

# EECS 151/251A SP2022 Discussion 1

GSI: Yikuan Chen, Dima Nikiforov

# My job:

- To help **you** get the most out of this class!
- Discussion sections
  - Cory 540 every Friday
  - Review this week
  - Answer questions
  - Example problems
- Office hours
  - Yikuan's: Thursday 10-11am
  - Office hour of each staff listed on course website:
    - <https://inst.eecs.berkeley.edu/~eecs151/sp22/>
- Piazza
  - Please contact GSIs here instead of email so your question could bring value to all students
  - GSIs will try to respond within 24 hours

# How to success

- Put the most effort into labs/project

They make you a great engineer, not just a good IC student

- Understand abstraction leverage it for productive design

Stay in circuit design: Apple shows you how desperate they are!

- Choose final project partners wisely. It Takes Two to make a team

# What else

- Homework stresses understanding  
All questions will be graded on correctness, but [open-ended](#) questions have higher weighting
- Stay up-to-date on industry & research trends!  
[IEEE Computing & Semiconductors](#), [EE Times](#), [Semiconductor Engineering](#), [TechInsights](#)  
[IEEE CAS](#) & [Computer Societies](#), [ACM](#), etc.

# Agenda

- Administrative
- ASIC vs FPGA
- Verilog intro

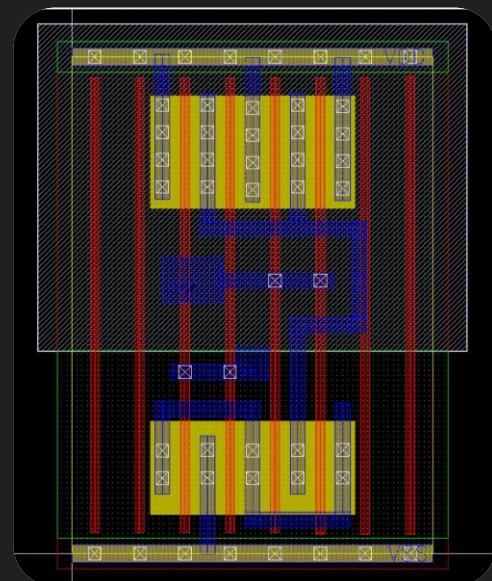
# Administrative

- Homework 1 will be posted this week due on Friday.
- Labs are in Cory 111/117 – There's a Monday Lab 2-5pm, GSI Seah Kim  
Western Entrance

# ASIC vs FPGA

# ASIC

- Application-Specific Integrated Circuit
- Optimized for the application
- Use standard cells, SRAM, custom analog circuits



2-input NAND gate

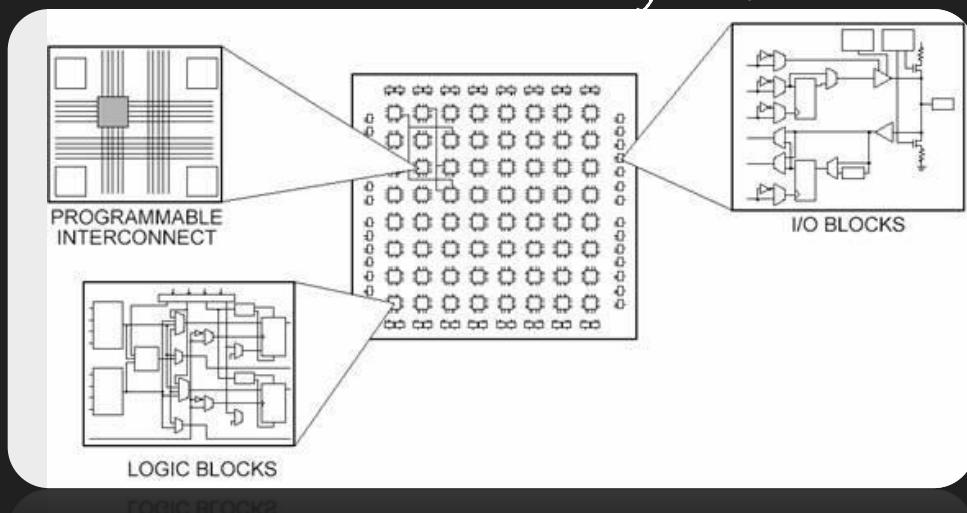


Apple's A11 Bionic SoC

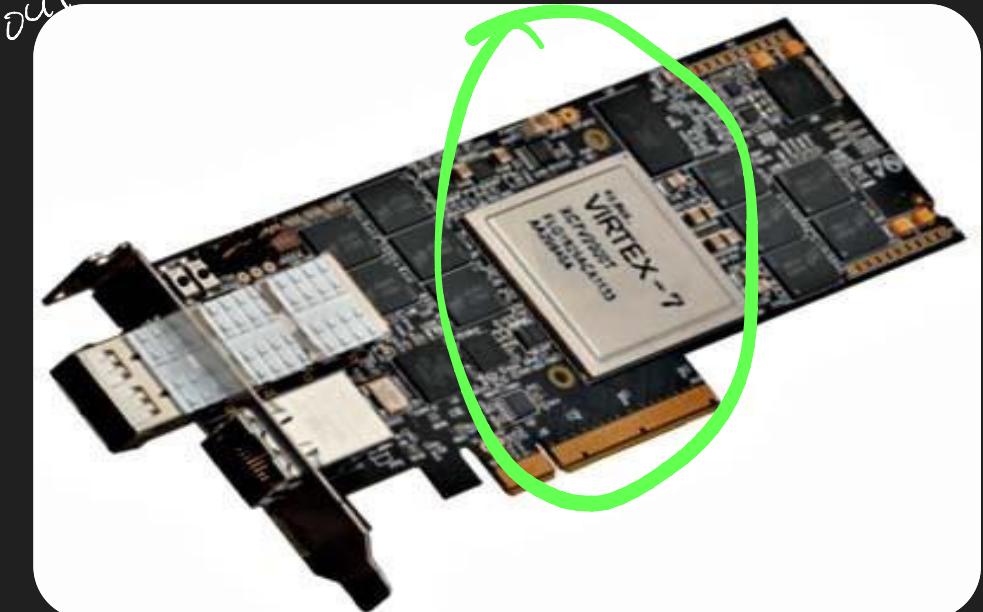
# FPGA

- Field-Programmable Gate Array.
- Can be programmed on the fly, hence Field-Programmable
- Look-up Table (LUT, /lʌt/) based, pre-placed

Flexibility



Typical FPGA Structure



Xilinx Virtex Family

# ASIC vs FPGA

	ASIC	FPGA
Performance	higher	med
Design cost	expensive (very!)	cheap
Per-unit cost	lower	higher
Design time	long	short ~ minute
Custom blocks	high customizability	low
Job perspective	Big Tech Companies	Research, non-EE field.

# Verilog

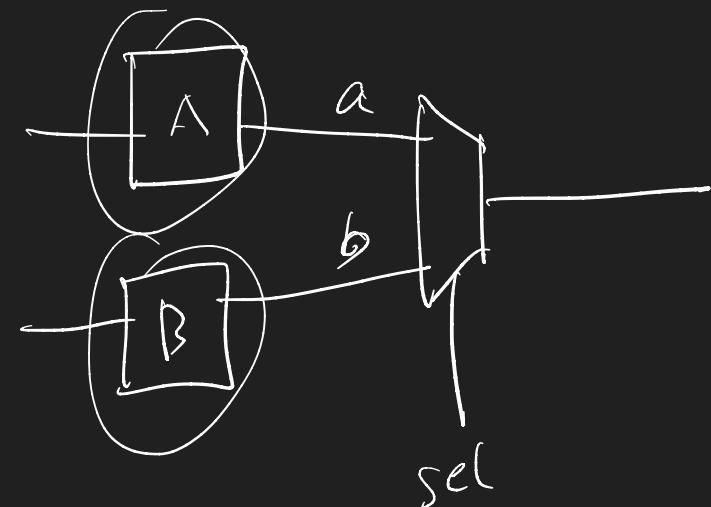


Verilog

# Hardware description language (HDL)

UFT words  
Quartus

- Describes a digital system
- Tools can synthesize the code to emulate a circuit
- Keep reminding yourself: I'm describing hardware, not writing a program
- Always sketch out your circuit, even if just at a high level (modules, ports, connections, aka boxes, texts and lines)
- Learn by hands-on practice!



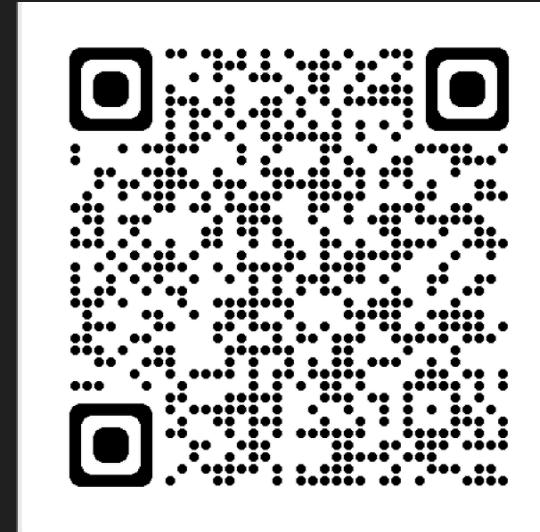
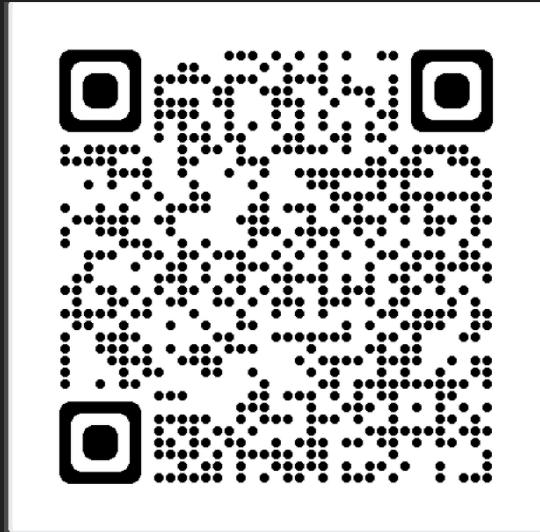
# Keep in mind

- Not a programming language!
  - Only the syntax is based on C for familiarity
  - Circuits are not programs and follow different rules
  - Think about it from a **circuit** perspective
- Not "programming a circuit"
- Writing a description

// Comment } ; at the  
end

# Verilog Syntax References

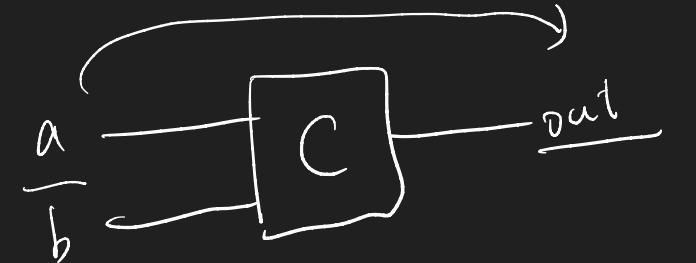
- Like any program languages, there're many good online references for Verilog HDL
  - <https://www.chipverify.com/verilog/verilog-syntax>
  - [http://www.emmelmann.org/Library/Tutorials/docs/verilog\\_ref\\_guide/vlog\\_ref\\_top.html](http://www.emmelmann.org/Library/Tutorials/docs/verilog_ref_guide/vlog_ref_top.html)



- Different sources might prefer a different flavor of Verilog syntax & formatting, we recommend that you keep **consistent with the lectures**.

# Verilog Basics

- Examples of differences

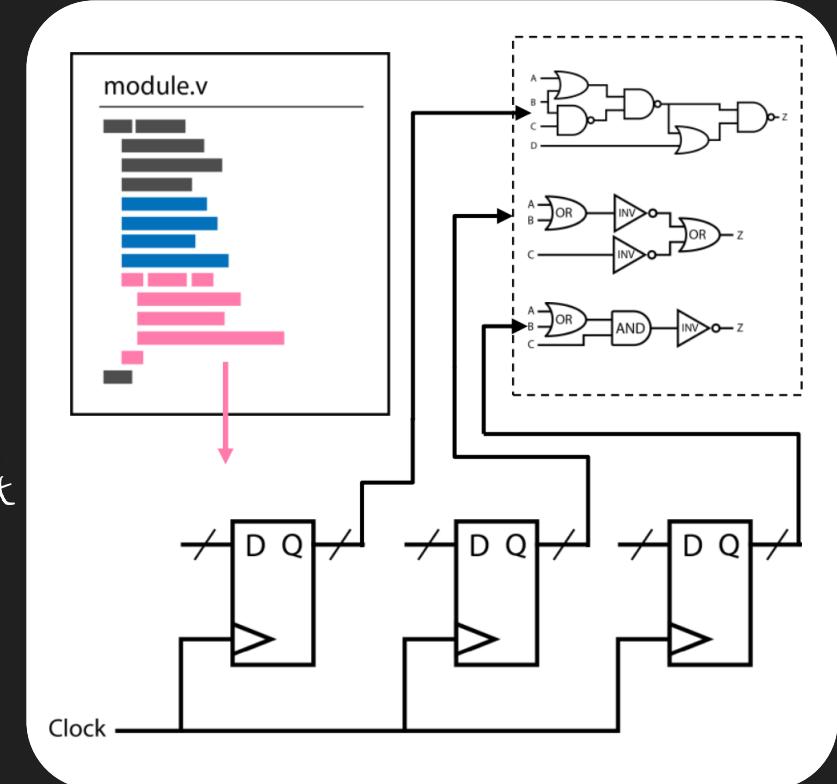
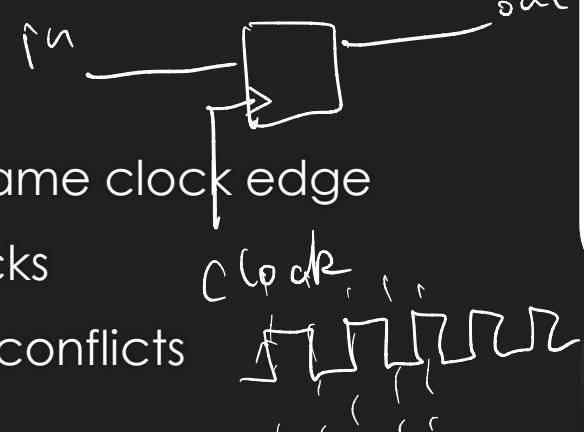


- Combinational logic

- All combinational blocks are always running in parallel
- Output updates immediately\* with input
- Just because something is assigned at a later line doesn't mean it runs later!

- Sequential logic

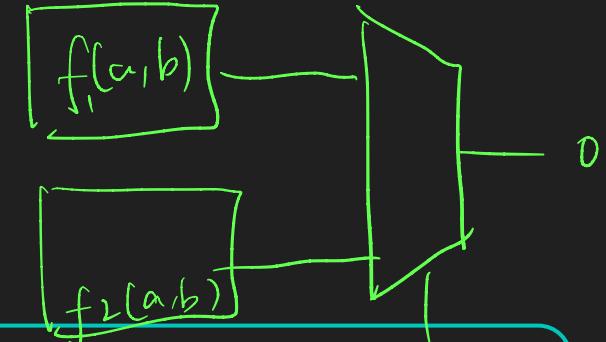
- Many registers can update on same clock edge
- Usually drive combinational blocks
- Need to be careful not to have conflicts



\*Immediate only in RTL simulation. In gate-level simulation, there will be some gate delay before the output updates

# Basic Verilog module

Two Equivalent Modules



✓ ✓

```
module module1_name (
    input [1:0] a,
    input b,
    output o
);
    assign o = (condition) ? expression1 of a
        and b : expression 2 of a and b;
endmodule
```

I/O declaration  
ternary operator (good for mux)

Condition .

```
module module1_name (a,b,o);
    input [1:0] a;
    input b;
    output reg o;
    // we call it an "always block"
    always @ (*) begin
        if(condition) begin
            o = expression1 of a and b;
        end else begin
            o = expression2 of a and b;
        end
    end
endmodule
```

trigger  
always @ (posedge clock) begin

# Basic Verilog module

Instantiating a module in another module

```
module sub_module_name (
    input [1:0] a,
    input      b,
    output     o
);
    assign o = (condition)?expression1 of
        a and b : expression 2 of a and b;
endmodule
```

module big\_module\_name (

```
    input [1:0] input,
    output      output
);
```

wire b = 1'b1; //local wire with fixed value

```
    sub_module_name my_sub(
        .a(input),
        .b(b),
        .o(output)
    );

```

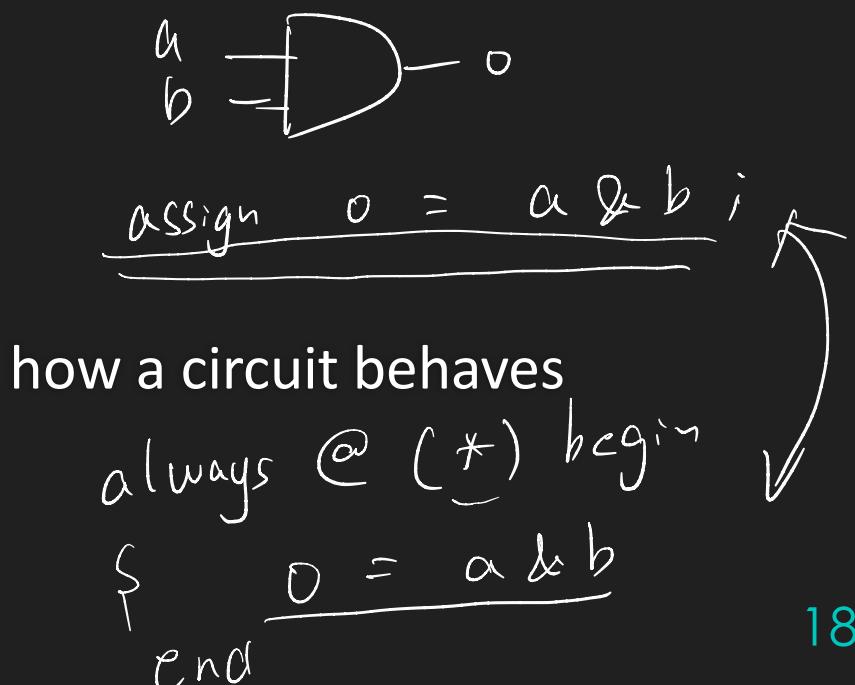
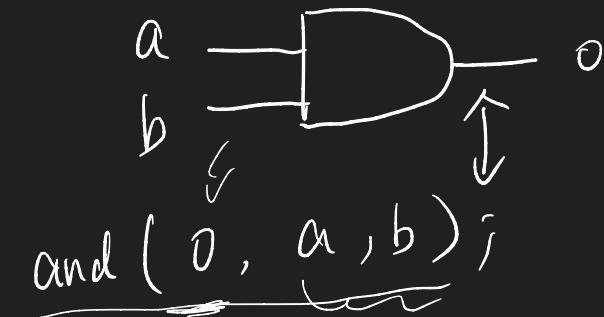
internal ( external )

```
endmodule
```

- **Structural Verilog vs Behavioral Verilog**

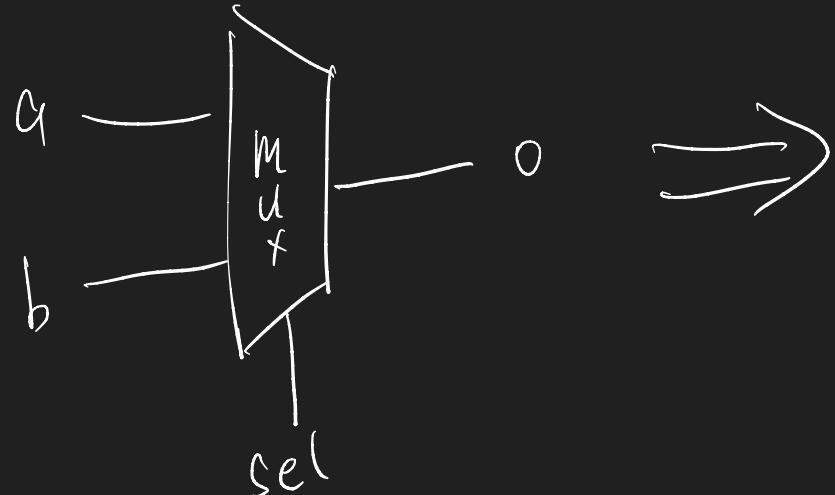
- **Structural Verilog**

- Directly describe the physical **relationship & connection** in code
- Typically combinational logic circuits build out of basic gates and module instances



# Verilog modules

- Black box design unit with a specific purpose
- Designer specifies inputs, outputs, and behavior
- Example: 2-input multiplexer



*body of code*

```
module mux (
    input a,
    input b,
    input sel,
    output o)
    assign o = sel ? a : b;
endmodule
```

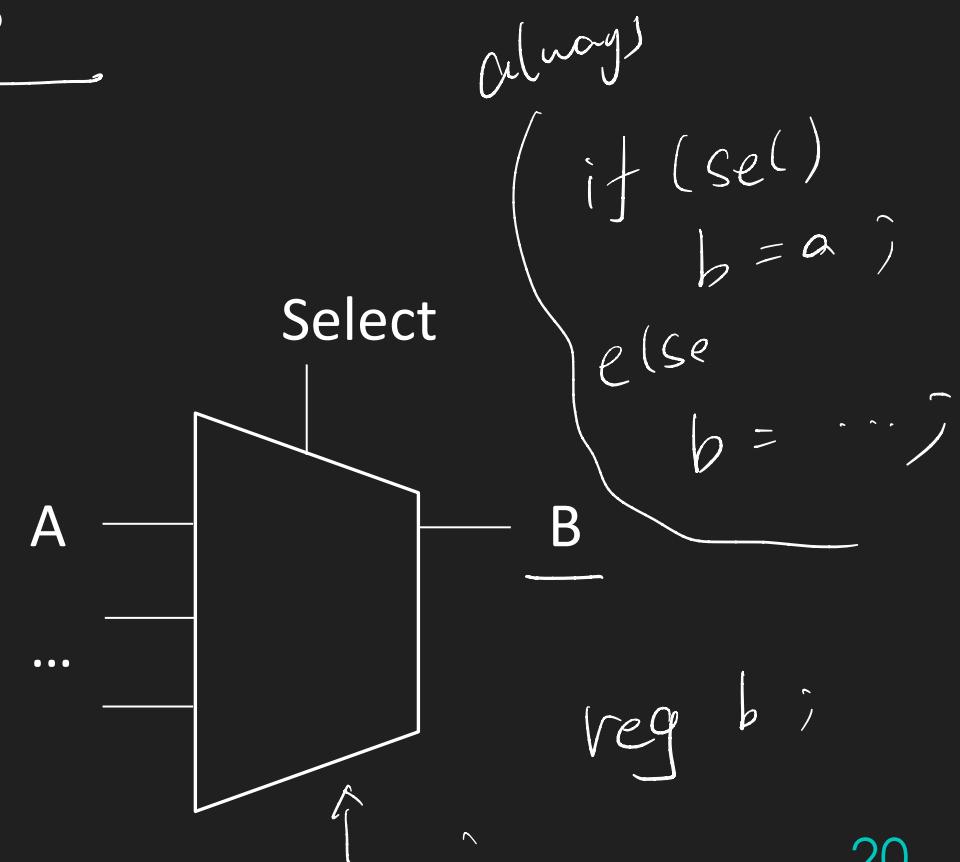
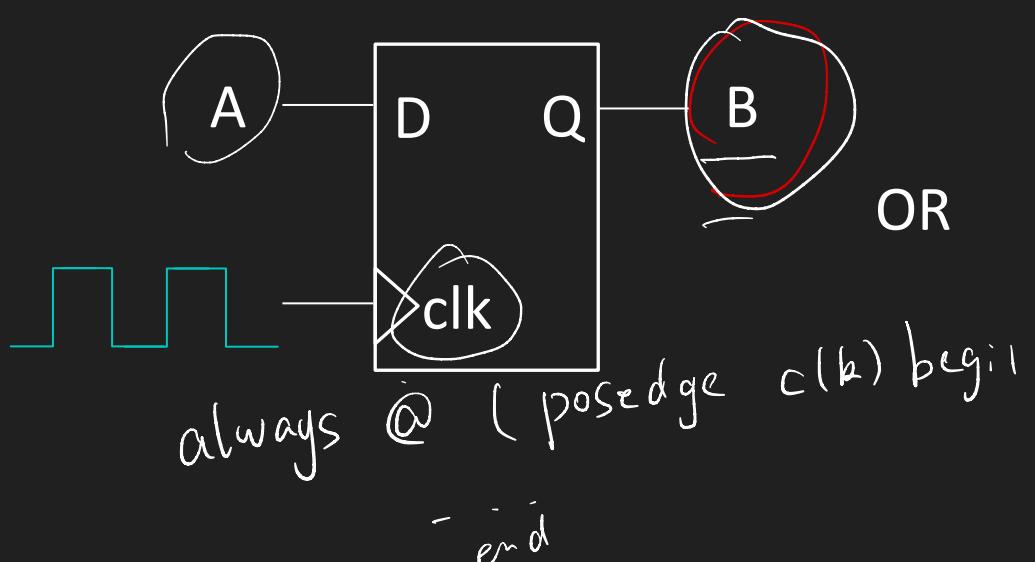
# Wire vs. Register

- wire

wire b ;  
assign b = a;



- Register (reg)



# Examples (with common syntax errors)

```
module mux2 (
    input [1:0] in,
    input select,
    output out
);
```

```
assign out = (select=1)? in[1] : in[0];
endmodule
```

Select == 1

Wire a  
a<--  
000

```
module mux2_behav (in,out,out_b)
    input [0:1] in;
    input select;
    output out;
    output out_b;
    always @(*) begin
        if (select == 1) begin
            out = in[1];
            out_b = ~in[1];
        end else begin
            out = in[0];
            out_b = ~in[0];
        end
    end
endmodule
```

endmodule

# Examples (error corrected)

```
module mux2 (
    input [1:0] in,
    input      select,
    output     out
);

assign out = (select==1)? in[1] : in[0];
endmodule
```

```
module mux2_behav (in,select,out,out_b);
    input [0:1] in;
    input select;
    output reg out;
    output reg out_b;
    always @(*) begin
        if (select == 1)begin
            out    = in[1]
            out_b = ~in[1]
        end else begin
            out    = in[0]
            out_b = ~in[0]
        end
    end
endmodule
```