Name:

Collaborators:

Homework 6

The git repository hws/6 directory is the basis for this homework. Space for answers intentionally left out - include these in your single pdf in the submission zip file.

Reading

- Reread:
  - H&H 3.5: Timing, Metastability, except 3.5.6
    - Definitely read sections 3 and 4 of this guide (the rest is great too, read it if you have time!) [Synchronization and Metastability, Steve Golson, Synopsys Users Group Conference 2014]
  - H&H 5.2.5: Shifters and Rotators
  - H&H 5.5: Memory Arrays
- New:
  - H&H 6.1 to 6.4
  - H&H 7.1-7.2

0. Spiral 2 Feedback

Please fill out this [form] to give me feedback on how the last spiral through the course went.

1. Metastability and Designing for Failure

(a) Reasonable MTBFs (Mean Times Between/Before Failure)

For each of the following systems, pick an MTBF (specify your time unit!) that you think makes sense as a design requirement for the entire system. Note that for systems with constant probability of failure rates ~66% of units will have failed at least once before the MTBF has elapsed, and that in the case of a metastability failure a full system reset (power cycling) is required. Write a sentence explaining why you chose your number for the application.

Toddler Toy Piano (ages 2 to 4): A minimum of a month would probably fine, since the worst case scenario here is an upset toddler, and the uptime of a toy piano is probably less than a day in most cases.

Industrial Robot Arm: An MTBF of at least 500 years would be reasonable - this equipment malfunctioning could be dangerous for workers near it. With a large enough MTBF, it would be unlikely to fail before the equipment is replaced or upgraded.

Vehicle ADAS (Automated Driver Assistance System): 100,000 years or more - Vehicle ADAS is safety critical and shouldn't just be restarted while someone is driving.

(b) Case Study: Toy Piano

Our toy piano has 24 different digital asynchronous inputs (keys) that an eager toddler can hit very quickly. Using Equations (24) and (25) from the Synchronization and Metastability guide (page 22), and the following device parameters, determine the fastest clock speed f_c that lets the **full system** meet the MTBF you found in part (a). You can assume the following device parameters:

- τ= 175 *ps*
- T_0 = 225 *ps*
- f_d = 10 *Hz* (these are very eager toddlers)
- $t_R = t_{setup} - Tc$, *where* $T_c = 1/f_d$ *and* t_{setup} = 200 *ps*

This problem isn't supposed to be about annoying algebra, so there is a python script in the homework folder that shows you how to sweep some frequencies on a log scale to see what works (there's no analytical solution for this, so you have to guess and check).

3.632487e+08 Hz for an MTBF of .08402 years, which is 1.00824 months.

2. Combinational Review: A complete 32-bit ALU

The “thinky” element in our custom RISC-V CPU is the component that can do just about any math we need for reasonable computation. This is called an Arithmetic Logic Unit, or ALU. You should have all the building blocks for this module.

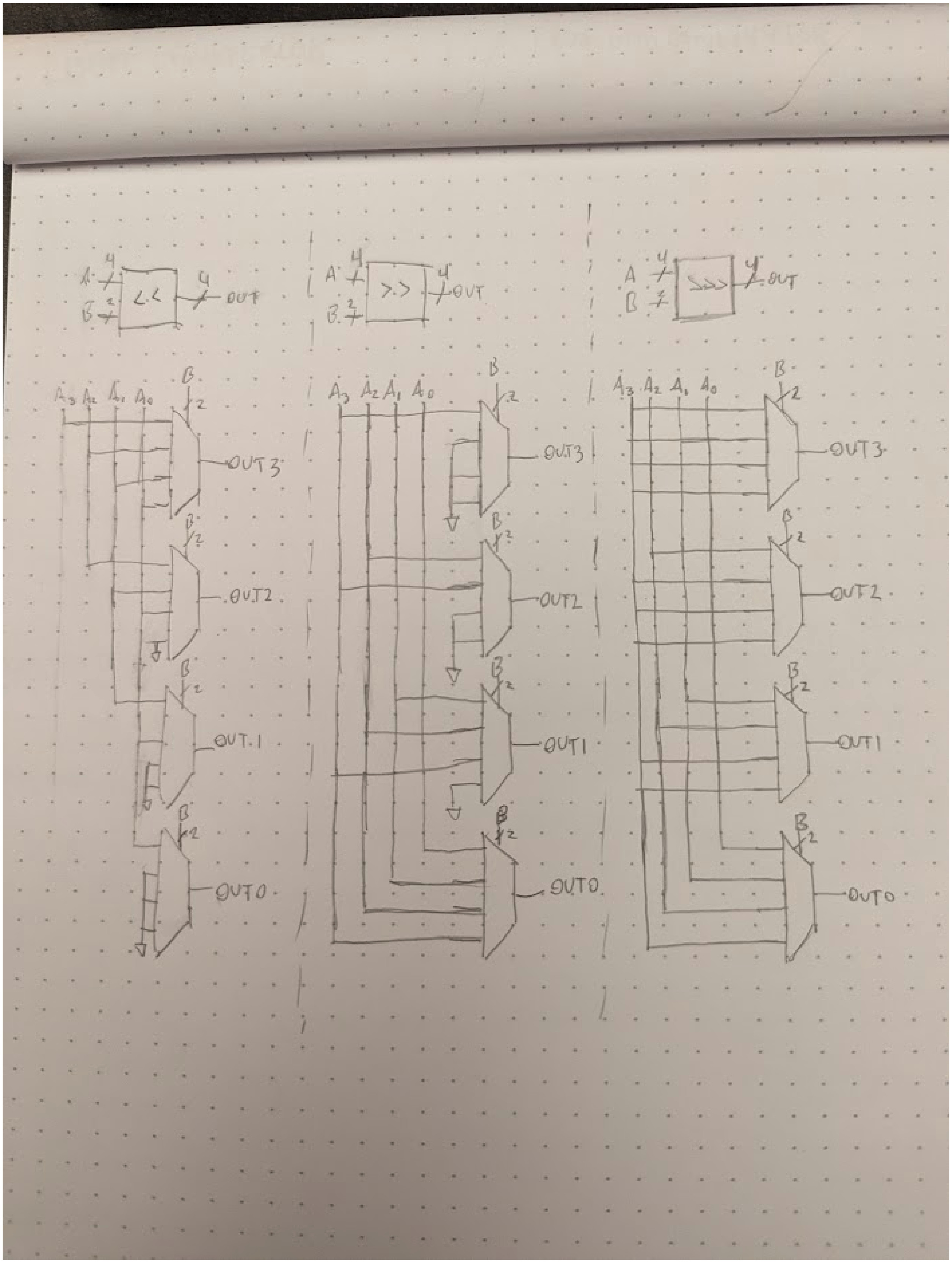
Use only structural combinational logic (no flops, no ifs/elses, etc.). That means use `always_comb` statements with only `~&|^?` operators for these modules. Working adders and muxes are included in the stub folder. There are many opportunities to be clever with submodule re-use, but remember that it is better to have a working large or slow ALU than a very fast but incomplete one. The only new features that shouldn’t be in prior examples/solutions are:

- Implement an SLL (shift left logical) operation that shifts input a to the left by b bits. The result should be padded with zeros.
- Implement an SRL (shift right logical) operation that shifts input a to the right by b bits. The result should be padded with zeros.
- Implement an SRA (shift right arithmetic) operation that shifts input a to the right by b bits. This result should be “sign extended” - that means that you need to copy the most significant bit.
- **STRETCH** - add correct overflow logic so that the ALU reports if an operation resulted in a 32 bit overflow. Only do this if you’ve finished the rest of the assignment.

You should augment the test_alu.sv example with more test cases to give you confidence that you have implemented the different operations correctly. Include descriptions **and** schematics in the top level PDF that show your approach to the shifters.

Confidence/Skills Check

With the hints from the reading this should be starting to feel straightforward in theory but a little tricky in execution (there are a lot of wires to deal with in 32 bit systems). See the solution muxes for examples of using python to auto generate long connection lists.


A
B
<<
OUT
>>
OUT
>>>
OUT
A3 A2 A1 A0
B
OUT3
OUT2
OUT1
OUT0
A3 A2 A1 A0
B
OUT3
OUT2
OUT1
OUT0
A3 A2 A1 A0
B
OUT3
OUT2
OUT1
OUT0

3. Sequential Review - Register File

Implement a 32-bit register file using **structural synchronous** and **combinational** logic (i.e. only `always_ff`, `always_comb` with basic ops `~&|^?` allowed). A working register.sv file is provided, but you will need to add the other modules together to create this file. Hint, there are some combinational modules not included in the folder that you might need for this one.

Confidence/Skills Check

This is more about understanding how a register file is supposed to work and getting better at wiring large systems together - the underlying circuitry is simple once you understand how a register file is supposed to work.