smaller than 8
do not propagate to the output

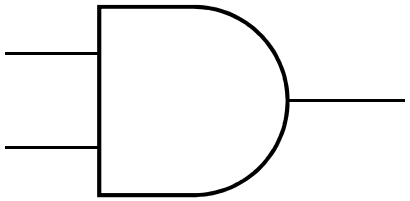
Delay Propagation Models

- Gates use *inertial delay*
- Designers may also want to use *transport delay* (for example to model slow wires).

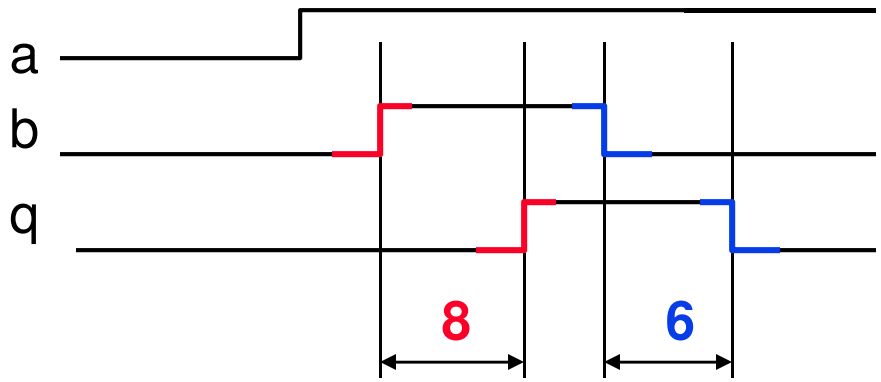


Gate-level timing

- Advanced model specifies propagation delay for up- and down-transitions separately



```
and #(8, 6) n1(q, a, b);
```



Gate-level timing

- Other delays follow automatically, for example

```
and #(8, 6) n1(q, a, b);
```

Rising Delay	Falling Delay
0 to 1	1 to 0
X to 1	X to 0
Z to 1	Z to 0

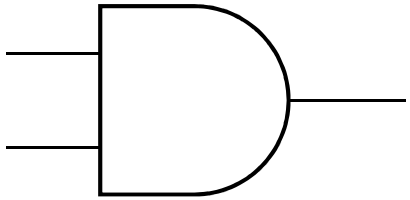
In all other cases: Delay = min(Rise, Fall)

E.g. 1 to X ? Take min(8,6) = 6

Min:Typ:Max Delays

- Circuits show delay variations
  - Due to implementation & process parameters
  - Due to voltage and temperature
- Designers create circuits to work within an operation-range, corresponding to a domain (military, commercial, ..)
- Verilog allows to capture the effect of these variations by supporting multiple delay specifications for a gate
  - 'Typical': room temperature, normal voltage
  - 'Max': high temperature, low voltage, [slow circuit]
  - 'Min': low temperature, high voltage, [fast circuit]

Min:Typ:Max Delays



```
and #(5:8:10) n1(q, a, b);
```

minimum

typical

maximum

In Modelsim, select the configuration to use with:

```
vsim +mindelays
vsim +typdelays
vsim +maxdelays
```

Where are the time units?

- Verilog specifies relative delay in *units*

- The simulation timescale is provided through a directive at the top of the Verilog file (or modelsim.ini)

```
`timescale 10ns / 100ps
```

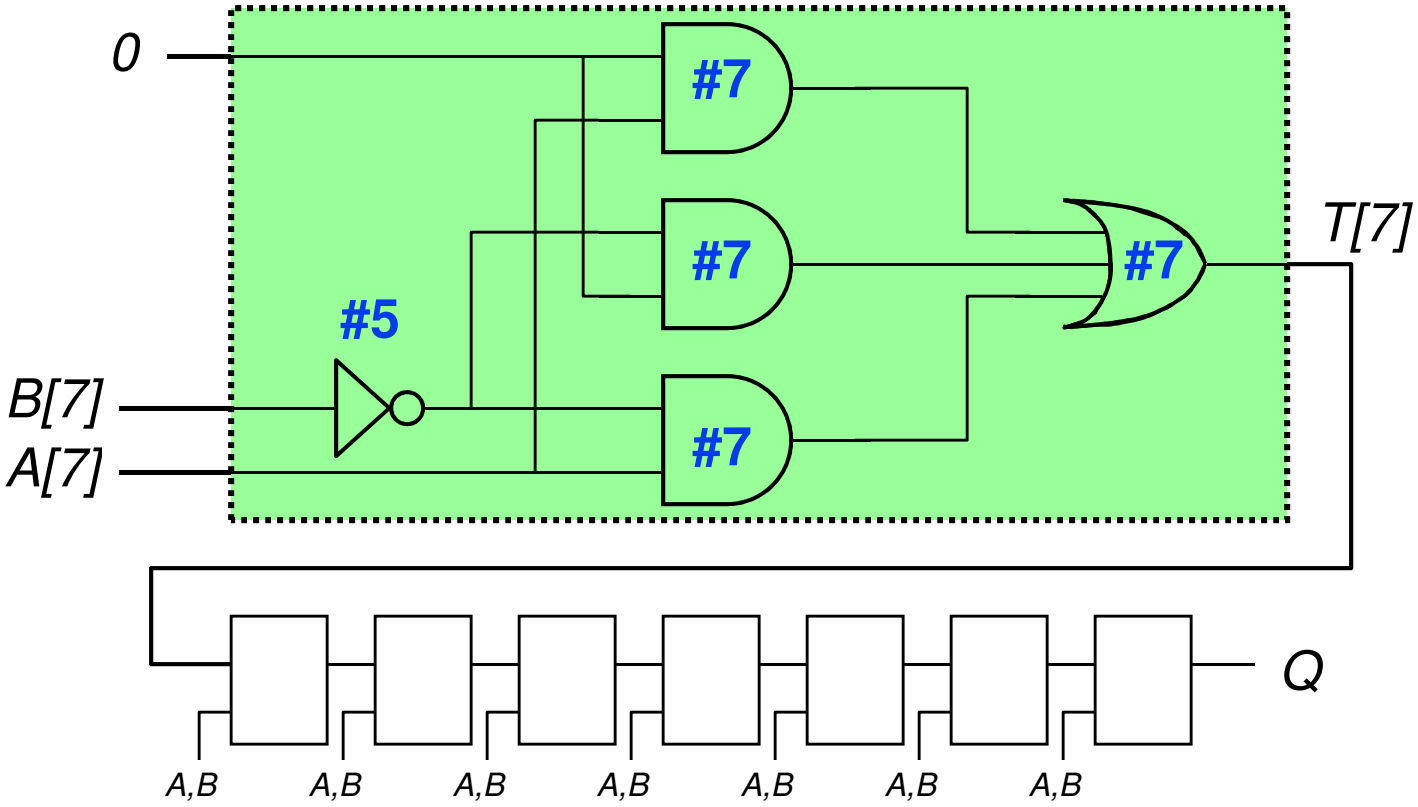
Time Unit
#33 equals 330 ns

Simulation Precision
#45.1355 equals 451 ns
#66.97 equals 670 ns

Gate Level Timing Example

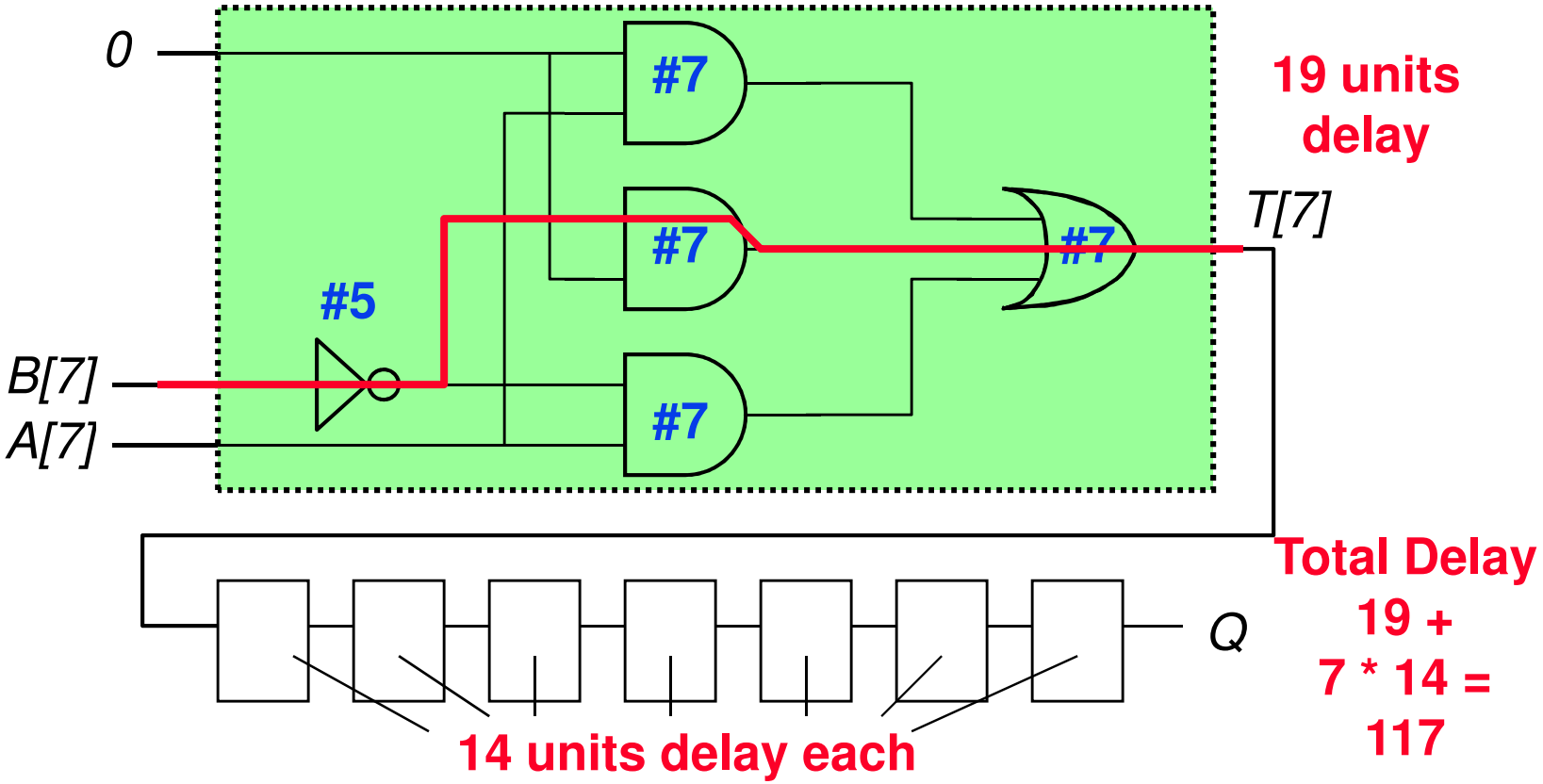
```
module bytecomp(q, a, b);
  output q;
  input [0:7] a, b;
  wire  [0:7] t, u, bbar, w1, w2, w3;

  not #5 n1[0:7] (bbar, b);        // 8 not gates
  and #7 n2[0:7] (w1, bbar, a);    // 8 and gates
  and #7 n3[0:7] (w2, bbar, u);    // ...
  and #7 n4[0:7] (w3, a, u);
  or  #7 n5[0:7] (t, w1, w2, w3);

  assign u[7]   = 0;               // wire-to-wire
  assign u[0:6] = t[1:7];          //    connection
  assign q      = t[0];
endmodule
```

Estimating worst-case timing of bytecomp



1. **What is the worst possible path through the byte comparator ?**
2. **How can you trigger it ?**

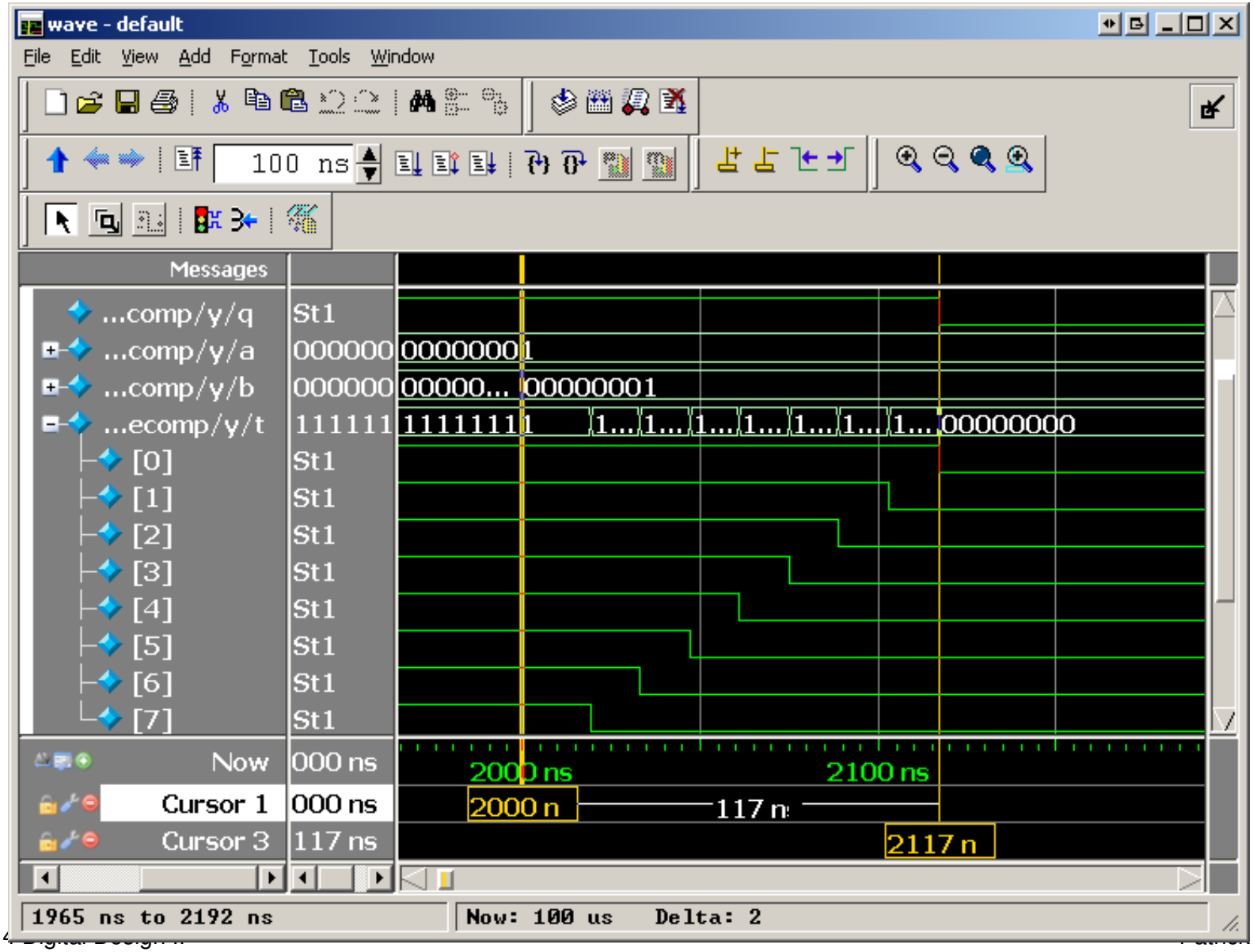
Estimating worst-case timing



1. **What is the worst possible path through the byte comparator ?**
2. **How can you trigger it ?**

Apply 1 to A and 0 to B -> Q is high
then set B to 1 -> Q falls to low after 117

Simulating worst-case timing



Testbench for worst-case timing

```
module tbytecomp;
    reg [0:7] a, b;
    wire  q;

    bytecomp y(q, a, b);

    initial begin
    a = 0;
    b = 0;
    end

    always begin
        #1000
        a = 1;
        b = 0;
        #1000
        a = 1;
        b = 1;
    end
endmodule
```

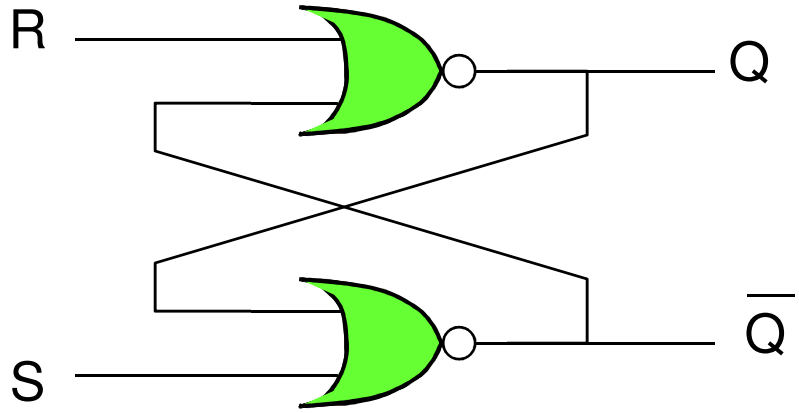
Latches and Flip-flops

- So far, the only flip-flop we used was a D-flipflop described at *dataflow* level

```
module syntst(q, d, clk);
  input clk, d;
  output q;
  reg q;

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

endmodule
```

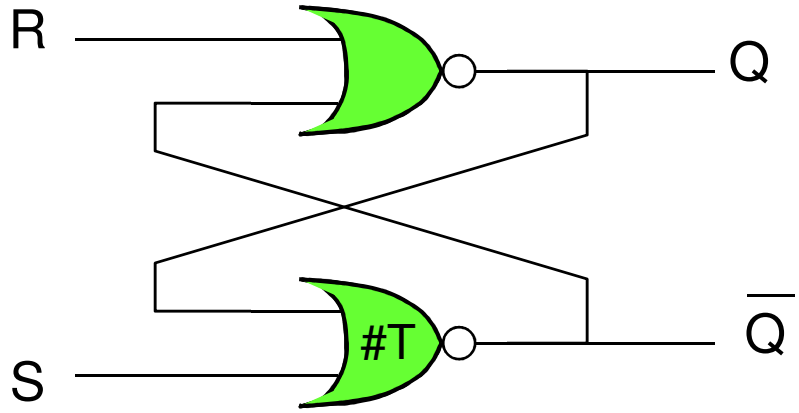
- We can also create storage modules out of gates

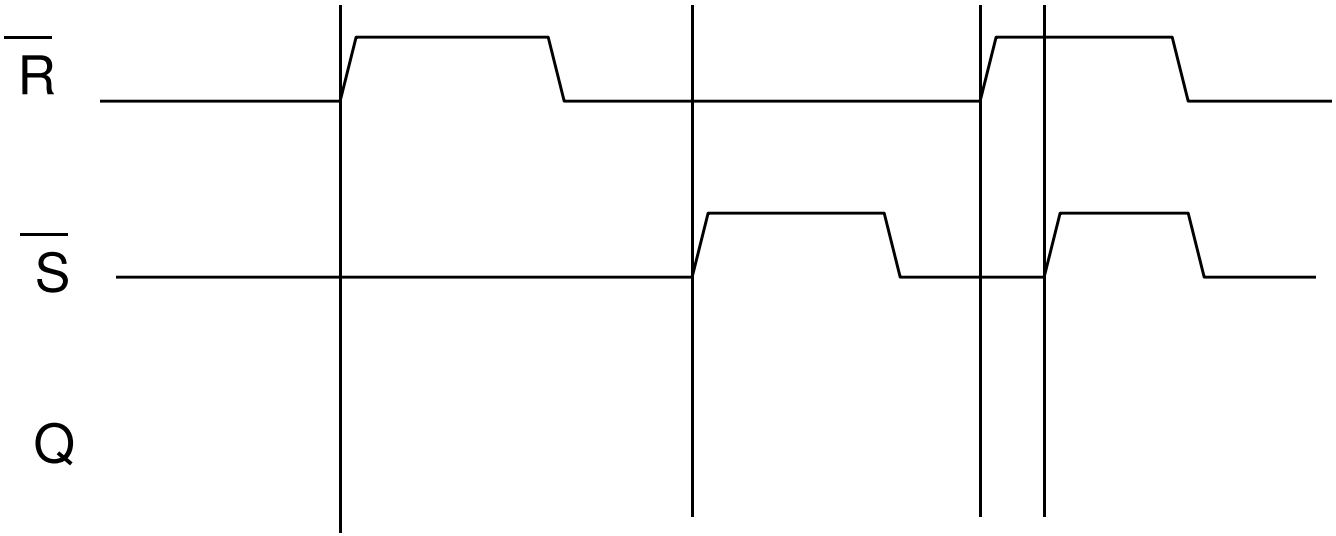
SR Latch



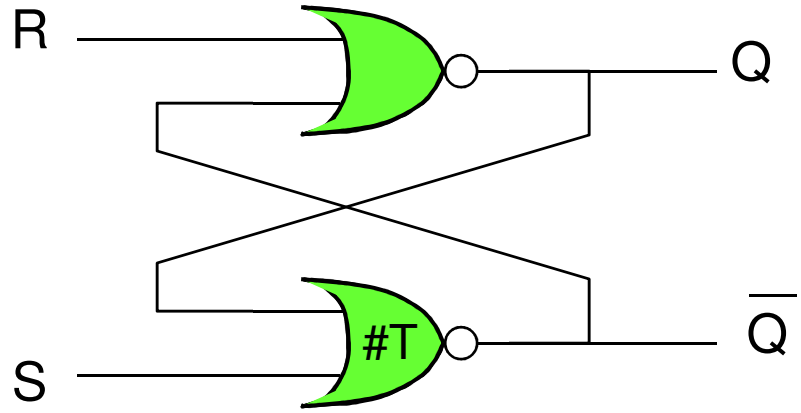

```
module sr_latch(Q, Qbar, S, R);

output Q, Qbar;
input S, R;

nor n1(Q, Sbar, Qbar);
nor n2(Qbar, Rbar, Q);

endmodule
```

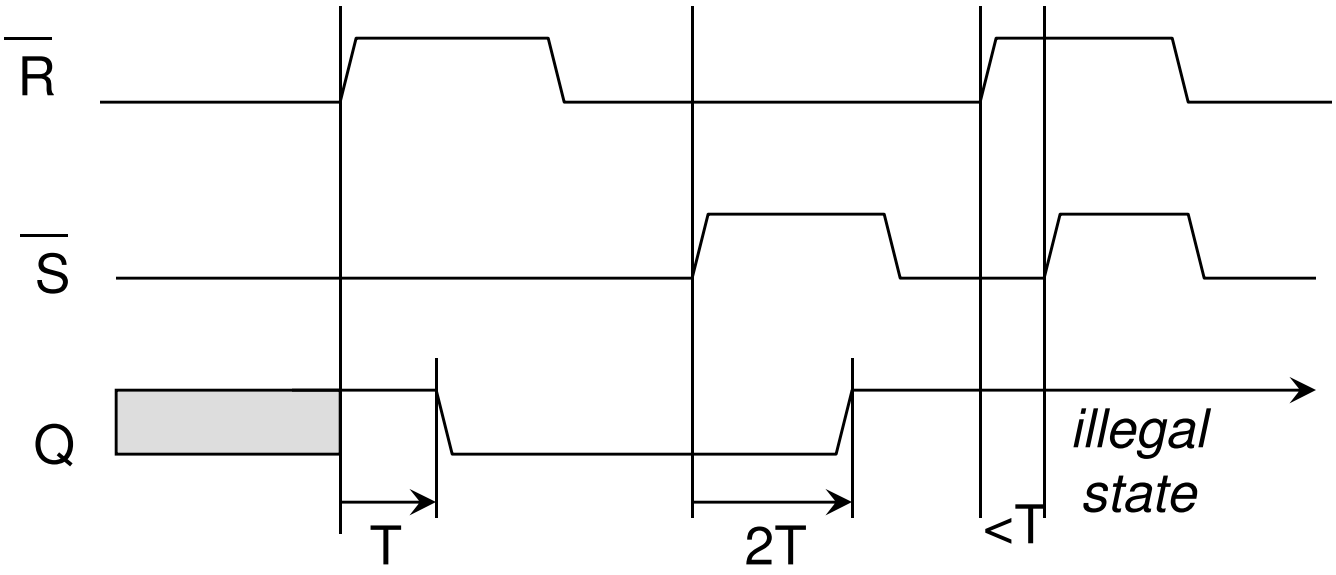
SR Latch



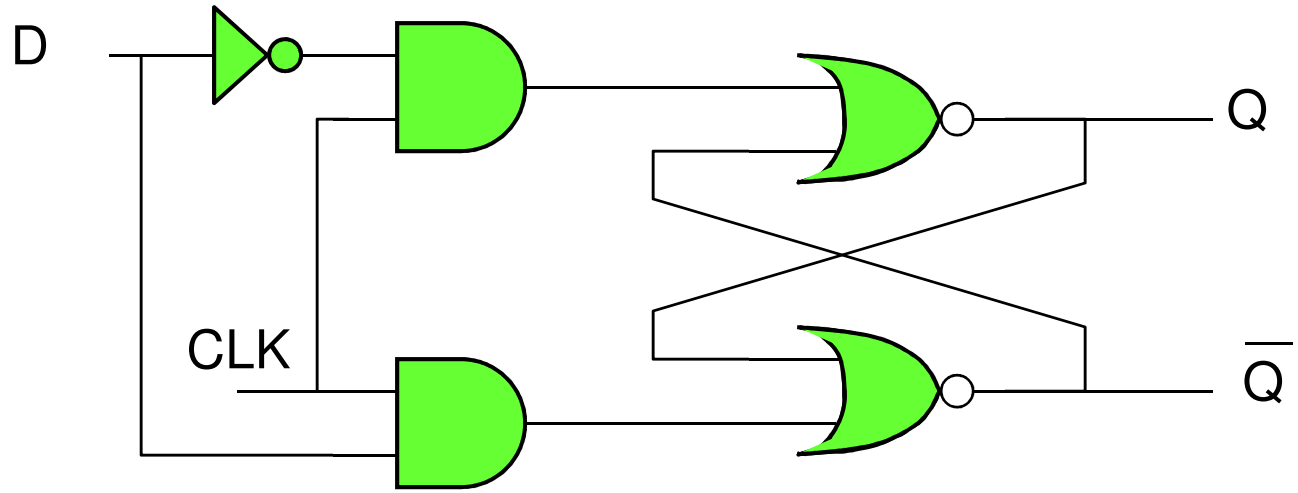


SR Latch



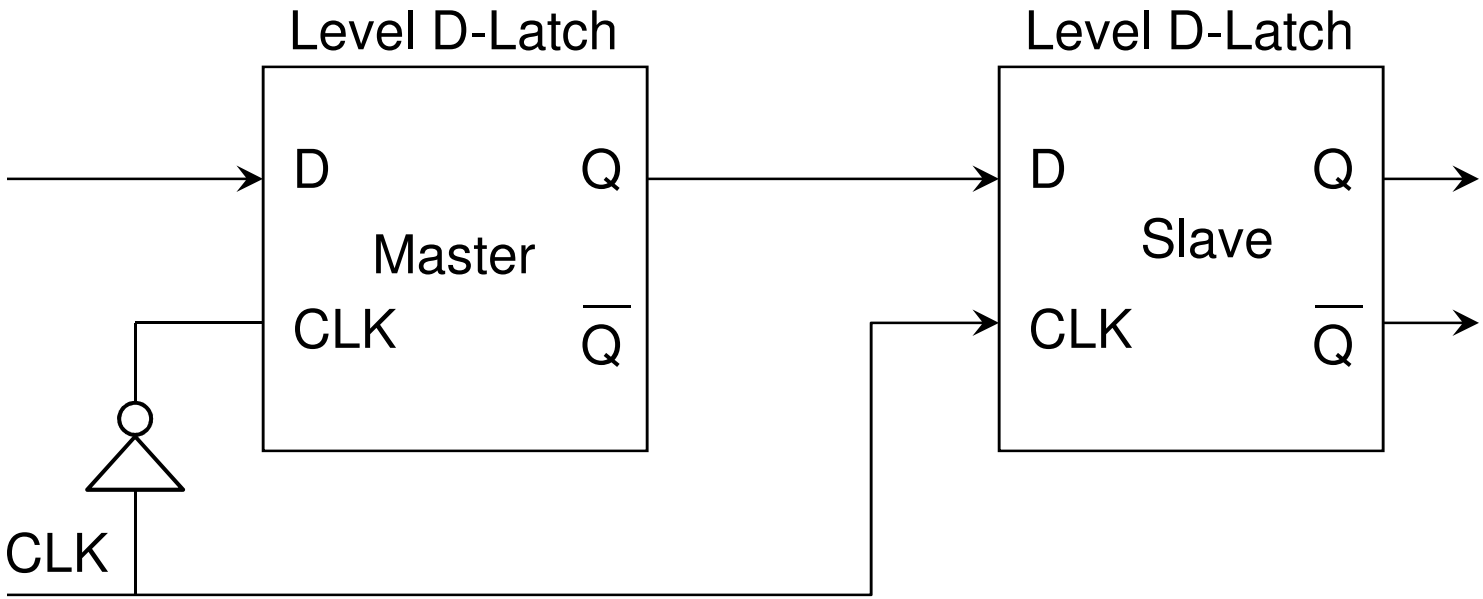


Level-Sensitive D-Latch



- clk to short: D will not propagate to SR
- clk to long: D may change while clk is high

Edge-Sensitive D-Latch



CLK	low	high
Master	open	close
Slave	close	open

Summary

- Gates are primitive modules for the Verilog simulator
- Create gate networks as structural Verilog netlists
- Annotate netlist with delays for timing analysis
- Evaluation of critical path requires more than decorating times - also clever analysis to trigger worst-case condition