

Name: **HK Rho**Collaborators: **Mai Kang**

# Homework 4

## 1. Reading and Review

- Continue referring back to Ch3, Ch4, and the Verilog Cheatsheet as needed
- Reread Ch1, 1.4.5, 1.4.6
- Chapter 5: 5.1 to 5.2.3
  - Section on Carry-Lookahead/Prefix Adders is optional

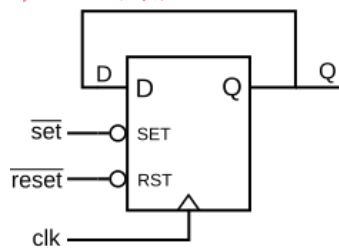
These questions are to make sure you get enough from the reading. Show your work!

### a) Review: Flip Flops

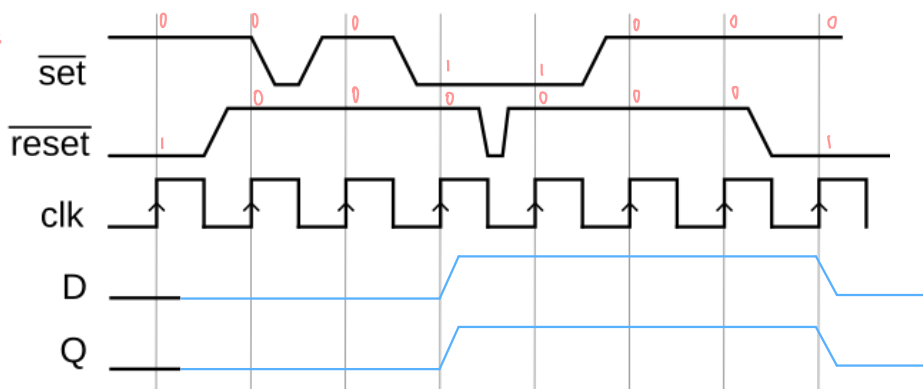
Sketch the waveforms for the following circuit under different types of set/reset. Note that for this flip flop set and reset are **active low** (action occurs when the voltage is zero).

*D Flip-Flop*

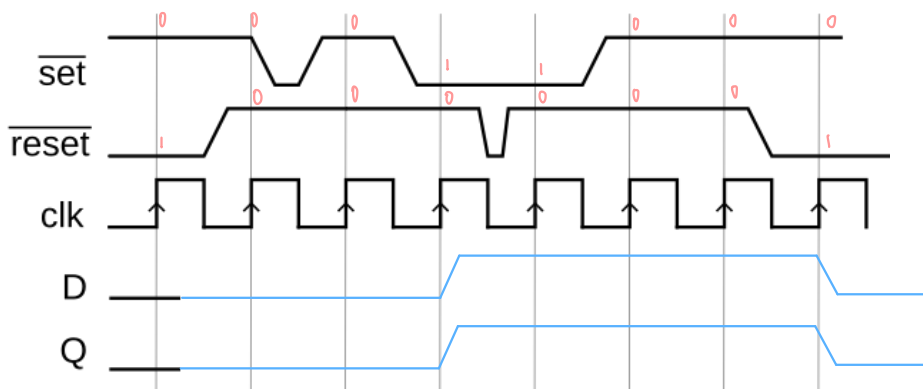
*D and Q is the same wire; D and Q has identical signals*



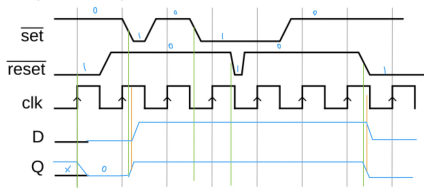
#### i) assume synchronous set/reset



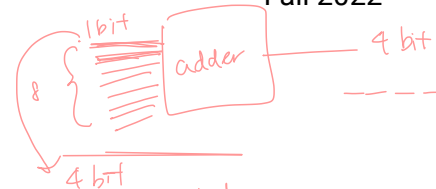
#### ii) assume asynchronous set/reset



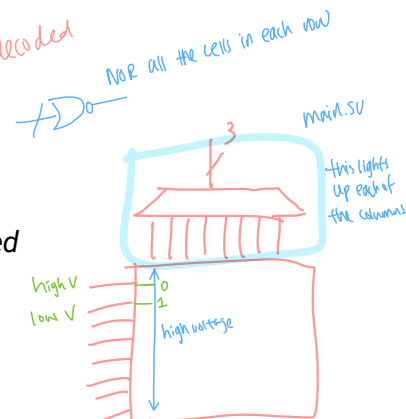
#### i) assume synchronous set/reset



*two's complement*  
*32' hX*  
*rtEX*

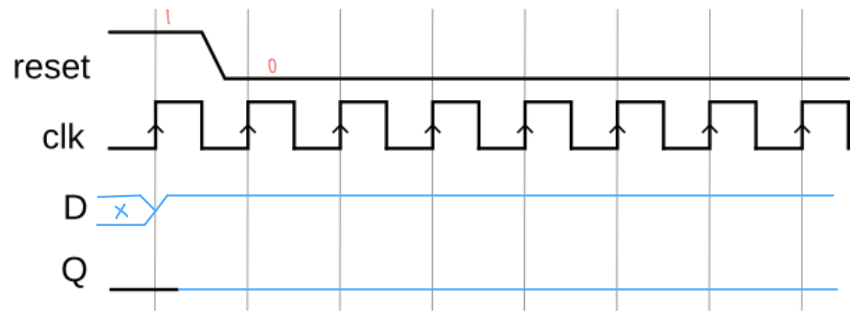
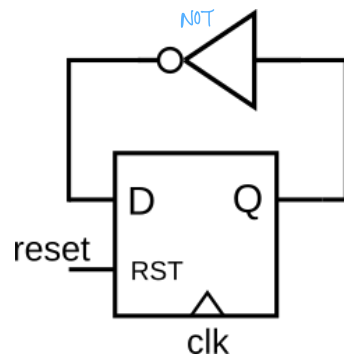


*always-comb smf = X-decoded*



*\$ random*

iii) Assume reset is **active high and synchronous**. Sketch the waveforms for D and Q below.



does d take effect on the next clk signal?  
No

## 2. Verilog Skillbuilding

### a) Verilog to Schematic

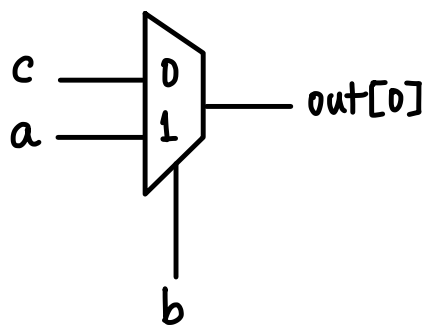
For each output, draw an equivalent gate level schematic **\*and\*** write a phrase or sentence describing what that output does in terms of the inputs.

```

module mystery(a,b,c, out);
  input wire a, b, c;
  output logic out[3:0];
  always_comb begin
    out[0] = b ? a : c;
    out[1] = (~a & b) | (~b & a);
    out[2] = c | ( b & c);
  end
  logic d;
  always_comb d = b ? a : 1'b0;
  always_ff @(posedge c) begin
    out[3] <= d;
  end
endmodule

```

out[0]:



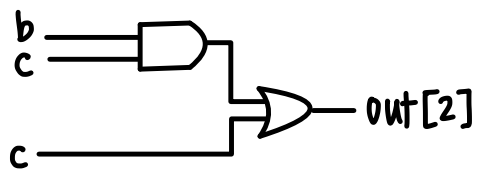
if  $b=0$ , then  $out[0] = c$   
 if  $b=1$ , then  $out[0] = a$

out[1]:



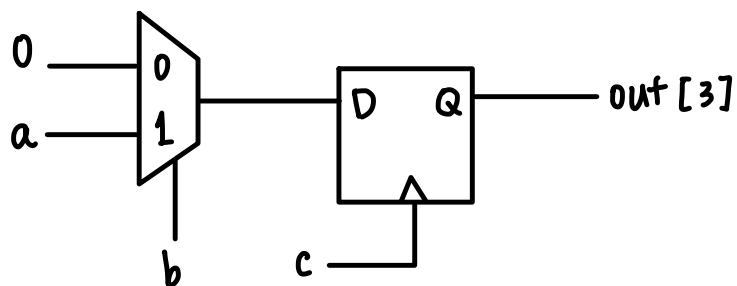
only true when  $a \neq b \Rightarrow a \otimes b$   
 XOR

out[2]:



either  $c$  or  $c$  and  $b$  has to be true  
 for  $out[2] = 1$

out[3]:

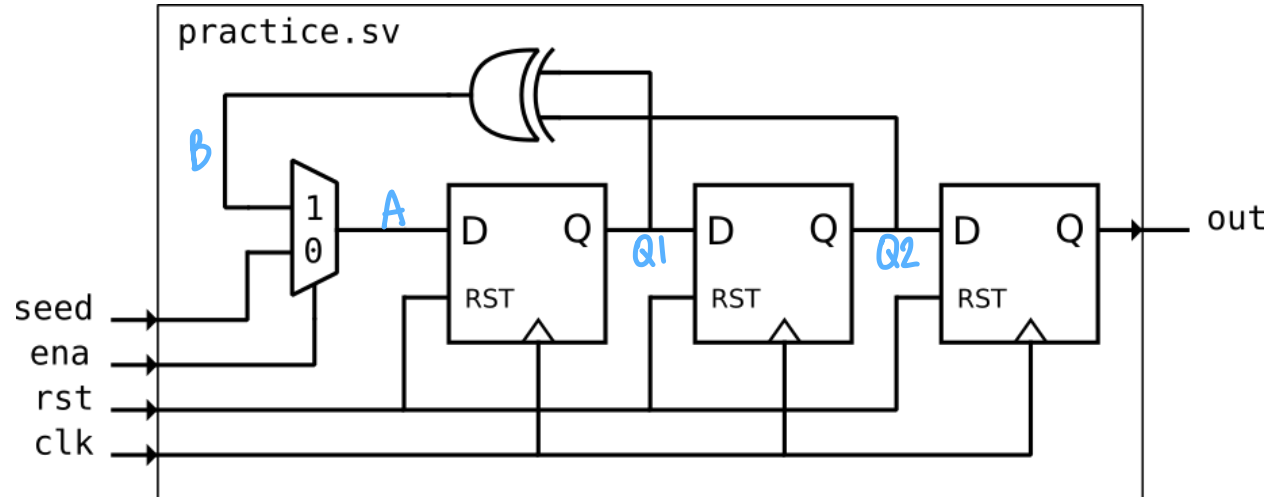


out[3] is controlled by the mux and  $c$  (clk)

C	D	out[3]
0	X	out[3]
1	0	0
1	1	1

## b) Schematic to Verilog

Create this module as part of your zipped submission. There's a testbench to go with it so you can see what it does (it's not an obviously useful circuit but it has some interesting use cases).



### 3. Combinational Design Challenge - ALU Part One

An arithmetic logic unit is a critical building block for a CPU. It does a variety of mathematical operations on two inputs (addition, subtraction, comparison, etc.). The next few problem sets will have us build up the different units inside it. For this phase we will focus on making a 32-bit adder and a 32:1 32-bit mux. The homework folder is a starting point, but you will need to create/edit the following:

- a mux32.sv file
- an add32.sv file (or an adder\_n.sv file that you can parametrize up to N=32).
- all submodule files that you used to make the above
- testbenches that give you confidence that you implemented them correctly
- a Makefile that runs your tests
- a README.md file with
  - A description of how you implemented the modules (you can include pictures or reference notes in your homework pdf)
  - A description of how you tested the mux32.
  - How to run your tests

*behavioral mux*

You may use any of the basic gates for this task (and, or, xor, nor, nand, not, mux2), and are encouraged to build up any modules you want to reuse. This cannot have any sequential logic. See the examples directory in the git repository for how to approach this!

*sequential logic's output depends on present inputs & previous output*

Bonus: there's enough code in the examples to make a ripple carry adder, but that's the slowest way to do this. Instead, implement a faster adder, like Carry Look Ahead!

### Submission

Put a pdf copy of all of your written responses in the hws/4/ folder, then run make submission to generate a zip you can upload on canvas.

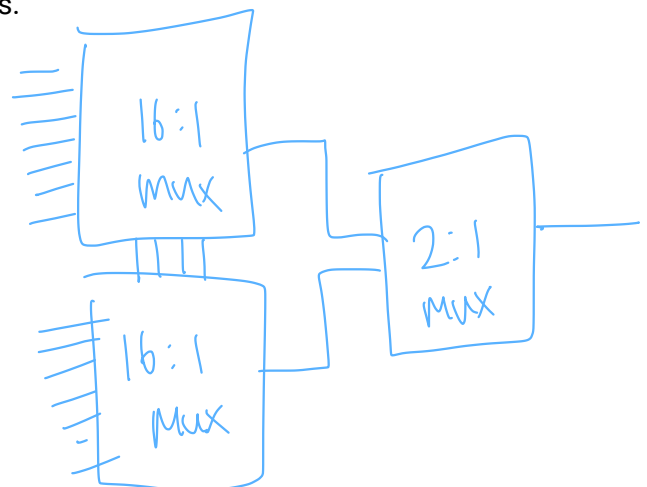
### Metadata

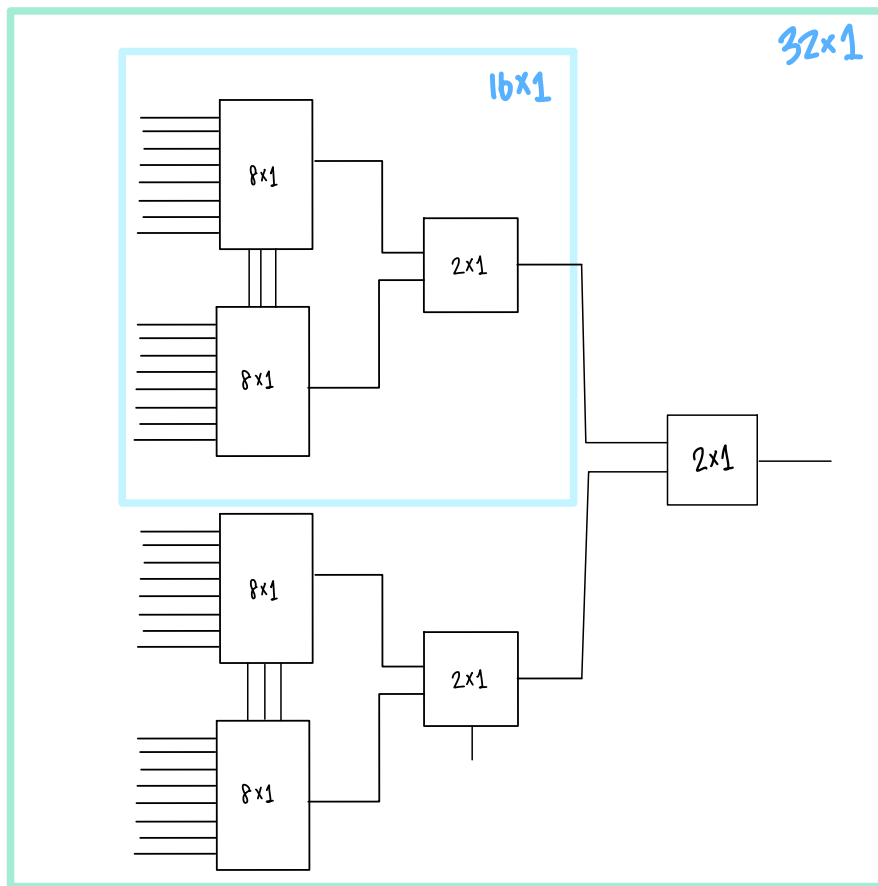
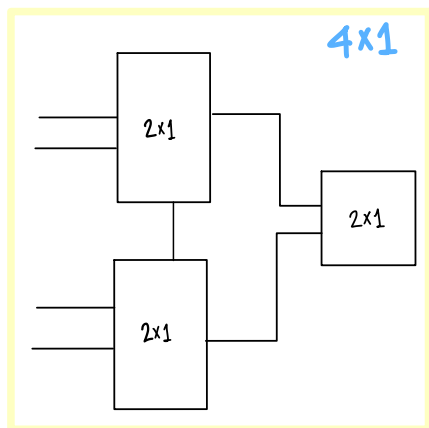
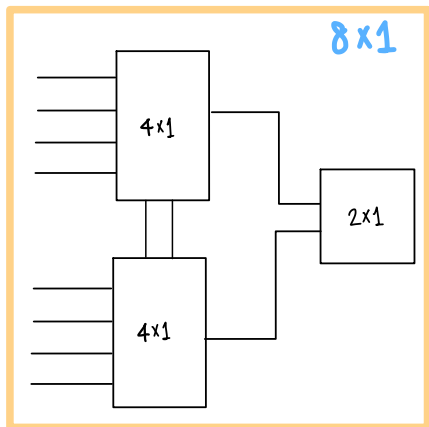
How long did the reading take you?

*~ 1 hr*

How long did the homework take you?

*~ 7 hr*





32x1

