Guidelines for Using LogicCircuit with the Nand2Tetris HDL Tool Suite

John K. Bennett
University of Colorado
jkb@colorado.edu
March 2022

Introduction and Installation

These notes describe how to use a modified version of Eugene Lepekhin's *LogicCircuit* program to generate the ".hdl" files expected by the Hardware Simulator that is part of the *Nand2Tetris* tool suite created and distributed by Noam Nisan, Shimon Schocken, Mark Armbrust, and their colleagues.

The HDL suite is also capable of generating structural VHDL and Verilog suitable for use with FPGA design software such as the Xilinx Vivado Design Suite.

You will need the modified version of LogicCircuit that supports HDL and VHDL generation. For better compatibility with the Nand2Tetris software, you will also need additional builtInChips. Both of these resources, plus others, can be found at:

https://github.com/jkb-git/LogicCircuit-to-HDL-or-VHDL/upload/main/windowsInstall

LogicCircuit is currently only available for Windows.

I have successfully used the modified version of LogicCircuit to produce hdl design files for all hardware projects enumerated in the Chapters 1, 2, 3, and 5 of "The Elements of Computing Systems."

To install the modified version of LogicCircuit, first uninstall any existing versions, then double click on the installer. In order to make LogicCircuit interact with the N2T Hardware Simulator more seamlessly, you will also want to add additional builtInChips (these are explained below) by copying the contents of the provided builtInChips directory to

`<N2T_install_directory>\tools\builtInChips`.

You do not need these additional to use LogicCircuit's "SaveAsHDL" feature, but you will have to create HDL wrappers for these chips if you want to test the resulting HDL files using the N2T tool suite, which names basic devices and their pins differently than LogicCircuit.

Using the modified LogicCircuit to complete the hardware assignments in the Nand2Tetris book, I have found the design methodology that uses LogicCircuit to complete the design, and the N2T Hardware Simulator and test scripts to validate that design, to be a powerful and intuitive combination. I hope and anticipate that this combination will help students learn the course underlying principles more readily than if they just used HDL to develop their circuit designs.

With regard to the implementation, translators are ugly. Whenever we translate from one form of digital expression to another, many special cases must be addressed. This code is no exception to that rule. The good news is that, in general, translators do not have to be efficient. The "SaveAsHDL" feature has been tested reasonably thoroughly in the course of implementing all of the N2T book's hardware projects. Others will no doubt encounter problems not yet identified. If you do find a bug, please send the circuitproject file that

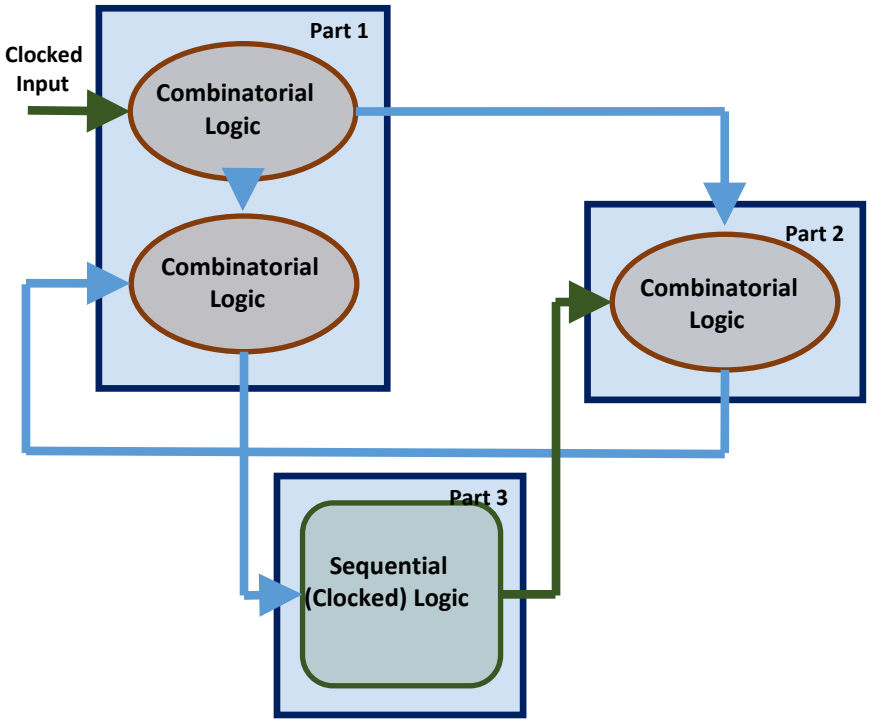
demonstrates the problem, together with a short explanation of the problem, to jkb@colorado.edu. I will endeavor to address repeatable problems as they are identified.

The structural VHDL and Verilog generated by this code is targeted to the Xilinx Vivado Synthesis Suite. I have used this code to successfully implement a VHDL version of the N2T CPU using this tool suite (implementing the full N2T Computer requires additional platform-specific code not included here). There are many issues related to the correct use of these tools with actual hardware that cannot be easily addressed within LogicCircuit.

Specific Recommendations

1. Do not use underscore ("_") in names for HDL parts or pins. (N2T HDL compatibility issue).
2. Do not use dash ("-") in names for VHDL or Verilog parts or pins. (VHDL/Verilog compatibility issue).
3. All circuits in a project must be given names that are acceptable Windows file name prefixes, e.g., "**2:1**" is not OK; "**Mux2-1**" is fine. The translator will attempt to correct such use (and warn you that it has done so).
4. Make sure part pin names match builtIn pin names (from the book) if you want to use builtIn parts with the N2T Hardware Simulator.
5. LogicCircuit features with no corresponding HDL, VHDL, or Verilog equivalent (i.e., "Button", "Sensor", "LED", "7-seg", "LED Matrix", "Buzzer", "Probe", and "Tristate") will be ignored by the translator (with a warning). You generally may include these circuits in your LogicCircuit design, but they will have no impact on the translated HDL, VHDL, or Verilog. It may be possible to construct a circuit sufficiently convoluted with unsupported parts (e.g., deeply imbedded tristate gates) so as to cause the translator to complain.
6. Make sure your LogicCircuit project can successfully power-on before exporting to HDL, VHDL, or Verilog. This helps ensure that there are no hidden wiring errors. Of course, powering on will not expose logic errors; that is the purpose of simulator testing.
7. There is a maximum limit of 100 parts per visible circuit project. If you are approaching this limit, your design is probably too dense to be readable.
8. In general, HDL does not care (or want to know) about unused inputs and outputs. VHDL and Verilog do, so we mark unused outputs as "open" and tie unused inputs low when saving to VHDL. This may not be what you want to happen. In general, you should specify the value of every input; this is just good practice
9. Nand2Tetris HDL does not allow us to sub-bus internal pins. This means that we cannot create an internal wire name at U10, say "U10out", with width 16, and then at the U12 "in" pin, with width 8, say "in = U10out[0..7]". This is an unfortunate restriction, but we can live with it, and I do not want to figure out how to fix it in the N2T Java source code, at least not now. The good news is that HDL is fine with us saying in the U10 PART spec: "out = name, out[0..7] = U10out0, out[8..15] = U10out1," etc., and then at U12 saying "in = U10out0". This is what the HDL export code does. It is OK to sub-bus project input and output pins, the code tries to use this capability when possible.

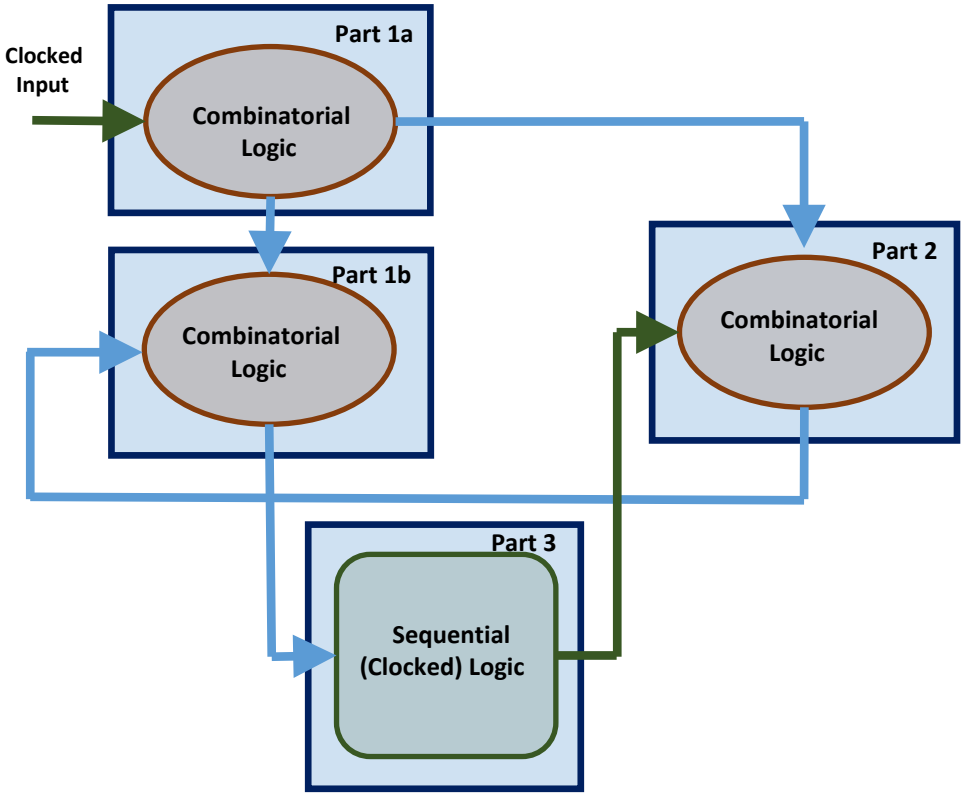
10. Cycles in combinatorial logic are bad (unless you are trying to build an oscillator), and neither LogicCircuit nor HDL allows them to exist (such cycles are discussed in Chapter 3 of the N2T book). Unfortunately, the cycle detection code in the N2T Hardware Simulator is not as smart as one might wish, as depicted in the Figure below.



There are three parts in this example. Part 1 contains two combinatorial sub-parts, the top one of which has as input some clocked signals, and the bottom of which takes as input an output of Part 3. Part 2 is also combinatorial. The output of the top portion of Part 1 is fed to Part 2 and the bottom of Part 1. The bottom part of Part 1 feeds to sequential circuits in Part 3, which in turn outputs to Part 2. The output of Part 2 feeds back to the bottom half of Part 1.

This arrangement is perfectly valid design, however, this circuit, represented as drawn in HDL, will cause the N2T Hardware Simulator to produce the error: "*The chip has a circle in its parts connections*". This not-too-helpful error is produced because the N2T Hardware Simulator cycle detection algorithm is unable to detect that there is fact no cycle in this circuit. In contrast, this circuit will work fine in LogicCircuit, which has a more sophisticated cycle detection algorithm.

To implement this circuit in a way that the N2T Hardware Simulator will accept, we have to separate Part 1 into two parts, Part 1a and Part 1b, as shown below.



Lest one think this an academic exercise; this issue is likely to arise in the design of the N2T CPU, where the Instruction Decode logic might represent Part 1a; the Jump Decode logic could represent Part 1b; Part 3 the PC; and Part 2 the ALU and its associated logic.

11. LogicCircuit introduces "splitters" to support, among other things, the creation of multi-wire busses to make schematics easier to both draw and read. Splitters have a single pin on the right or left side (whose bit-width can vary) and multiple pins on the other side (whose bit-width can also vary). The sum of all bit widths on each side has to match. While splitters may be rotated to horizontal or vertical orientations in LogicCircuit, **this should not be done when designing for export to HDL**. This is because portions of the export logic currently rely on a vertical orientation to detect certain characteristics of use. Splitters are considered "combiners" if they are used to combine several wires in to one.

    Splitters can be used in one of (at least) six ways:

    a. As a way to split out pins from a wide bus, e.g., one 16-bit wide bus in, 16 one-bit wires out.
    b. Like (a), but with > two multi-bit wires out, e.g., one 16-bit wide bus in, 4 four-bit wires out.
    c. The opposite of (a), e.g., 16 one-bit wires in, one 16-bit wide bus out (all signals with the same name)
    d. As a way to convert a single source signal to multi-bit wide signal (e.g., like (c), except all of the 16 inputs are wired together).
    e. As a way of combining several different signals into one bus (e.g., like (c) but some of the 16 inputs have different names).

f. As a way to change bus width, by dropping or adding signals between to splitters of different widths connected back to back.

As you might imagine, detecting and handling all of these possible uses introduces some complexity into the translation problem. However, since splitter/combiners are such an important aspect of LogicCircuit, we go to some effort to take advantage of their use. Of course, HDL has no idea what a splitter is, so our approach is to consider splitters a "pseudo part." In part (no pun intended), this is because of the limitations of HDL. There is no mechanism in HDL for a "part-less" circuit (e.g., (f) above). For VHDL and Verilog, the situation is better, since we can simply use signals and concurrent assignment statements to achieve the desired result.

Therefore, for HDL, we use new built-in "buffer" parts that directly connect inputs to outputs, and the HDL output will use these "parts" to implement certain kinds of splitters. These buffers come in two varieties, e.g., "Buff4" and "Buff4x1." A "Buff4" has four single wires on one side and four single wires on the other side. A "Buff4x1" has one four-bit wire on one side and four one-bit wires on the other. Note that we do not know which side of the splitter is the "source" until we reach (while tracing a circuit network) one of the sides from a real source (either a project input, constant, component part output pin) when tracing connections.

The good news is that the introduction of buffers in to the resulting HDL is transparent to the user. All buffer sizes relevant to N2T HDL have been pre-compiled as builtIn chips.

That said, the LogicCircuit HDL export code goes to considerable effort to avoid introducing a buffer. Buffers are only introduced if the export code determines that buffer introduction is unavoidable.[1] In general, splitters used as "splitters" do not require a buffer to be introduced. For splitters used as "combiners," we must add a buffer part, unless the use case meets certain criteria. These criteria differ depending upon whether the splitter is a single ("n x 1") or a multi-bit ("n x m", where m > 1) combiner. Generally speaking, the rules for buffer creation are as follows:

For the multi-bit case, we have to make a buffer part, unless it is an n x m combiner where:

a) All "thin" pin wire name prefixes are the same.
b) All of the range differences match, i.e., given wireName[x - y] and pinRange[r - s], (x - r) and (y - s) are constant for each pin and across all "thin" pins.

If these conditions are met, we can name the wide pin "thinPinPrefix [firstWireLowRangelastWireHighRange]."

For the single bit case, we have to add a buffer part, unless it is an nx1 combiner where:

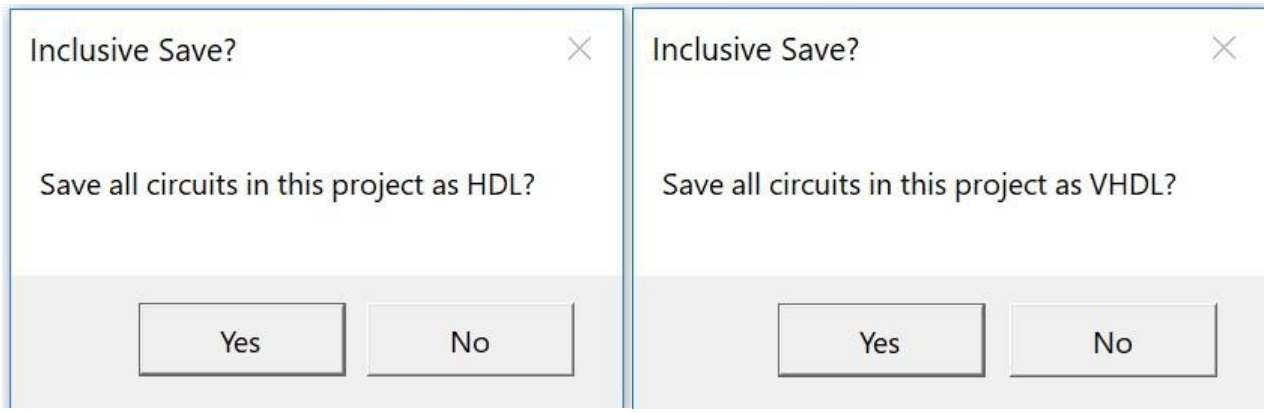
a) All thin pin wire names are the same.
b) All of the thin ranges are in sequence, i.e., given wireName[x] and pinRange[y], (x - y) is constant for all thin pins.

If these conditions are met, we can name the wide pin "thinPinPrefix [minWireRange-maxWireRange]"

[1] "Unavoidable" is not entirely accurate. It would be more correct to say the effort required to avoid buffer use in these cases exceeds the willingness of the author to code for that eventuality.

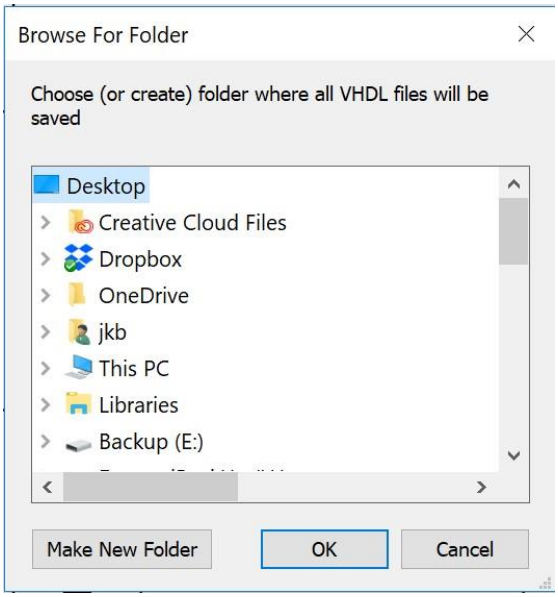
Finally, extra part output terms are sometimes generated when buffer parts are introduced. HDL does not care about these extra terms – you should not either.

SaveAsHDL, SaveAsVHDL, and SaveAsVerilog provide the ability to save either a single circuit, as shown below:



Selecting “No” saves a single file, and provides complete feedback regarding problems encountered.

Selecting “Yes” prompts for a directory name, as shown below:



If a new directory is desired, click in “Make New Folder,” enter the directory name, and hit return (do *not* hit “Make New Folder again. All circuits in the project that are capable of being translated will then be translated. Note that when using the inclusive save option, the translation will ignore any circuits that cannot be translated (e.g., ones that have a “Button”, “Sensor”, “LED”, “7-seg”, “LED Matrix”, “Buzzer”, “Probe”, or “Tristate” present).

Additional Translation Notes

When creating files representing a circuit, the translator will rename any LogicCircuit circuit name that is not a valid Windows filename prefix, taking into account the particular needs of HDL, VHDL, and Verilog. The translator will also prohibit or modify any entered filename that is not a valid filename.

Example: a multiplexor named “2:1” (perfectly ok in LogicCircuit) would be renamed to “N2-1” in HDL translation, and “N2_1” in VHDL or Verilog translation. These changes are transitive, i.e., any such modification is carried through the entire translation. For example the specification of a “Mux16” that used sixteen “2:1” multiplexors would have sixteen “N2-1” parts in its HDL translation. The user is warned if a circuit is modified in this way.

During VHDL translation only, any circuit name that happens to be a VHDL reserved word will be silently translated to include a “v_” prefix, e.g., “Mux” will become “v_Mux.” This change is also transitive. The user is not informed of this type of change. Ditto for translation to Verilog.

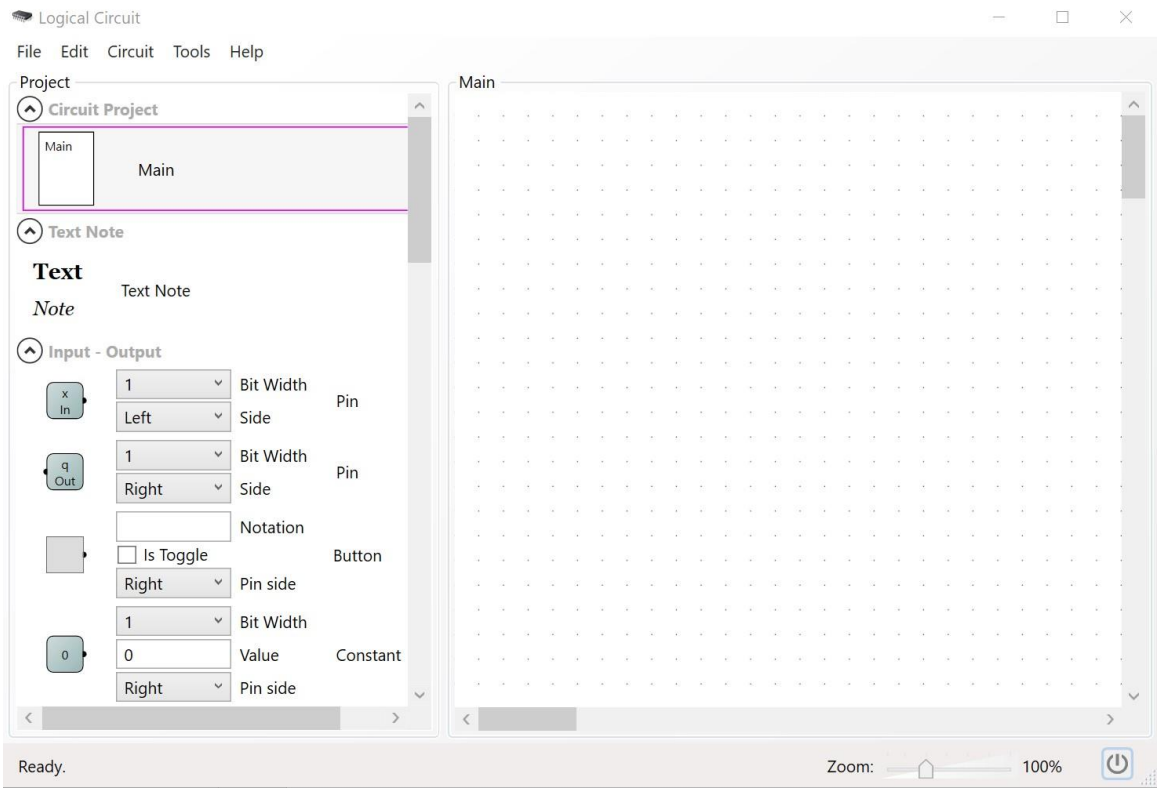
During VHDLand Verilog translation only, the translator attempts to make deliberate appropriate use of Xilinx VHDL and Verilog primitives (e.g. “OR2”), and avoids unintentionally invoking Xilinx VHDL or Verilog macros.

Example of Using Translator to Convert LogicCircuit Projects to HDL

The simple (and somewhat contrived) example demonstrates how to use the “SaveAsHDL” feature of LogicCircuit, to translate a LogicCircuit project directly to HDL, and then use the N2T Hardware Simulator to test our design.

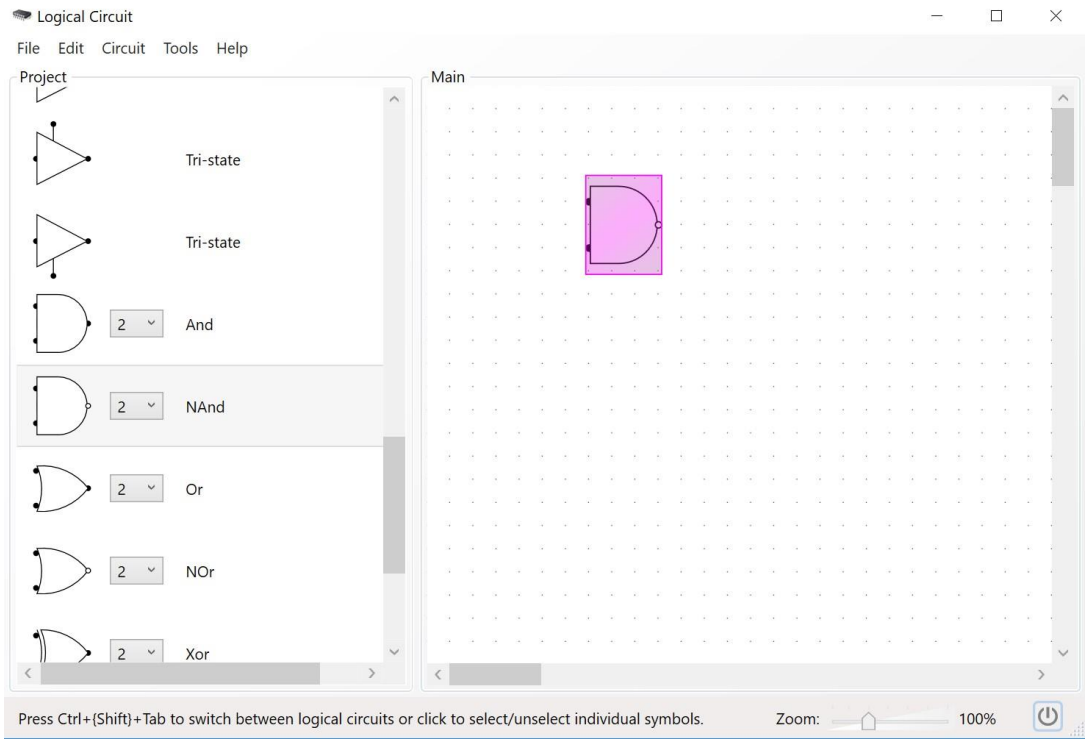
First, we need a Nand gate. LogicCircuit has a built-in Nand gate, but we want our gate to have pin names that correspond to what the Nand2Tetris (N2T) software tools expect. Let’s start with a new LogicCircuit project, which will look like Figure 1.

Figure 1



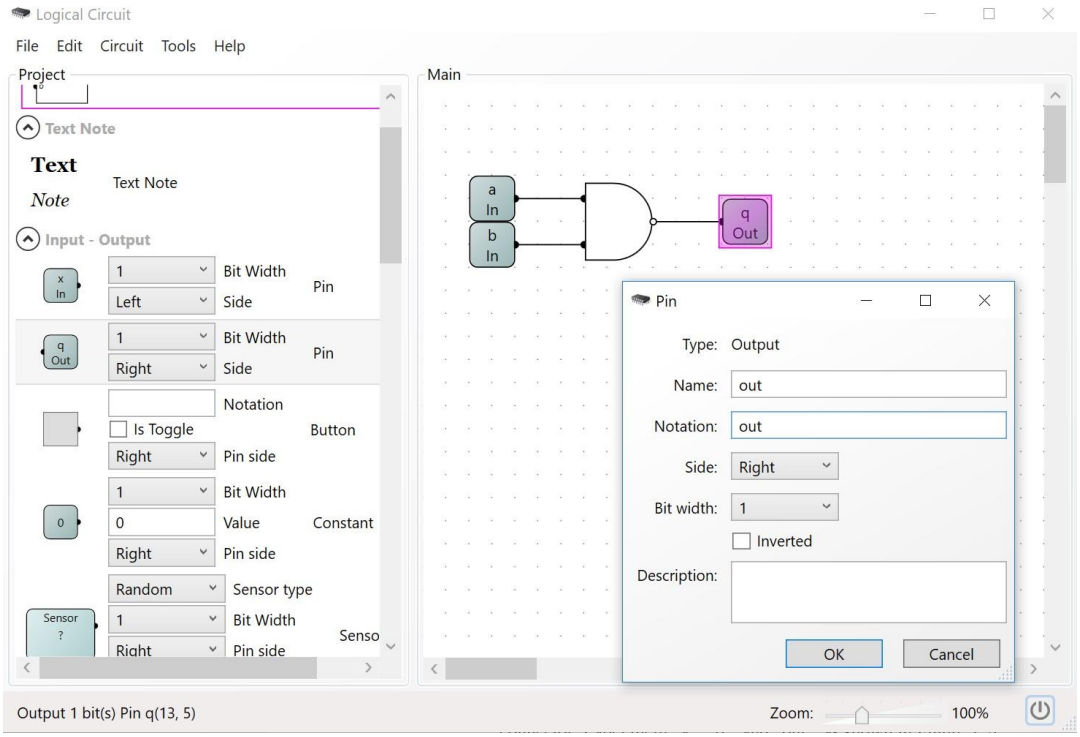
The menu of parts is on the left side, the design canvas is on the right. Add a two-input Nand gate from the primitive menu, as shown in Figure 2.

Figure 2



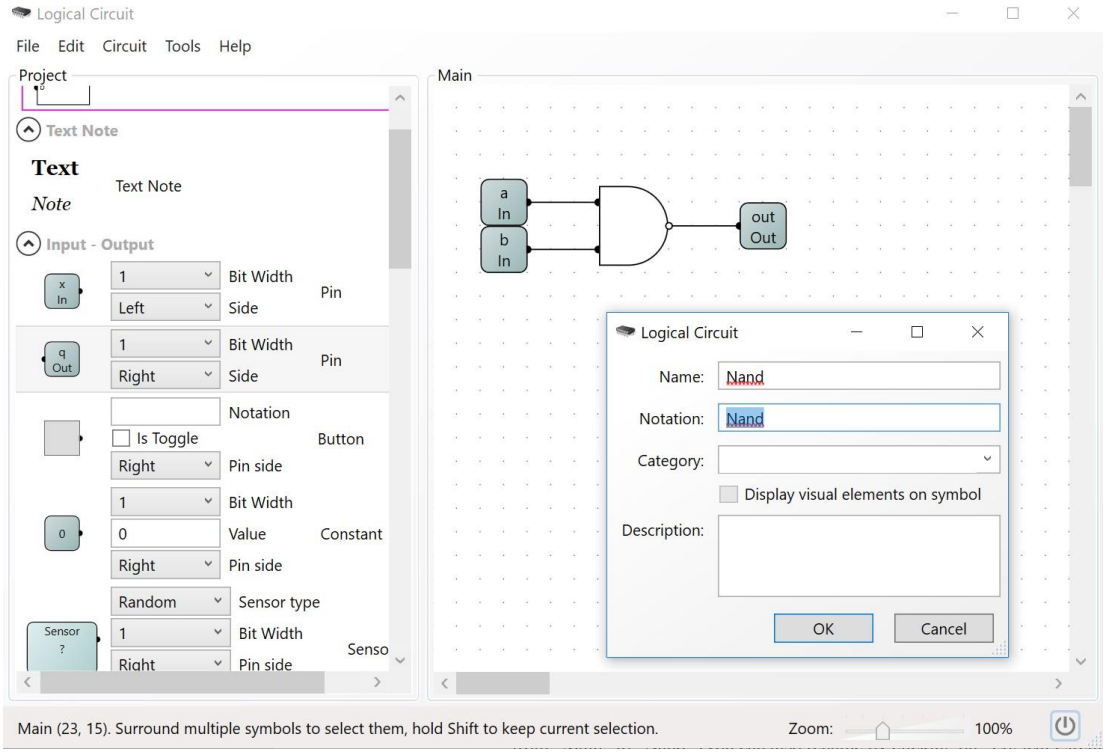
Now we need external connectors; add two input connectors, and one output connector. Double click on each one, and label them “a”, “b”, and “out,” as shown in Figure 3.

Figure 3



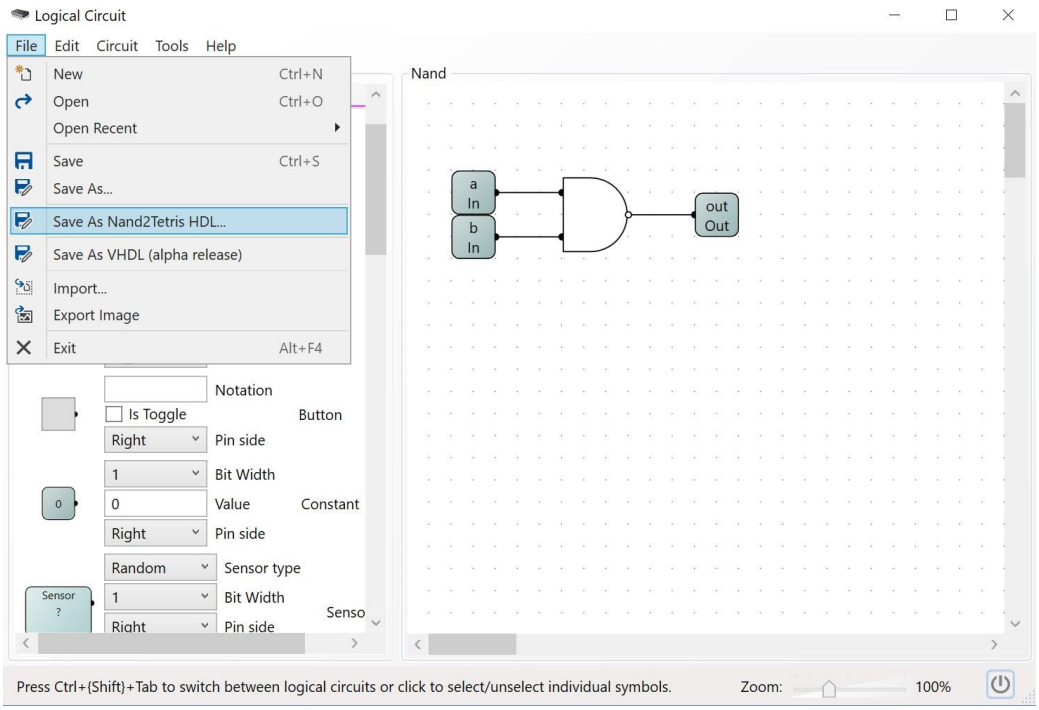
Next, double-click on an open area in the design canvas, and change the name the circuit from “Main” to “Nand” (you can also rename by clicking on “Logical Circuit” under the “Circuit” menu), as shown in Figure 4.

Figure 4



Finally, right click on “File,” then “SaveAsHDL,” as shown in Figure 5, and save the file to your desktop, or other convenient location. Answer “No” when prompted.

Figure 5



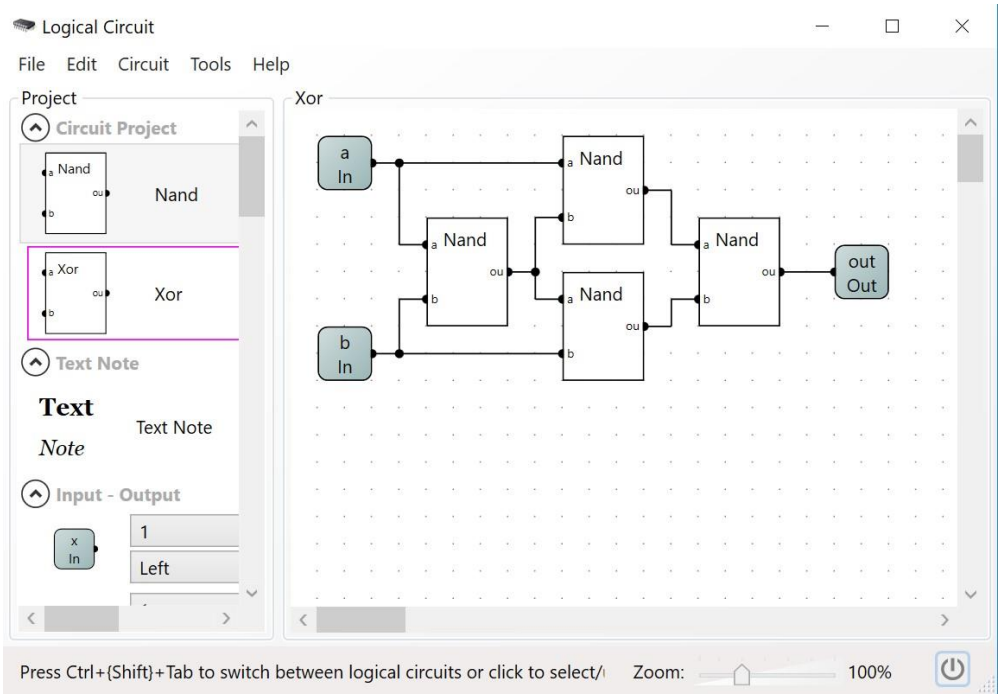
Examine the file you just created, which will read as follows:

```
// This file was generated from LogicCircuit CircuitProject: Nand
// Please report issues to jkb@colorado.edu
// 7/3/2021 11:40:19 AM

CHIP Nand {
IN a, b;
OUT out;

PARTS:
Nand2 (x1 = a, x2 = b, q = out, q = U0q);
}
```

This text describes a "chip" named "Nand" that is made from a "Nand2" gate, with two inputs (a and b), and one output (out). We will now use our new part to make an Xor gate.

In the same project, create a new logical circuit ("Circuit => "New Logical Circuit"), and name it "Xor." Now, place four of the Nand gates that you just created onto the design canvas, and wire them as shown. When two wires cross or intersect, and connection is not made automatically. To create a connection (signified by a little black dot) select the wire you are attaching to, and "alt-click" on the intersection. You should now see a dot. When you are done, your circuit should look like Figure 6. make sure it does, then save your Xor circuit as HDL (it will be convenient if you save to the N2T Project 01 directory, as this directory already has all of the files needed to test the circuit) , as before (be sure to also just save your work).

Figure 6



Now save as HDL as you did before. The "Xor.hdl" file just produced should contain the following text:

```
// This file was generated from LogicCircuit CircuitProject: Xor
// Please report issues to jkb@colorado.edu
// 7/3/2021 11:50:30 AM

CHIP Xor {
IN a, b;
OUT out;

PARTS:
Nand (a = a, b = U2out, out = U0out);
Nand (a = U2out, b = b, out = U1out);
Nand (a = a, b = b, out = U2out);
Nand (a = U0out, b = U1out, out = out, out = U3out);
}
```

Let's test our new circuit. In the "tools" directory of the N2T software, double click on HardwareSimulator.bat (Windows). The Hardware Simulator will open, as shown in Figure 7.

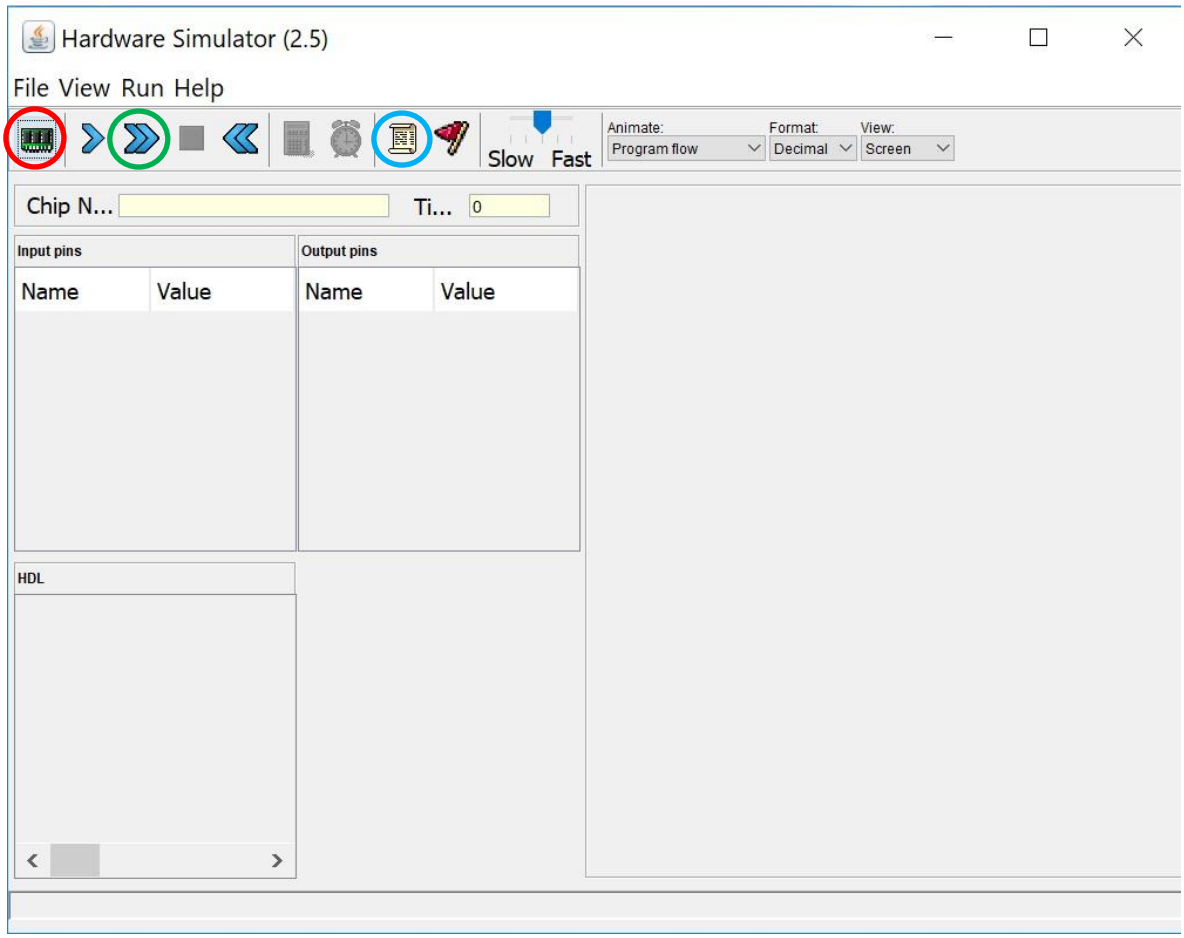


Figure 7

Click on the tiny printed circuit board with three black chips (red circle), and the Load Chip dialog menu will appear. Click on the "Xor.hdl" file you just created, then click on "Load Chip."

Now click on the small icon that looks like a scroll (blue circle), and the "Load Script" dialog will appear. Click on "Xor.tst," then click on "Load Script." Move the slide bar to "Fast," and click on the double right arrow (green circle).

The script will run, and the text at the bottom of the screen should read: "End of Script - Comparison ended successfully", as shown in Figure 8. Congratulations, you have just completed part of Project 1!

All of the N2T hardware assignments can be completed in this fashion, building more complex circuits out of simpler circuits that you have already built. Be sure to stay in the same LogicCircuit project, so that you will have those simpler circuits available.

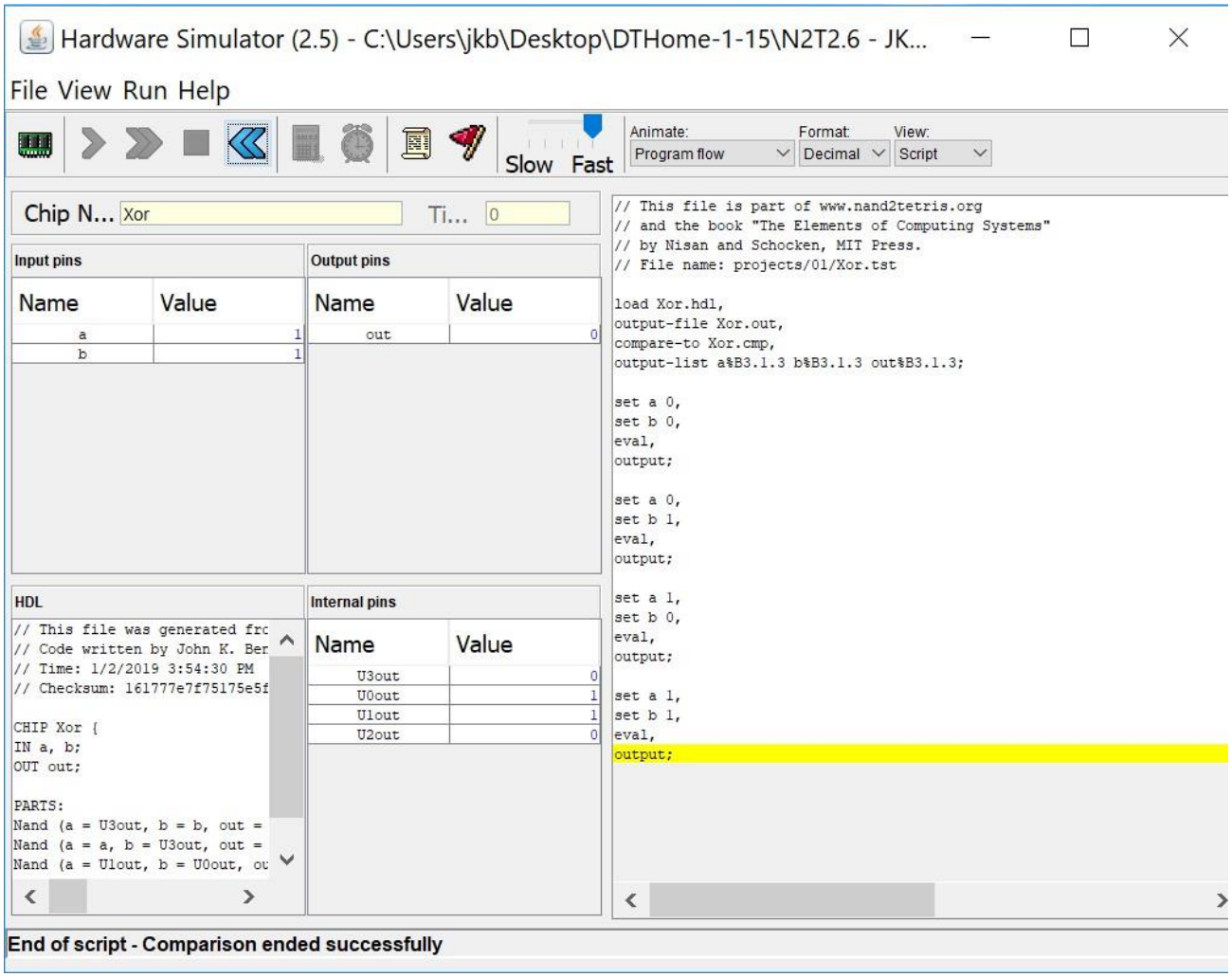
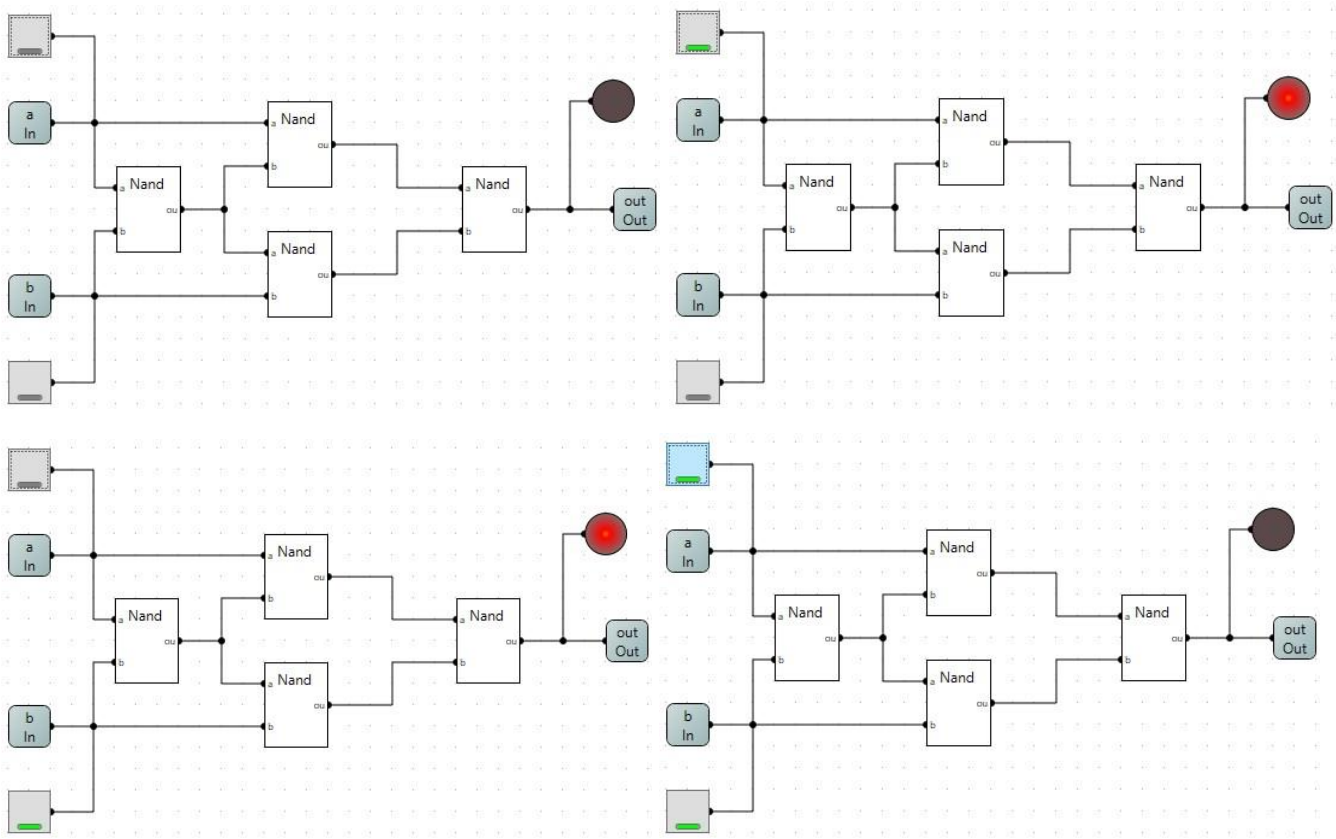


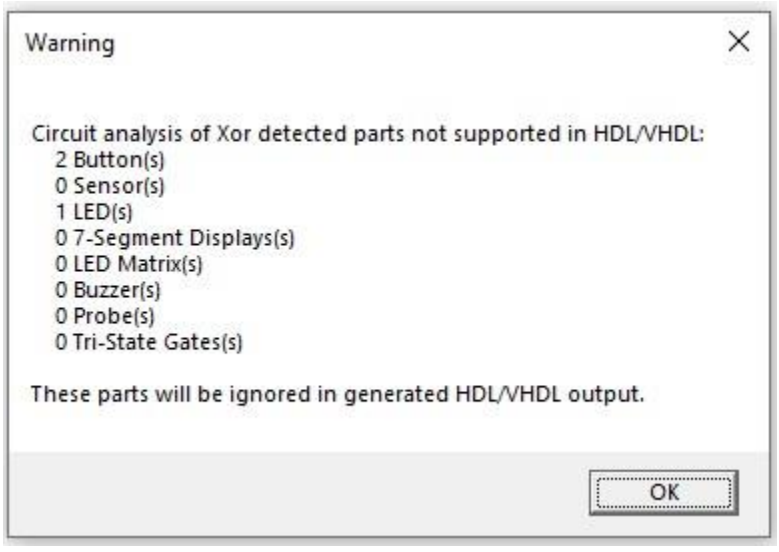
Figure 8 Testing the Xor gate using the N2T Hardware Simulator

Testing in LogicCircuit

If desired, we can add instrumentation in LogicCircuit to verify operation before translation to HDL. For example, if we add two toggle switches and an LED to our Xor circuit, turn on the power, and try all four combinations of the switches, we should get the results shown below.



Note that adding test instrumentation will not adversely impact HDL translation. If we save the Xor to HDL, we will get a warning that the test instrumentation has been ignored, as shown below,



but the generated HDL will be no different than if the instrumentation were absent.