Final Exam (version 1.0) [100 points]

Deadline: 11:59PM PST, Friday Mar 18, 2022)

Live URL: https://docs.google.com/document/d/1_Ei5_sYEdxEXVEq5gyz2RItlp6Nx2rHuwAcBSP9daXQ

NOTE:

1. **This final exam is to be done individually using Logisim Evolution v3.7.2.**
2. **Submission Procedure: You need to submit your solution online via Gradescope. Please read the section titled "What to submit?" at the end of this document.**
3. **Late Submission: There is no late submission.**
4. **Logisim:**
   - **You must use the version of Logisim made available for download on Piazza. As you edit in Logisim, save frequently and also make backup copies of your design file.**
   - **We will grade using the top-level circuit (it is labeled DUT in most cases), but you are free to create additional sub-circuits which you use in DUT.**
   - **I highly recommend using Logisim's tunnels which make your circuit cleaner and allow for easier debugging; they allow you to move a value from one part of the circuit to another without having to drag a wire all the way across. You can create tunnels for all the inputs and their complements. Instead of hooking up the inputs directly to the gates, you can hook up using tunnels instead.**
   - **Do not rename or move any input or output pins, or add any additional pins. Doing so will in all likelihood cause the Autograder to fail and result in points to be deducted (see below).**
   - **Testing is as critical as design, and in particular, make sure to test for edge cases. For manual testing, you can use the hand tool and click on the input pins to change their values, which will propagate to the rest of the circuit. You can reset the simulation back to the start with Ctrl-R to test again after you make changes. You can also use the Test Vector feature to automatically test your circuit.**
   - **The Test Vector and Command-line Verification features in Logisim Evolution allow you to test your circuits. You can read more about them in the Logisim Evolution User's Guide. You are also free to exchange test cases (i.e. expected input-output pairs) or collaboratively create them with other students in the class, but are not allowed to share testbench (i.e. circuits designed to test) as that is viewed as part of the work.**
   - **You must use designs relying only on gates and within specified component limits that are permitted for each problem. Failure to do so will result in a zero score.**
   - **Save frequently and commit frequently! Try to save your code in Logisim every 5 minutes or so, and commit every time you produce a new feature, even if it is small.**
   - **As you work on your Logisim design, make use of additional subcircuits, tunnels, piobes, etc. to make your design modular so that the design is easier to understand and debug.**
5. **Please comply with all the instructions as failure to do so can cause Autograder to not be able to test your design. If we have to manually grade because your failure to comply with instructions (e.g, file names, pin naming, etc.) required us to fix, we will deduct 20% of the maximum points from each question where the problem occurs. Note that this does not apply to design bugs - there is no manual grading.**
6. **Grading: Please read the section titled "Grading" towards the end of this document.**

7. **You will need to submit a report as well, using the template at https://docs.google.com/document/d/1g2TwlxFcqt9d9W-zreTWJjueQkD67KuKbYCc8MVcV18 (make a copy, edit, save as PDF, and then upload on Gradescope)**
8. **Version History:**
    - **1.0: initial release**

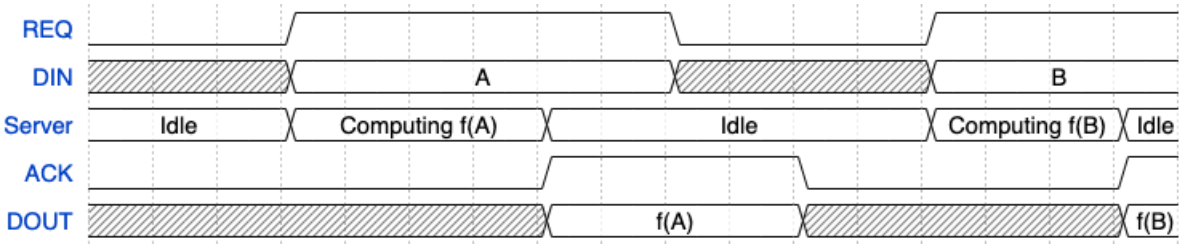
Task: Cube Root Computer

The goal in this task is to design a synchronous positive-clock-edge triggered sequential system that takes in a 2's complement number $DIN[23:0]$ as input and returns as a 2's complement number $DOUT[8:0]$ the real-valued cube-root of $DIN[23:0]$ rounded towards zero.

E.g. an input of $4194303_{10} = 001111111111111111111111_2$ will result in an answer of $161_{10} = 010100001_2$, and an input of $-4194303_{10} = 110000000000000000000001_2$ will result in an answer of $-161_{10} = 101011111_2$.

You can think of your system as a server computer that is responding to requests from an external client as in the figure below (note that the figure omits the clock and reset signals).

The client interacts with the cube-root server via a four-phase request-acknowledge handshake which involves a signal called REQ that goes from client to server, and a signal called ACK that goes back from the server to the client. Normally REQ and ACK are both 0. To initiate a new transaction with the server, the client will put the necessary inputs for the server on DIN and set $REQ = 1$, The server, upon seeing $REQ = 1$will know that there is a new task, and start computing using the values on DIN. When done, it will put the results on $DOUT$and also set $ACK = 1$. The client upon seeing $ACK = 1$ will know that the server is done, and read $DOUT$, and then set $REQ = 0$ to tell server that it is finished reading $DOUT$ and the server acknowledges by setting $ACK = 0$. Clearly, client must keep DIN stable while $REQ = 1$ and the server must keep $DOUT$ stable while $ACK = 1$. The figure below details this handshake (in the figure one an consider $f()$ to represent the computation of the real-value cube root).



Four-Phase Handshake (Return-to-Zero Handshake)

The following pseudo code describes the handshake from the perspective of your system:

```
Initial State:   REQ and ACK are both 0 after reset
Step 1:          While True:
Step 2:            Wait for REQ==1               // wait for new DIN
Step 3:            DOUT = RealCubeRoot(DIN)      // this may take variable # of  clock cycles
Step 4:            Set ACK=1                     // tell client that DOUT is ready
Step 5:            Wait for REQ==0               // wait for client to acknowledge reading DOUT
Step 6:            Set ACK=0                     // tell client that we are ready for next input
```

Your design should follow the following specification. You would likely find it useful to consider splitting the task into a controller FSM and a datapath.

Clock: CLK
Inputs: $RST,\ REQ,\ DIN[23:0]$
Outputs: $ACK,\ DOUT[8:0]$
Starter Logisim File: final.circ
Main Circuit: DUT
Allowed Logisim Modules

- From Gates library: only NOT, and NAND.
- From Memory: only D Flip-Flops and Register from Memory
- From Arithmetic: at most 2 Multipiers, at most 4 Adders, at most 2 Subtractors, at most 2 Comparators. No other components from Arithmetic.
- From Plexers: at most four Multiplexers. No other components from Plexers.
- From wiring: any except Transistor, Pull Resistor, POR, Transmission Gate, Power, and Ground.
    - Note that while you can use Pin in any subcircuits that you define, you must not add any Pin to DUT as that will cause failure with the Autograder.
- From Input/Output: Any.
- None from any other library.

Restrictions

- At most 1000 components total (your design will have a lot fewer; this is just a conservative limit and designed to prevent students who seek to game things by using excessive combinational logic). Logisim components such as pins, tunnels, wires, probes, splitters, LEDs, buttons, etc. that are used for wiring and I/O are excluded. The intent of the problem is that you should not attempt to build complex datapath blocks using simple gates. Please do not attempt to bypass this spirit of the problem.
- Since the system is required to be synchronous, you should not use any asynchronous inputs in any sequential elements that you use - e.g. do not use the asynchronous set or reset signals on flip-flops.

Cost of Logisim Components

- w-bitwidth NOT: $2 \times w$
- w-bitwidth NAND with n inputs: $2 \times w \times n$
- D Flip-Flop: 18
- w-bitwidth Register: $20 \times w$
- w-bitwidth Adder, Subtractor, Comparator: $28 \times w$
- w-bitwidth Multiplier: $28 \times w^2$
- w-bitwidth 2^s -to-1 Multiplexer: $6 \times w \times 2^s + 2^s + (s > 1\ ?\ 2 \times s \times 2^s : 0)$
    - For a w-bitwidth 2-to-1 Multiplexer, the cost expression simplifies to $12w + 2$.

Desired Behavior:

1. The active edge of CLK is the rising edge unless otherwise specified.
2. RST is meant to be used as a synchronous reset signal. Whenever the external world wants to reset the system, it will assert $RST = 1$ for at least one rising edge of CLK and then make $RST = 0$ to start normal operation. Note that reset may occur multiple times as the system runs, and even in the middle of a handshake.

3. The external world does not care about the values of output signals on clock edges before the first clock edge after the start at which your system sees $RST = 1$.
4. If $RST = 1$ at a clock edge, then the client must see $ACK = 0$ at the next clock edge and likewise your system is assured to see $REQ = 0$ at that clock edge (i.e. the next clock edge).
5. Your system should produce ACK and $DOUT$ in accordance with the functionality described earlier.

Desiderata:

- Correct functionality
  - If your design fails our tests, your score will depend on the report only.
  - Please test carefully, particularly for edge cases.
- Low $Area \times Delay$
  - $Area$ is measured by adding up all the component costs (you can get this from the Autograder when you upload)
  - We will measure $Delay$ in terms of average # of clock cycles from REQ becoming 1 to ACK becoming 1 while with correct $DOUT$)
    - You can obtain it during your testing. Think about the worst case.
  - Note that a simple algorithm that searches for solutions in a brute force manner will result in a $Delay$ of a little bit over 200. But smarter algorithms can reduce the $Delay$ to around 10 but at the cost of a more complex design.
- Good architecture and clean design, testing strategy, quality of explanation, etc.

What to Submit:

- Your design in final.circ (using the starter version of this file)
- For every control FSM in your system, you must provide a .gv file gv (state graph using https://edotor.net or https://sketchviz.com), naming them as *final_1.gv*, *final_2.gv*, and so on. T
- A report final.pdf using the template (see URL under instructions). The report will include (see template for details):
  - For every control FSM, the state transition diagram, the state transition table, assignment of states to bitvectors
  - An overall block diagram showing how the various control FSMs interact with each other and with the datapath, as well as a high-level via of the various components in the datapath.
  - A block diagram of your datapath and how it interacts with the various control FSMs
  - Description of testing strategy.
  - Analysis of $Area \times Delay$ product
  - Note that when grading the report, we will consider not only correctness but also efficiency (e.g., how many states did you used), clarity, and lack of unnecessary verbiage.

Grading

You will get a score of zero if any gates other than those allowed for a problem are used. You may use (and in fact encouraged to) use subcircuits in Logisim to create nice hierarchical designs. Problem scores will be split as follows:

Functionality:	50% (this comes from Autograder)
$Area \times Delay$:	20% (this comes from Autograder)

Report: 30%

Functionality Score: We will subject your sequential circuit to two equally weighted testing sessions, each with multiple equally weighted test runs where each test run startis with a reset and has multiple transactions via the handshake protocol. We will test both for correctly following the handshake protocol as well as correct computing of the value. Note that computing correct value at an incorrect time gets no credit. The first error will cost 20% of the functionality score, and every subsequent violation that we check for will result in a reduction. Specifically, if we do n checks and your design fails k of them (naturally $k \leq n$) then your score will be $MaxScore * (k == n ? 1 : 0.8 * (k/n))$. Note that certain errors are of a nature that prevent continuation of a test session (e.g., if your design violates the protocol) and then you will get a zero score on that session. If your circuit hangs, we will have to abandon testing entirely.

Area-Delay Score: Designs that have functionait errors will get zero scores on this metric. For correctly functioning designs we will give scores as per the rubric below. Let α be the value of $Area \times Delay$ that is min(our design, best submission).

- $Area \times Delay \leq 1.1\,\alpha$ will get full score for this category
- $Area \times Delay = 2\,\alpha$ will get 80% of the score for this category
- $Area \times Delay = 4\,\alpha$ will get 25% of the score for this category
- $Area \times Delay = 8\,\alpha$ will get 12.5% of the score for this category
- $Area \times Delay > 8\,\alpha$ will get 0 points
- Scores will be linearly interpolated in between the above thresholds.

What to submit?

In order for your design to work with our autograder, you must create your design by modifying the Logisim starter file final.circ provided in the final_files.zip archive provided on Piazza. You will see two subcircuits: one call DUT (stands for Design-Under-Test) and the other called TB (stands for Test Bench). You should put your design in the subcircuit DUT. As you do so, you are free to create additional subcircuits. You may also find it useful to read the Logisim user guide to familiarize yourself with advanced capabilities, such as subcircuits and tunnels which help make the schematics less cluttered. You MUST NOT rename or reposition or delete or add pins in any way in the subcircuit DUT, and must not modify the TB circuit in any manner. Also, you may use the TB subcircuit, which instantiates the DUT subcircuit, to test your design while making sure that you have not accidentally modified the I/O pins. After you are done, create and upload a **final_files.zip** archive to Gradescope consisting of your modified **final.circ** file as well as the **.gv** files for every control FSM (naming them as final_1.gv, final_2.gv, and so on). When you upload to Gradescope, the autograder will do some basic sanity checks and report any missing files but not run any actual tests. It is your responsibility to devise test cases to test your circuit. You will submit the PDF report under a separate Gradescope assignment. Both submissions should be done by the deadline.

Partial Credit and Optional Video

It is possible that your design does not work and fails most or even all of the tests. If so, we will attempt to give partial credit when we grade the report, but can do so only if we can understand your design and what parts worked and what didn't with reasonable effort. Your functionality score will not change but you may get partial credit on elements of the report. You can help in this by making your Logisim design clean

and modular, and by putting information in the final.pdf report file that explains what worked and what didn't, and document with timing diagrams exported from Logisim's Timing Diagram. If you prefer, you can also prepare a short video explaining whatever aspects did work and showing Logisim's timing diagram output. Please limit it to 5-10 min and upload the video file to some place where we can access it via the browser (e.g. YouTube). Provide the link to your video in the final.pdf report file. Ensure that the link is publicly accessible. Please note that the video is entirely voluntary and there is no credit given for that and we certainly do not expect them. We will not look at it unless your design totally bombs and we are left with no basis to grade. It can be useful if you know your design doesn't work and you want to tell us about the pieces that do.