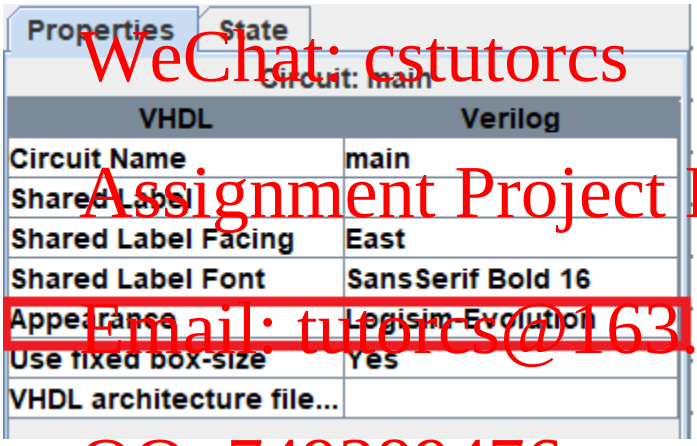
Final: Cache

1. Use the version from the class Google Drive of Logisim Evolution. Other versions may not work correctly
2. Do not renam e. If you do so you will automatically fail the tester when you sub
3. Put your solut into implementation subcircuit
4. Do not renam subcircuit anything else. If you do so you will automatically ou submit.
5. Do not chang he implementation subcircuit from what it is set as. Doing so will cause you to automatically fail the tester when you submit.
    a. That is this field right here



6. Do not move the pins inside of the implementation subcircuit as that affects the appearance of the circuit on the outside as you saw in discussion. Doing so will cause you to automatically fail the tester when you submit.
    a. If you want to "move the pins" instead connect tunnels to the pins and move the tunnels around.
7. Do not name any of the subcirucits in your solution main. Doing so will cause you to automatically fail the tester when you submit.
8. You **can** create as many other subcircuits as you want in your solution. Just make sure your solution ends up in the implementation subcircuit

Update Your Version of Logisim Evolution

A new release of Evolution has come out since class started. We are upgrading to the new version because it fixes a bug that prevented loading memory image files. We need this feature so the update is mandatory. Again you find the version of Logisim Evolution you need on the class Google Drive.

Warning!!!

This project will take a long time. Expect to spend at least 30 hours on it.

Components

For all problems in th y only use

- All of the com g
- All of the com **EXCEPT** for Controlled Buffer, Controlled Inverter, PLA
- All of the components under Plexers
- All of the components under Arithmetic
- All of the components under Memory **EXCEPT** for **RAM, ROM,** and **Random** Generator

Unless a problem specifies otherwise.

Problem Specification

Implement a Cache that meets the specifications below.

Cache Specifications

Parameter	Value
CPU Address	10 bits
Cache Data Size	24 Bytes
Number of Sets	2
Number of Ways (Blocks per Set)	3
Block Size	4 Bytes
Write Policy	Write Back
Eviction Policy	LRU

Inputs

Input Pin	Size (bits)	Description

CpuAddress	10 bits	The address the CPU wants to read from or write to
CpuRead	1 bit	1 If the CPU wants to Read and 0 otherwise
CpuWrite	1 bit	1 if the CPU wants to Write and 0 otherwise
CpuWriteValue	8	the CPU wants to Write. Will only have a ul value is CpuWrite = 1
LineFromMem	32	nts of the block you requested to read from memory

All of the inputs will o e **SINGLE** clock cycle that your Cache says it is Ready. This means that you will have to save your inputs if you need them beyond that first clock cycle.

The only **exception** to this is LineFromMem. It will be valid as long as you say you want to read from memory. If you aren't reading from memory LineFromMem will be EEE… and Red (an Error). Don't worry about this. It is normal behavior. As soon as your request to read from memory it will be the value at MemAddress.

Outputs

Output Pin	Size (in bits)	Description
Ready	1 bit	1 if your Cache is ready to start a new request. If you are ready to start a new request it means you must have finished the old request. So at this point in time, your output will be checked.
DidContain	1 bit	1 if your Cache contained the value the CPU asked for and 0 otherwise. Checked when Ready == 1
ByteRead	8 bits	The value at the address the CPU requested to read. Checked when Ready == 1 and the last request the CPU made was a read.
MemAddress	8 bits	The address of the **block** you want to read from memory or the address of the **block** that you want to write Line2Memory back to.
MemRead	1 bit	1 if you want to read the **block** at MemAddress and 0 otherwise.
MemWrite	1 bit	1 if you want to write Line2Mem to MemAddress and 0 otherwise.

Line2Mem	32 bits	The **block** you want to write to MemAddress

Why is MemAddress 32 bits instead of 10?

Our Memory is going to be **Block Addressable** instead of Byte Addressable. This means that each block has an address instead of each byte. This enables you to read a block from/write a block to memory in a single operation, which should help simplify your design a little. It should be pretty easy to figure out how to chop to form MemAddress. Hint: it is bits that help you pick out the byte with

When are outputs checked?

- DidContain: At the end of every operation once your circuit becomes **Ready**.
- ByteRead: At the end of a read operation once your circuit becomes **Ready**.
- MemRead: Whenever you read from memory it increments the MissCounter

Timing Restrictions

Your circuit must be able to complete a Read/Write request **within 10 clock cycles**. If you take longer than 10 clock cycles the tester will automatically advance to the next case. This will cause your Cache to start to "desync" from the tester and so will get everything wrong if you are taking too long. You can certainly finish in less than 10 clock cycles, I only took 7 but I've seen students get as low as 2.

Testing

1. Open the grading circuit
2. Click on the Cache folder and select reload library
    a. Double-check that your updates are visible inside of the grader. If your changes don't show up, close your solution circuit and try reloading again. If the changes still fail to show up, close the grader and reopen it and then reload the library.
3. Open the subcircuit named CheckerCirc and
    a. Right-click the ROM inside of it
    b. Select Load Image
    c. Select one of the provided files that ends in _sol.txt.
        i. Eample: seq_read1_seq_mem_sol.txt
4. Open the subcircuit named InputGeneratorCirc and
    a. Right-click the ROM inside of it
    b. Select Load Image

   c. Select the corresponding input file that matches with the solution file. This will be the file that shares its name with the first part of the solution file.
      i. Example: seq_read1.txt
5. Go back to the main subcircuit
   a. Right-
   b. Select
   c. Select emory file that matches with the solution file. This will be the me with the second part of the solution file.
      i. .txt

Anytime your reset a **ELOAD the RAM!** The RAM is the only one that has to be reloaded. The ROM's will maintain the last value you put in them.

At the bottom of the the main circuit you will find probes that show your circuit's inputs, outputs, answers, how many reads you got correct, how many times you said you hit, and how many misses(memory reads) you had. This is a good place to look when you are debugging to get a high level overview of what your cache is doing.

If you open the solution files in a text editor you will find what the correct values for all of the stats should be for that test case.

Testing Order

I recommend running the test cases in the following order

1. seq_read1_seq_mem_sol.txt
2. seq_write1_seq_mem_sol.txt
3. set1_targeted_read_seq_mem_sol.txt
4. set0_targeted_write_seq_mem_sol.txt
5. random0_seq_mem_sol.txt
6. final_test_rand_mem_mem_sol.txt.

User Created Tests

If you want to make your own test, you can use cache_test_maker.py to create your own tests. It should be fairly easy to follow the documentation inside if you feel like using it.

Scoring

Your score is calculated as num_correct_read_values - abs(num_correct_read_value - number_of_reads_you_got_correct) + num_correct_hits - abs(num_correct_hits - your_number_of_hits) + num_correct_misses - abs(num_correct_misses -

your_number_of_misses). Essentially you lose points for every value you are off from the right answers.

Debugging

To help you with debu ent, Noah Rose Ledesma, created a program to help show you what your doing so you don't have to do it by hand. You can find his program at https: seLedesma/CacheHitDetector . Give it a star if you find it helpful.

Once you know what is supposed to be happening you have to go back to the ctrl+t and watching everything your circuit is doing to make sure it is doing what you expect it to. This is one of the most time consuming parts of your program but you can speed it up by attaching probest to literally everything so you know what is happening.

What to Submit

A file named Cache.circ with your solution in the implementation subcircuit.

Hints

- Before you start doing anything in Logisim make sure you really understand how the Cache should work on both a read and a write. Write out Psuedo Code detailing what should happen. This Psuedo Code should be so detailed that you could write a program that emulates a cache. You need that thorough of a plan to correctly implement your cache.
- Make subcircuits for Sets and Ways. Then for each Set or Way you need you can lay down an instance of that subcircuit.
- Speaking of Subcircuits, use lots of them to help keep your circuits neat.
- The bit finder can be a very useful tool in helping you to figure out which way within a set you should be working with.
- Do the data path first and then the control path. You won't know what control signals you need until the data path is built so build that data path first. I ended having about 7 control signals. You may have more or less depending on your design.
- When it comes to data values, it is generally safe to pass the same data value to all of your Sets/Ways. Students are tempted to use Demuxes to "send" the data to the correct Set/Way. This is an incorrect way to think as the Sets/Ways that aren't selected still get a value, it is just 0.
- For control signals, you may need to send them to only a single Set/Way by using a Demux or you may need to send the signal to all Sets/Ways. To figure this out simply,

ask yourself, “Should all Sets/Ways do this or just one of them?” If it is just one, Demux the signal, if it is both don’t Demux it.

- Your circuit does not have to start out being ready. I needed an initialize state so that I could set the ages of the lines and as such I wasn't ready to begin until after I had passed throug
- You can greatl by just thinking that you will get your value from a set or a line. Obv e the right one, but you can easily select among them by using mux t right one gets the correct input by using dmuxes. This will save enumerate all the possibilities of where the element you are lookir
- Here are som ade that I made solving the problem. Hopefully you can learn from them
    - Forgot to rename some labels
    - Needed to both add and remove states from my FSM
    - Mixed up the Id of a Way with its Age
    - Messed up on the aging of lines. I assumed that they would all start with unique values but of course they don't.
        - The fix was to add an initialization state where I would set the ages of the lines to 0, 1, and 2.
    - Used the wrong Tag bits to when addressing memory on writes.
        - Fix. When writing to memory you should use the Tag bits found in the Line that you will be writing. When reading from memory you should use the Tag bits from the CpuAddress.