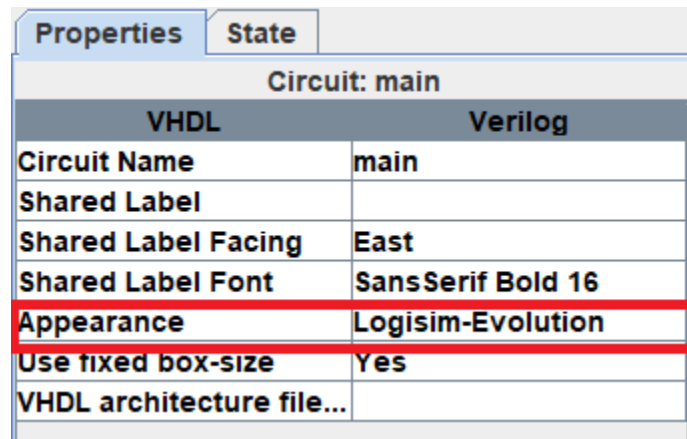


ECS 154A: Homework 3

Logisim

1. Use the [version from the class Google Drive of Logisim Evolution](#). Other versions may not work correctly.
2. Do not rename the files you receive. If you do so you will automatically fail the tester when you submit.
3. Put your solution for each problem into implementation subcircuit
4. Do not rename the implementation subcircuit anything else. If you do so you will automatically fail the tester when you submit.
5. Do not change the appearance of the 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



| Properties | | State |
|---------------------------|-------------------|---------|
| Circuit: main | | |
| VHDL | | Verilog |
| Circuit Name | main | |
| Shared Label | | |
| Shared Label Facing | East | |
| Shared Label Font | SansSerif Bold 16 | |
| Appearance | Logisim-Evolution | |
| Use fixed box-size | Yes | |
| VHDL architecture file... | | |

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 subcircuits 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

Restrictions

For all problems in this homework, you may only use

- All of the components under Wiring
- AND, OR, NOT, and XOR gates
- All of the components under Plexers
- All of the components under Arithmetic
- All of the components under Memory **EXCEPT** for **Counter**, **RAM**, **ROM**, and **Random Generator**

Unless a problem specifies otherwise.

Problem 1: UpDownCounter.circ (25 points)

Implement a 3 bit synchronous up/down counter that stops counting when it reaches the minimum/maximum count. For example, if the count is at 111 and you are told to count up you should stay at 111. Or if you were at 000 and told to count down you should stay at 000.

CountUpIn and **CountDownIn** in will never both be 1 at the same time.

This circuit should be constructed as a **Moore** Model Machine

Inputs

| Pin | Size (in bits) | Explanation |
|-------------|----------------|--|
| EnableIn | 1 | Connect to the enable port of your registers/flip-flops. When 1 your circuit works normally. When 0 your circuit does nothing. |
| CountUpIn | 1 | Increase the count by 1 unless you are at 111. |
| CountDownIn | 1 | Decrease the count by 1 unless you are at 000 |
| clkIn | 1 | Connect this to the clock ports of your registers/flip-flops. Do nothing else with this. |

Outputs

| Pin | Size (in bits) | Explanation |
|-------|----------------|----------------------------------|
| Count | 3 | The current value of the counter |

Problem 2: FourBitParity.circ (25 points)

Build a **Moore** Model circuit that calculates the even parity over a **group of 4 bits**. You will receive 1 bit of input at a time. The circuit should output a 1 if after receiving the last bit in the group there were an even number of 1's in the group and a 0 at all other times. 0000 is considered to be an even number of 1's.

This is not a sliding window problem but instead a chunk based problem. This means instead of looking at the last 4 bits received you look at the first group of 4 bits, then the second group of 4 bits, then the third group of 4 bits, etc.

Example Input and Output

Input: 0111 1000 1010 1100

Output: 0000 0000 0000 1000 1

If the output looks like it "is delayed one clock cycle" that is because you are building a Moore Model Machine and the output of a Moore Machine cannot change until the next rising clock edge because its output depends on state only.

Inputs

| Pin | Size (in bits) | Explanation |
|------------|----------------|--|
| EnableIn | 1 | Connect to the enable port of your registers/flip-flops. When 1 your circuit works normally. When 0 your circuit does nothing. |
| InputBitIn | 1 | The current bit being input into your circuit. |
| ClkIn | 1 | Connect this to the clock ports of your registers/flip-flops. Do nothing else with this. |

Outputs

| Pin | Size (in bits) | Explanation |
|-----------|----------------|--|
| IsEvenOut | 1 | 1 if there were an even number of 1's in the last group of 4 bits and 0 otherwise. |

Problem 3: Vending.circ (50 points)

Implement a **Mealy** model circuit to control a coin-operated vending machine. This machine only accepts quarters, dimes, and nickels. Coins are inserted until a total of 30 cents or more is deposited. **Only 1 coin is deposited at a time**. Once a total of 30 or more cents has been deposited your circuit should set Give_Mechandise to 1 to dispense the product in the vending machine. You should also set the change outputs for the circuit to give back the appropriate amount of change. Assume that the machine can give one dime and/or nickel back. If the customer does something silly like entering a quarter followed by a quarter (total of 50 cents), correct change does not have to be provided (20 cents) but the maximum amount of change should be given (15 cents). Note that only 1 input can be high at a time.

Inputs

| Pin | Size (in bits) | Explanation |
|-----------------|----------------|--|
| EnableIn | 1 | Connect to the enable port of your registers/flip-flops. When 1 your circuit works normally. When 0 your circuit does nothing. |
| QuarterReceived | 1 | 1 if the user entered a quarter into the machine and 0 otherwise |
| DimeReceived | 1 | 1 if the user entered a dime into the machine and 0 otherwise |
| NickelReceived | 1 | 1 if the user entered a nickel into the machine and 0 otherwise |
| ClkIn | 1 | Connect this to the clock ports of your registers/flip-flops. Do nothing else with this |

Outputs

| Pin | Size (in bits) | Explanation |
|----------------------|----------------|--|
| Give_Merchandise_Out | 1 | 1 if the user has entered 30 cents and 0 otherwise |
| Give_Dime_Out | 1 | 1 if a dime should be given back as change and 0 otherwise |
| Give_Nickel_out | 1 | 1 if a nickel should be given back as change and 0 otherwise |

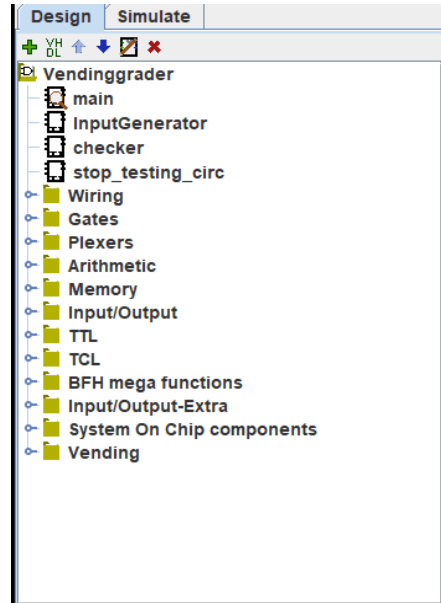
Credit

Credit for this problem goes to Sean Davis.

Testing

Testing for this problem is much different than for previous assignments because we are building sequential circuits. After you finish building your circuit and are ready to test it

1. Open the associated grader circuit
2. Scroll down on the left and side until you find your circuit. Right-click on it and select Reload Library



- a.
3. Optionally turn StopOnMismatchIn on to have the testing stop on the first incorrect output
 - a. If testing stops because of this you'll need to reset the simulation after you make your fixes otherwise you'll be stuck here forever.
 4. Start the simulation by going to Simulate and Ticks Enabled (ctrl + k)
 - a. To increase the simulation speed go to Tick Frequency and select your desired frequency
 - b. You can manually do one tick at a time by selecting Tick Half Cycle (ctrl + t). This is useful when you are debugging
 - c. You can reset the simulation by selecting Reset Simulator (ctrl + r)
 5. The counter at the bottom will tell you how many you got right and listed next to it is the total number of points for the problem

Debugging

Here is my recommendation for how to debug

1. Turn on StopOnMismatchIn
2. Start the simulation
3. Note down the test number that your circuit fails on
4. Reset the simulation

5. Start ticking until you get to the test number two or three before the one you failed
6. Use the poke tool to go inside of your circuit
7. Check if everything is as you expect it to be. Check absolutely everything including state, next state, and output.
8. Tick to the next state.
9. Repeat 7 and 8 until you find your error
10. If go too far go back to step 4

Debugging sequential circuits is much harder than combinational circuits as the output depends on both the current inputs and the **current state**. This means that you may be failing test case X not because it is doing something wrong on that test case but because you transitioned to the wrong state N test cases ago. So if you are failing test case X you may need to check all of the test cases before X to validate you are switching to the correct state prior to test case X

Making Fixes to Your Solution

After you make changes to your solution you will need to reload your circuit in the grader circuit. If you don't it won't see the updates. To reload your circuit, select your circuit in the grader, right-click it and select Reload Library.

Submitting

Submit to

Logisim Homework 3 on GradeScope.

What to Submit

Submit a zip file that contains the following .circ files

1. UpDownCounter.circ
2. FourBitParity.circ
3. Vending.circ

Inside of each .circ file leave a comment with you and your partner's names.

Make sure that you submit a zip that contains the files and **NOT** the **folder containing the files**. Check out the animation below for what to submit.

