

Logisim Design Guidelines

Frequently Asked Questions

Q: How does component xxx work in Logisim?

We have had many questions asking about how to use a particular component in Logisim. In general, the help file in Logisim has explained things very well. You can access the help file from the 'Help' menu in Logisim. You could either search by keyword or browse the entire reference from the left pane.

Alternatively, you can take a look at our CS 3410 [Logisim Component Guide](https://canvas.cornell.edu/courses/42905/pages/logisim-component-guide) (<https://canvas.cornell.edu/courses/42905/pages/logisim-component-guide>).

Q: How to create a sub-circuit in Logisim?

Please refer to [Sub-circuit creation](http://www.cburch.com/logisim/docs/2.6.0/en/guide/subcirc/creating.html) (<http://www.cburch.com/logisim/docs/2.6.0/en/guide/subcirc/creating.html>).

Q: How to build a one-bit adder?

Logisim has a very nice feature for editing a truth table and generating the corresponding logic circuit. You could refer to the following pages [Edit truth table](http://www.cburch.com/logisim/docs/2.6.0/en/guide/analyze/table.html)

(<http://www.cburch.com/logisim/docs/2.6.0/en/guide/analyze/table.html>) and [Generate Circuit](http://www.cburch.com/logisim/docs/2.6.0/en/guide/analyze/gen.html)

(<http://www.cburch.com/logisim/docs/2.6.0/en/guide/analyze/gen.html>) for an explanation.

The following steps have worked well:

1. Place the desired input and output nodes on the canvas.
2. Save the changes to the canvas.
3. Right Click on the symbol that represents the current canvas in the explorer pane, and select 'Analyze Circuit' to bring up the 'Combinational Analysis' window
4. Edit the truth table and Generate the circuit

Q: How does this multiplexer thing work?

It takes multiple inputs and selects one of them based on the control signal. For instance, suppose I have 4 inputs into the multiplexer in the following order: 1, 2, 3, 4. Since we have 4 inputs, our control needs 2 bits (base 2 log of the number of inputs) to determine the output of the multiplexer. In this particular case, the mapping would be: 00 => 1, 01 => 2, 10 => 3, 11 => 4.

Q: Can logisim make these circuits for me?

Yes! Take advantage of Logisim's combinational analysis window (found under *Project > Analyze Circuit*), which can automatically generate near-optimal circuits given a truth table. This only works for circuits with a few inputs, but is very well suited to control logic.

Logisim Design Best Practices:

Good Design #1: Labeled Circuit Diagrams

- A circuit with no labels is like code with no comments.

For both your own understanding and our grading, it is easier to follow your logic if you specify what the subcircuits are and what they're used for.

Good Design #2: Reasonably Sized Muxes

- Having an overly large mux is both inefficient and potentially redundant.

For example, there is no need for a 16-input mux in your ALU, especially for selecting a simple output like V. The cases in which V matters are few and the majority of the time it should be 0. Instead of looking at the whole ALU opcode to select V, think of other simpler options. Rule of thumb, keep your muxes **at or below 4 inputs**.

Good Design #3: Bit Extenders Rather Than Multibit Constants

- Constants should not be used to extend a number if a bit extender can be used instead.

Bit extender is neater and easier to understand.

Bad idea: Using a splitter and constant to perform extension.

wrong

right

It is better design to use as few constants as possible (to be clear, you should never use the "Constant" logisim component nor any behavior that is trivially equivalent to it).

Good Design #4: Not Muxing a Signal and its Inverse

- Muxing between a signal and its inverse unnecessarily increases gate count.

This can be replaced with an XOR gate.

Let's look at the truth table of the mux:

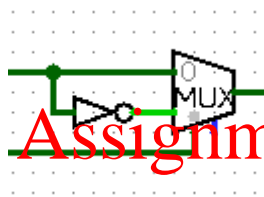
Input (top wire)	NOT Input (bottom wire)	Signal (wire at bottom of mux)	Output
0	1	0	0 (choose top wire)
0	1	1	1 (choose bottom wire)
1	0	0	1 (choose top wire)
1	0	1	0 (choose bottom wire)

The truth table for the XOR gate in which we XOR the input and the signal is the same as above!

Input (top wire)	Signal (bottom wire)	Output
0	0	0
0	1	1
1	0	1
1	1	0

This holds for inputs of any number of bits, as NOT and XOR are bitwise operations.

Bad idea: Use a multiplexor with inverses as inputs.



wrong



right

The circuits on the left and right are equivalent.

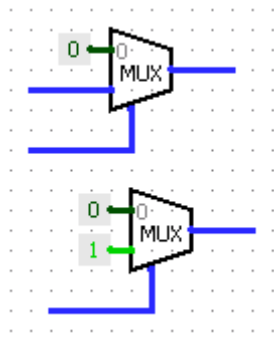
Good Design #5: Avoiding Use of Constants

- In general, you should not need **any** constants in your circuits.
- Gates that always evaluate to constants regardless of the circuit input are even worse (e.g. XOR a signal with itself to produce 0, or OR a signal with its inverse to produce 1). These have all the same problems as a constant, except they increase the gate count and critical path too (making the circuit both more expensive and slower).
- In addition, do not use any low-level/below-circuit-level primitives such as ground or transistors to take the place of constants.

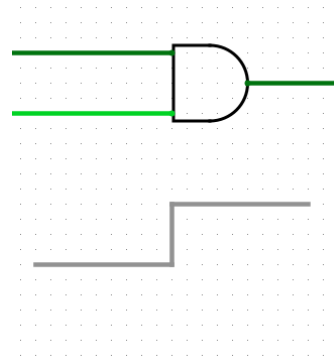
The only (rare) exceptions are for (a) providing inputs to a sub-circuit that has more inputs than you need, or (b) making your circuit look slightly simpler in certain cases. Any time you feel like putting down a constant, ask "can I just optimize this away by using a different gate?"

Bad idea: Use a multiplexor with constants as inputs.

--	--



wrong



right

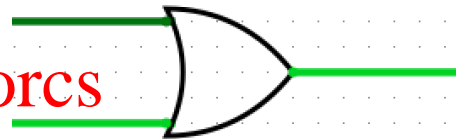
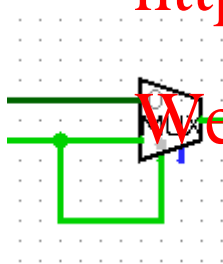
These almost always reduce to just an AND gate, without the constant. Or in the worst case, a no-op.

Good Design #6: Avoiding Use of Input for Both Data and Control

- There is no need to use the same signal for data and control of the same mux.

This case is very similar to the case above. It can easily be replaced with logic gates.

Bad idea: Use a multiplexor with same signal as input and control.



right

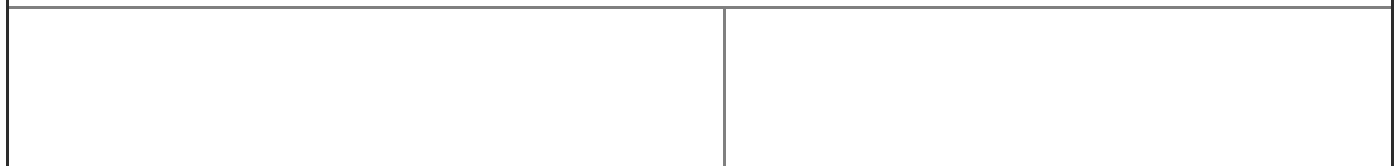
These circuits are equivalent. The control is just a case of OR.

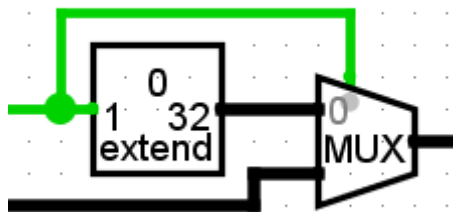
Good Design #7: Avoiding Use of Input for Both Multi-Bit Data and Control

- There is no need to use the same signal for data and control of the same mux even if it is a multi-bit mux.

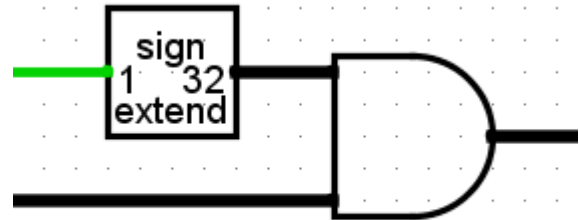
Like the case above, it can easily be replaced with a multi-bit logic gate with a sign-extended control bit.

Bad(ish) idea: Use a multiplexor with same signal as multi-bit input and control.





wrong



right

These circuits are equivalent. The control bit is extended into a string of all 0s or all 1s.

Good Design #8: Useful Sub-Circuits

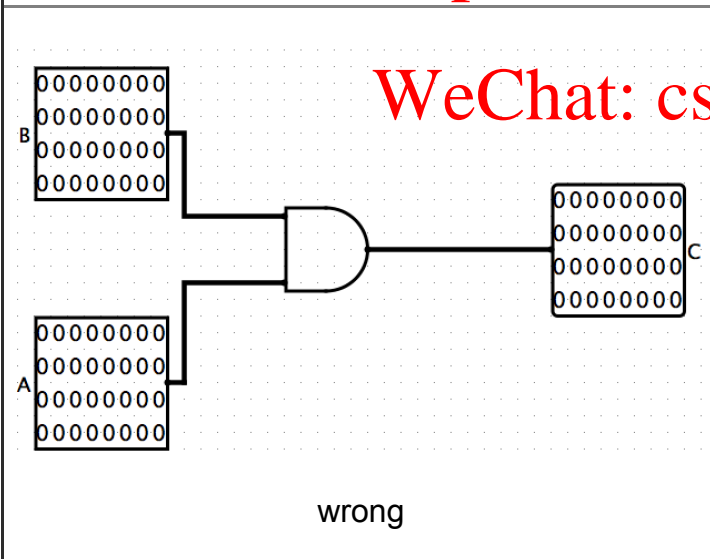
- Think of sub-circuits as functions in your code; abstraction can improve readability and prove quite useful, but too much can be the opposite. You wouldn't write a function in your program that simply performs addition when you can use the '+' symbol!
- If a sub-circuit has only one component, it probably doesn't deserve to be a sub-circuit.

Assignment Project Exam Help

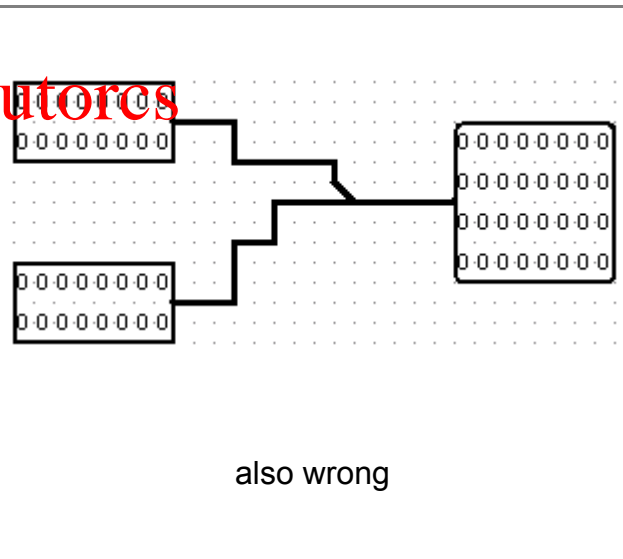
Instead, just use the component itself. An exception to this rule is the "bit reverser" circuit that many of you built (with no components, just splitters and wires), since it takes up so much screen space.

<https://tutores.com>

Annoyance: unnecessary and distracting sub-circuits



wrong



also wrong

These sub-circuits are just re-packaging and re-naming already existing Logisim primitives. Just use the primitive instead.

Good Design #9: Testing Completely

- Make sure the circuit you turned in does not have any red wires and can pass all your own test vectors. If your circuit is supposed to work on signed two's-complement numbers, please test it with both positive and negative numbers.

At minimum, do positive+positive, positive+negative, negative+negative, and negative+positive. And try a couple of each combination, too.

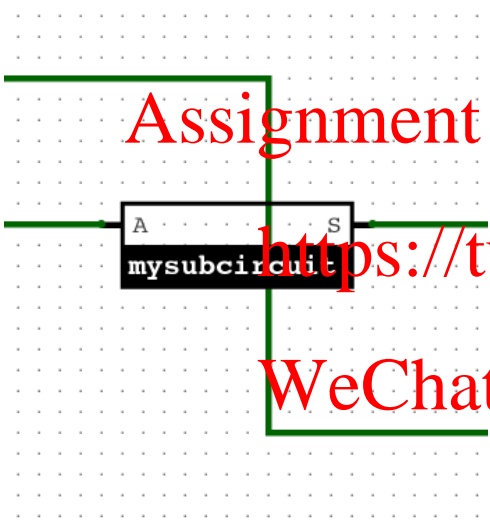
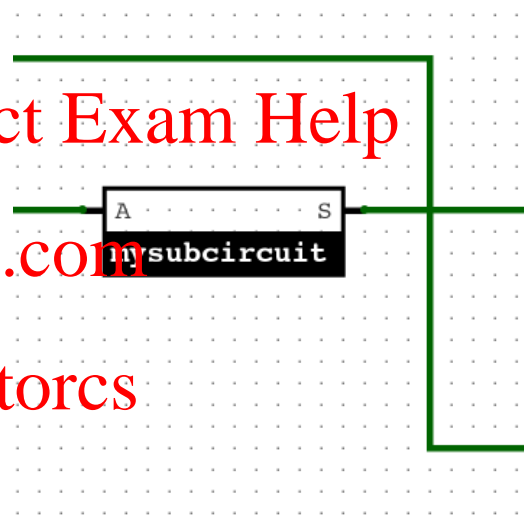
Good Design #10: Not Using External File Dependencies

- Ensure that the circuit you turn in does not have external file dependencies, such as using a circuit or subcircuit from another file. To do this, you can move your submission file to a new and empty folder and run it from there before you submit.

Good Design #11: Not Crossing Wires over Subcircuits

- Your wires should travel around your subcircuits, **not** through them. Wires can overlap other wires, but not subcircuits.
- This extends to gates and pins as well. To reiterate, wires should only ever overlap **other wires**.

Bad idea: Crossing the beams wires with subcircuits

 <p>wrong</p>	 <p>right</p>
---	--

The right circuit looks much cleaner and more readable.

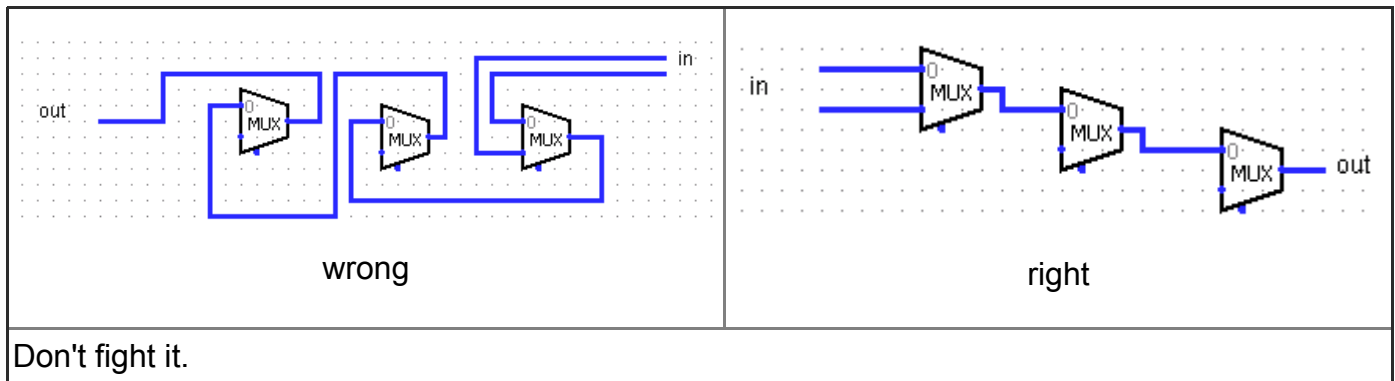
Good Design #12: Limited Use of Tunnels

- Tunnels should always be used sparingly and thoughtfully, **not** for every wire in your circuit.

There is no hard cutoff/metric for when too many tunnels have been used, but keep in mind that this design guideline exists to maintain circuit readability. Unless there is a cluster of wires that could easily be cleaned up with the use of a tunnel, it is usually far simpler and quicker to trace a wire manually rather than having to figure out where each end of a tunnel is located in the circuit.

Logisim Quirk: Data Flow

This is a minor point. But you have probably noticed that Logisim likes data to flow left-to-right on the screen. For example, the default orientation of primitives are all left-to-right, and multiplexors can't be rotated at all. Many of you displayed an amazing stubbornness by having your circuits go in other directions.



Assignment Project Exam Help

<https://tutorcs.com>

WeChat: cstutorcs