# EECS 151/251A
# SP2022 Discussion #2

GSI: DIMA NIKIFOROV, YIKUAN CHEN

# Agenda

- Administrivia
- More Verilog
- Testbenches
- Combinational Logic

# Administrivia

- Thoughts on hybrid instruction/labs?
- Homework 2 posted

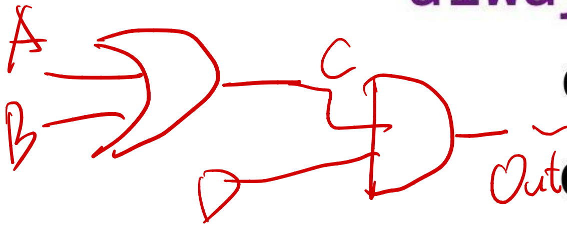
# More Verilog

# Blocking vs. Nonblocking

<=

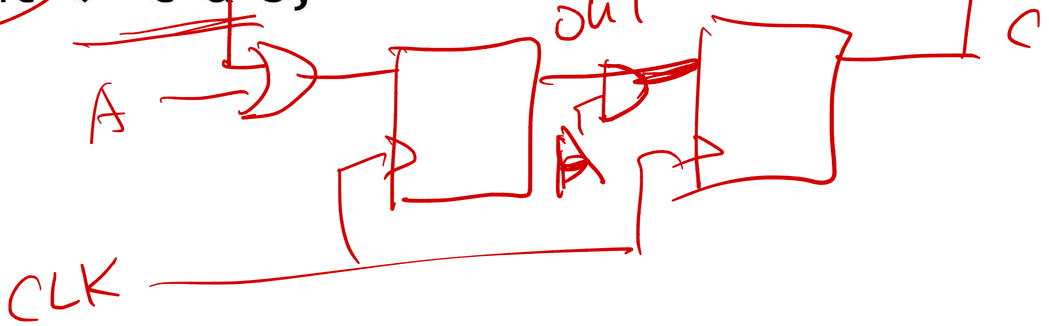
## Blocking:

```
reg c, out;
wire a, b, d;
always @(*) begin
    c = a | b;
    out = c & d;
end
```



## Non-Blocking:

```
reg c, out;
wire a, b, clk;
always @(posedge clk) begin
    //use val of ‘out’ before clk edge
    c <= out | a;
    //use val of ‘c’ before clk edge
    out <= c & b;
end
```

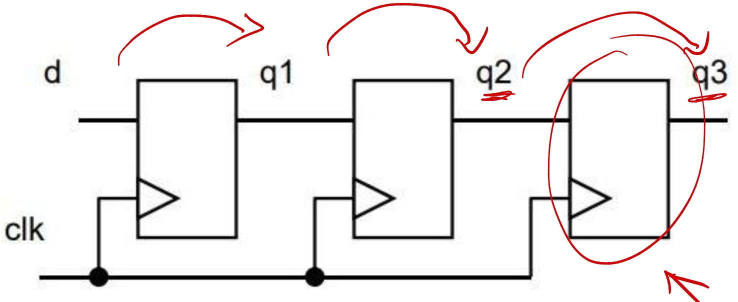


- Don’t mix blocking and nonblocking!
- When would you use either one?

# Race Conditions: Synthesis vs. Simulation

multi driver

always@(..)
q1 = ... ;
end
always @(..)
q1 = ... ;
end
drive.



Want: a register pipeline

Determine:

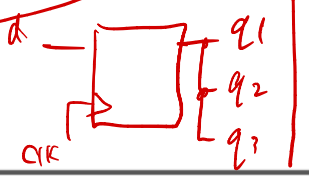
1. Does it *synthesize* correctly?
2. Does it *simulate* correctly?
    - Note: always blocks may simulate in any order
3. Is it good coding practice?

YNN

<=

Candidate #1:

```
always @(posedge clk) begin
  q1 = d;
  q2 = q1;
  q3 = q2;
end
```



YNN

Candidate #4:

```
always @(posedge clk) q1 = d;
always @(posedge clk) q2 = q1;
always @(posedge clk) q3 = q2;
```

YYN

Candidate #2:

```
always @(posedge clk) begin
  q3 = q2;
  q2 = q1;
  q1 = d;
end
```

Candidate #5:

```
always @(posedge clk) q3 = q2;
always @(posedge clk) q2 = q1;
always @(posedge clk) q1 = d;
```

YYY

Candidate #3:

```
always @(posedge clk) begin
  q1 <= d;
  q2 <= q1;
  q3 <= q2;
end
```

new

old

☆

Candidate #6:

```
always @(posedge clk) q1 <= d;
always @(posedge clk) q2 <= q1;
always @(posedge clk) q3 <= q2;
```

# Testbenches

# What’s a Testbench?

z1top.v

↑

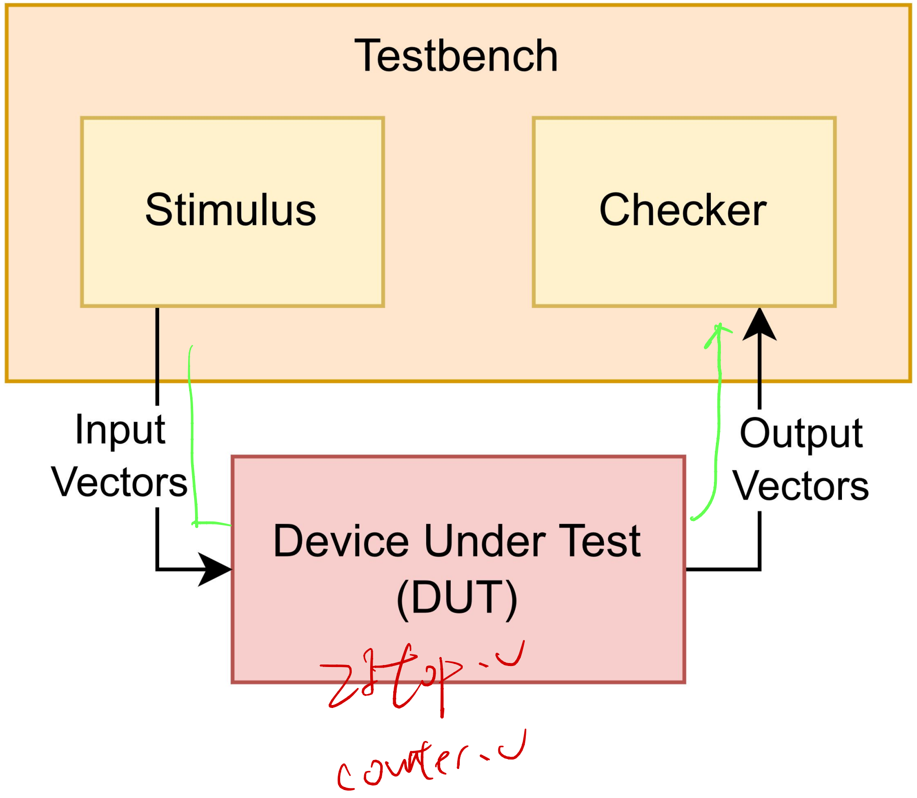
top

z1top_tb.v

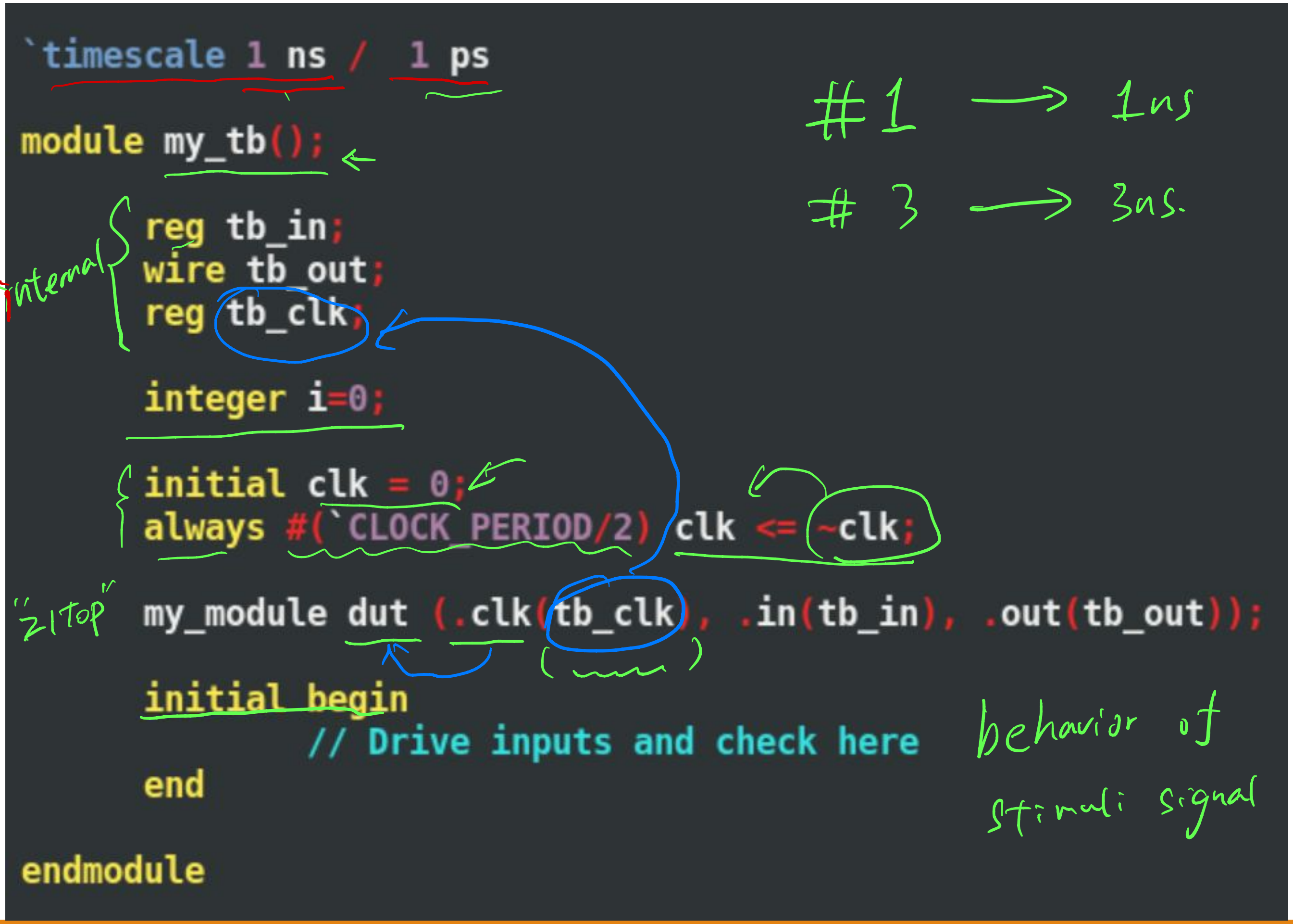
↑

sim top

- Tool to verify that design behaves as specified
- Generate inputs to drive design
- Compare outputs against expected results



# Example Testbench



```
`timescale 1 ns /  1 ps

module my_tb();

    reg tb_in;
    wire tb_out;
    reg tb_clk;

    integer i=0;

    initial clk = 0;
    always #(`CLOCK_PERIOD/2) clk <= ~clk;

    my_module dut (.clk(tb_clk), .in(tb_in), .out(tb_out));

    initial begin
            // Drive inputs and check here
    end

endmodule
```

#1 → 1ns

#3 → 3ns.

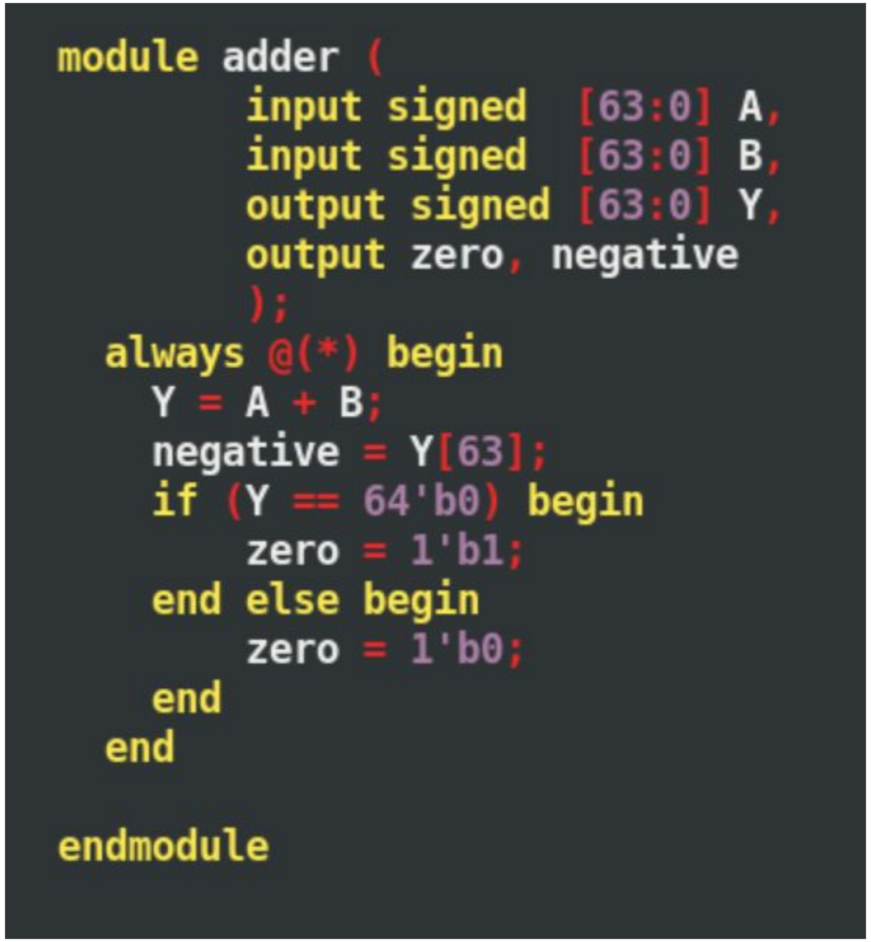
internal

"Z1Top"

behavior of stimuli signal

# What Makes a Good Testbench?

- Code coverage:
  - Statements
  - Branches
  - Toggles
  - States
- Functional coverage
  - Features/Requirements



```
module adder (
        input signed  [63:0] A,
        input signed  [63:0] B,
        output signed [63:0] Y,
        output zero, negative
        );
  always @(*) begin
    Y = A + B;
    negative = Y[63];
    if (Y == 64'b0) begin
        zero = 1'b1;
    end else begin
        zero = 1'b0;
    end
  end

endmodule
```

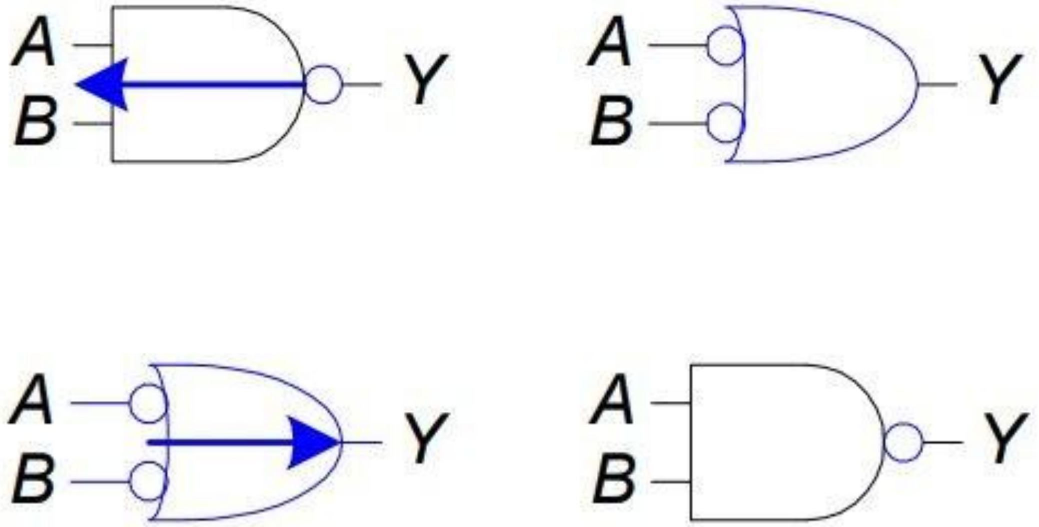
# Combinational Logic

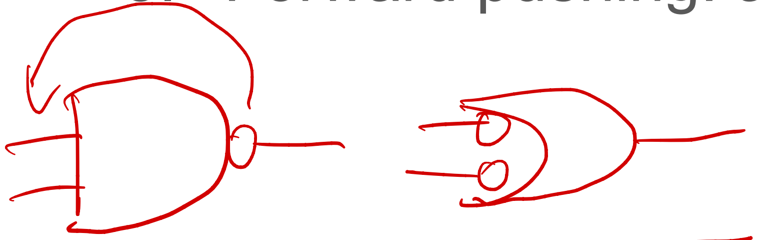
$\overline{\overline{\overline{AB(C+D)}}\cdot E}$

# Boolean Algebra: DeMorgan’s

- First step towards logic simplification
- Recall: (x+y)’ = x’y’ , (xy)’ = x’+y’
- Bubble = inversion (NOT)
- Steps for a single gate:

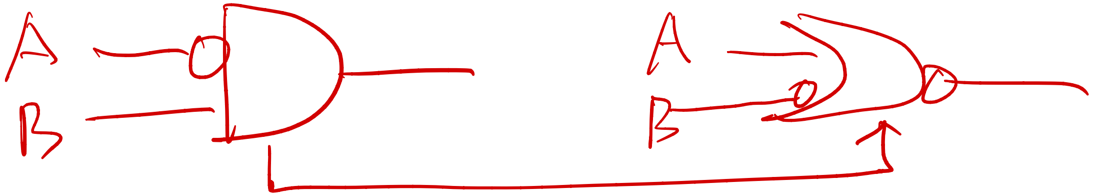
1. Swap AND for OR & vice versa
2. Backward pushing: add bubbles to inp
3. Forward pushing: add bubbles to outpu





NAND

$\overline{A\cdot B} = \overline{A}+\overline{B}$

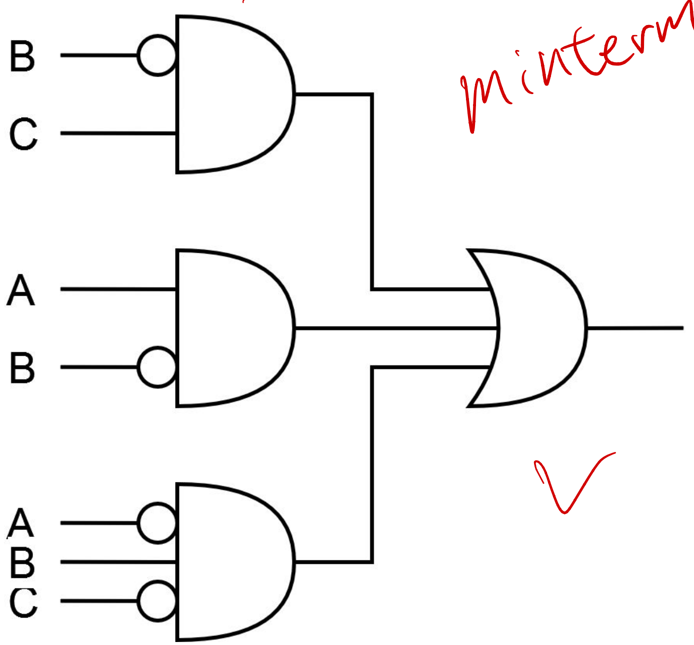


# Canonical Forms

SoP

- Sum of Products (SoP):
  - $\underline{\overline{B}C} + \underline{A\overline{B}} + \underline{\overline{A}B\overline{C}}$

minterm

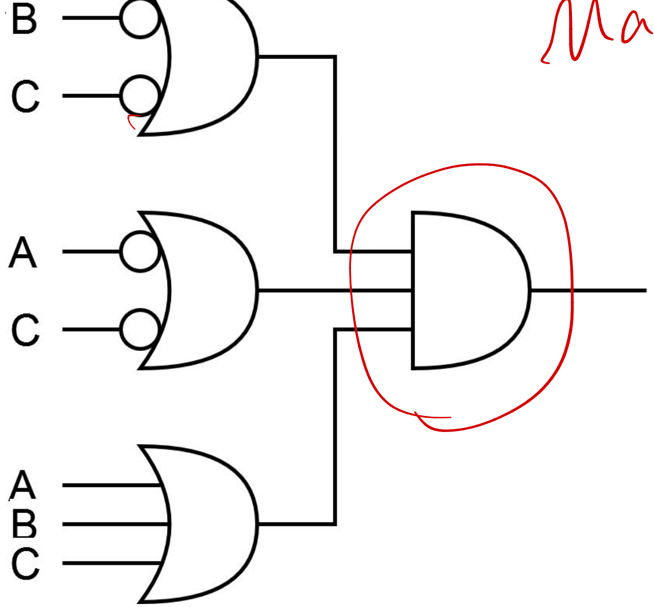


✓

Product of Sum

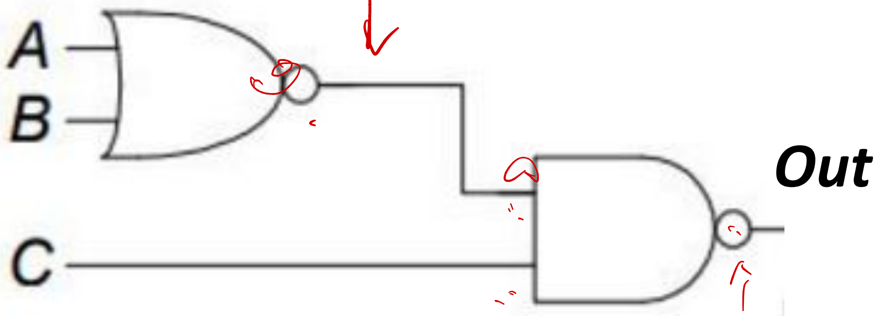
- ~~Sum of Products (SoP):~~
  - $(\overline{B} + \overline{C})\ (\overline{A} + \overline{B}\ )\ (A + B + C)$

Maxterm



# Truth Tables

$\overline{(A+B)}$



$C$

$$\overline{\overline{(A+B)}\cdot C} = (A+B)+\overline{C}$$

$$= A + B + \overline{C}$$

0 0 1

| *A* | *B* | *C* | *Out* |
|---|---|---|---|
| 0 | 0 | 0 | 1 |
| 0 | 0 | 1 | 0 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 1 |
| 1 | 0 | 0 | 1 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

# Truth Tables

| A | B | C | Out |
|---|---|---|---|
| 0 | 0 | 0 | 0 |
| 0 | 0 | 1 | 1 |
| 0 | 1 | 0 | 1 |
| 0 | 1 | 1 | 0 |
| 1 | 0 | 0 | 0 |
| 1 | 0 | 1 | 1 |
| 1 | 1 | 0 | 1 |
| 1 | 1 | 1 | 1 |

SoP

$\bar{A}\bar{B}C + \bar{A}B\bar{C} + A\bar{B}C + AB\bar{C} + ABC$

PoS

# 2-input K-Map

$F(A,B) = \overline{A}B + AB + A\overline{B}$

0 1 ↑

$F = A + B = \overline{\overline{A}\,\overline{B}}$

| A \ B | 0 | 1 |
|---|---|---|
| 0 | 0 | 1 |
| 1 | 1 | 1 |

# 4-input K-Map

$$F(A,B) = \bar{A}\bar{B}\bar{C}\bar{D} + \bar{A}\bar{B}C\bar{D}$$

$$+$$

$$\bar{A}B\bar{C}\bar{D} + \bar{A}B\bar{C}D +$$

$$AB\bar{C}\bar{D} + ABC\bar{D} +$$

$$A\bar{B}CD + A\bar{B}\bar{C}\bar{D} +$$

$$ABCD$$

| AB \ CD | | | | |
|---|---|---|---|---|
| 00 | | | | |
| 01 | | | | |
| 11 | | | | |
| 10 | | | | |

# Questions?