Name: Krishna Suresh
Collaborators:

Homework 5

The git repository hws/5 directory is the basis for this homework. Space for answers intentionally left out - include these in your single pdf in the submission zip file. Also this grew beyond last year's version based on a lot of recent feedback. Let me know if this is on the right track!

Reading

- ***The verilog cheatsheet was just updated to include a ton of extra tips. Re-read it!***
- Chapter 5: 5.2 to 5.2.4, 5.4, 5.5
    - Optional: 5.3 Number Systems - Highly encouraged for anyone interested in scientific computing!
    - Next Time: 5.2, 5.6
- Context: "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information" G A Miller [pdf]
- Did you know that there is a comprehensive gtkwave manual? See Q4.

1. Context Beyond CompArch - Psychology/Information Theory

Read "The Magical Number Seven, Plus or Minus Two: Some Limits on Our Capacity for Processing Information" G A Miller. This is one of the most cited papers in the field of psychology, but it has tremendous implications to appropriate design of digital systems.

a) Describe at least one situation in which you had to use or measure more bits than were useful to a problem. When storing a float variable with 5 decimal points in a double.

b) Have you ever had a situation where you didn't use enough bits?
When tracking my calendar, I set events to 15 min increments instead of the actual time.

c) How many bits did you use to answer the previous question?
85 characters → ASCII (8 bits per) $8 \cdot 85 = 680$ bits

d) In your own words, what is the difference between a bit and a chunk?
A bit is a single decision made between 2 evil choices whereas a chunk is a combination of related bits which make up a full idea.

e) How are the generalizations from this paper still applicable half a century later? Alternatively, what do you feel no longer applies? The principles of bits in the context of information theory and digital computing are just as relevant today since the idea behind information are always useful. But the generalization of human processing has evolved to be supplemented by more data and is more nuanced.

f) (optional) This is a paper I feel everyone should read[1], do you have an equivalent must-read scientific article?

[1] I have sent this to family, friends, and colleagues. Amazingly I'm still on some of those group chats.

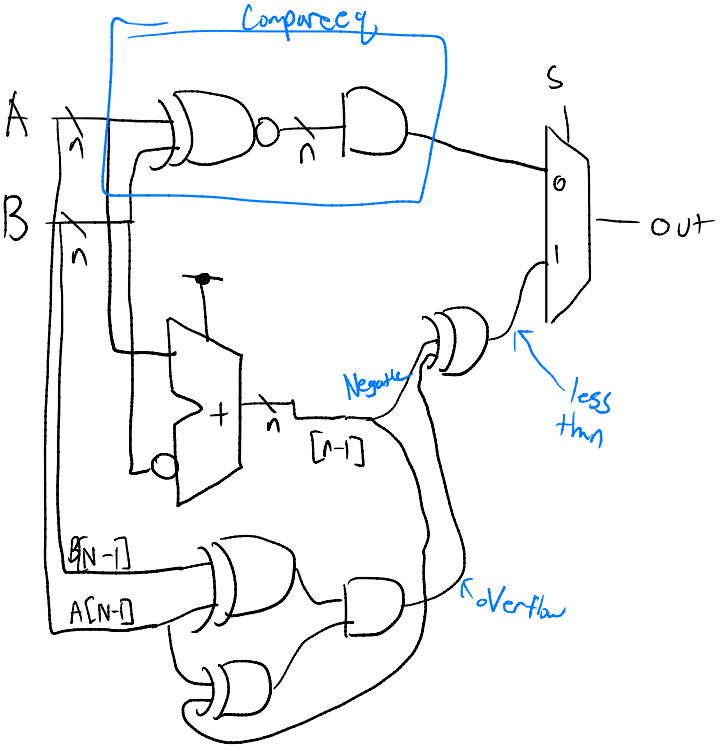
2. Combinational Review: ALU Part Two

Use only structural combinational logic (no flops, no ifs/elses, etc.). That means use `always_comb` statements with only ~&|^? operators for these modules. A working adder is included in the homework folder.

- Implement a compare_eq module that compares two n-bit numbers and outputs high if they are equal, low if they are not.
- Implement a compare_lt (less than "<" operator) that can support up to 32-bit inputs, that outputs high if a is less than b.
- Make sure the Makefile can still run the test (include any of the submodules in the target!).
- Augment the test_comparators.sv example with more test cases to give you confidence that you have implemented the two comparisons correctly.
- Descriptions **and** schematics in the top level PDF that show your approach to the two comparators.
- Bonus: Use the cheatsheet to load in test vectors from a memh/memb file. Update make test* and submission accordingly. *You'll get a guided version of this next homework.*

Confidence/Skills Check

This should feel straightforward at the block diagram level, but possibly still tricky in terms of execution in SystemVerilog. Contact an instructor ASAP if you don't know how to start, reach out to peers after ~15 min of (re)reading if you feel you can't sketch a block diagram, then follow up with more instructor time if you still feel shaky. Only start writing HDL after you have a block diagram that is at least vetted by a peer! Reach out to instructors after ~5-10 min of debugging HDL, Makefile, verilator, iverilog, or gtkwave issues.



Compare eq:
We know that for each bit of A and B the corresponding values must be equal. This is the same as NXORing each bit and then ANDing.

eq:

00	1
01	0
10	0
11	1

Less than:
We first want to find -B which is equivalent to $\bar{B}+1$ when B is in the two's complement form. Then when we add A and -B we get their difference. Next we can look at the MSB to see if the result is positive or negative. Finally we need to check for the case where an overflow occurs and compute L with N⊕V.
N: negative sum
V: Overflow (opposite signs → must be same result as A)

3. Design Challenge/Lab Prep

Implement the following using **synchronous** and **combinational** logic (i.e. only `always_ff`, `always_comb` allowed). You can either go behavioral (ifs, elses, ==, <,>, etc) or use the structural modules from the prior question/labs. Using `generates/for` loops is **not allowed**. For each module **you must include a schematic sketch in your top level PDF for full credit!**

- [straightforward] pulse_generator.sv
- [getting a little tricky] triangle_generator.sv
- [putting it all together] pwm.sv

Specifications for the modules are in the provided stub files. This is a lot to do, but if you manage time well **and ask questions early** you'll find that (a) all of these modules use the same tricks of combining counters, comparators, and registers (with a simple 1-2bit FSM), and (b) each module leverages techniques from the previous one to do more (this doesn't mean you should re-use the previous module, more that the same ideas apply).

Confidence/Skills Check

Getting to the block diagram level for each of these modules is the hardest part. I recommend ~15 min. trying to come up with a block diagram solo, ~15 min. working with peers (please cite them), and then getting instructor help before proceeding. Once you have a block diagram (labeled with bus widths, good wire names, etc.) don't spend more than ~5 min. or so stalled on syntax/verilog issues. Get verilator linting in your editor asap, it will save you a ton of time! See the next question

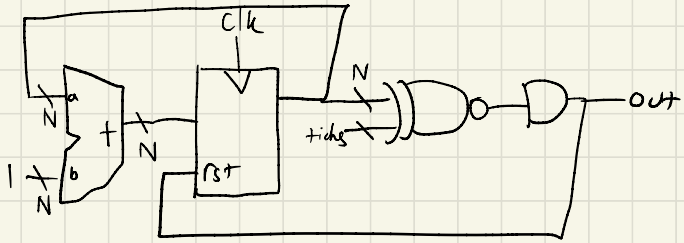
4. Tool Usage - gtkwave/Makefile practice

A poor engineer blames their tools. A garbage engineer doesn't even use them correctly[2].Good engineers RTFM (Read The Full[3] Manual).

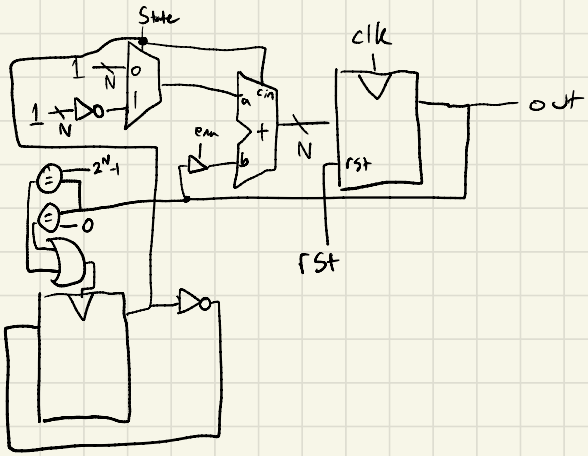
I've been noticing a strange reluctance to use gtkwave to analyze results. Printing ($display-ing) is barely sufficient for combination logic; sequential demands more. All the tools have easily searchable manuals, this question is about forcing you to RTFM. The homework folder includes an example of using a Makefile with a `waves_*` target that you can reverse engineer, but to get full credit you should have added Makefile targets `waves_pulse_generator`, `waves_triangle_generator`, `waves_pwm` that call your saved gtkw configurations - when the instructors call make waves_* we should get the same waveform (with comments!).2
For special fun - right click a waveform and select the analog option for the triangle wave generator.

[2] The unofficial ending to that joke is that a professor/engineer just destroys the tool, making more work for the technicians. I was gonna make you guess which institution taught me this, but the answer is all of them. Literally everywhere I've worked or interned or contracted at. And it applies to software too! Listen to the techs, they know more than you ever will, and appreciate the vice versa.
[3] I know, the F doesn't stand for Full.

Pulse Gen:



Triangle gen



PWM: