HW 4 Comp Arch

Monday, October 10, 2022 9:57 PM

Name:

Collaborators:

Homework 4

1. Reading and Review

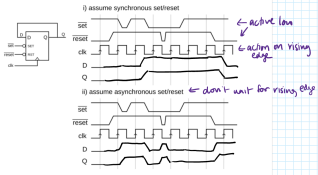
- *Continue referring back to Ch3, Ch4, and the Verilog Cheatsheet as needed*
- Reread Ch1, 1.4.5, 1.4.6
- Chapter 5: 5.1 to 5.2.3
  - Section on Carry-Lookahead/Prefix Adders is optional

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).

i) assume synchronous set/reset

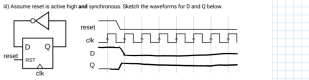


← active low

← action on rising edge

ii) assume asynchronous set/reset ← don't wait for rising edge

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



2. Verilog Skillbuilding

a) Verilog to Schematic

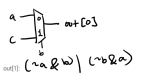
For each output, draw an equivalent gate level schematic ***and*** write a phrase or sentence describing what that output does in terms of the inputs.

```
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
```

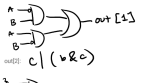
always 0?

← flip flop

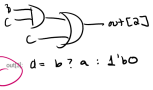
out[0]: b ? a : c



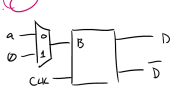
out[1]: (~a & b) | (~b & a)



out[2]: c | (b & c)

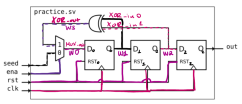


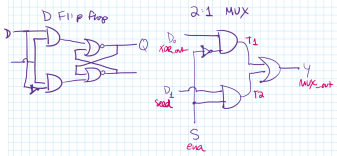
out[3]: d = b ? a : 1'b0



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).





XOR: XOR_out = XOR_in 0 & XOR_in 1

MUX: MUX_out = ena ? XOR_out : seed

input MUX_out //data input clk, input rst input

flip flop: always_ff @(posedge clk) begin

if (rst)

Q <= 1'd0

else

Q <= MUX_out; // end

flip flop 1.

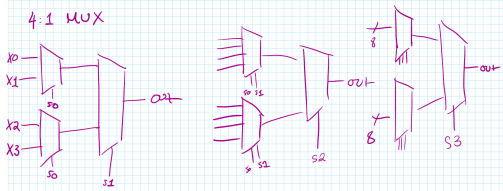
3. Combinational Design Challenge - ALU Part One

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

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!

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?

How long did the homework take you?