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 Armburst, and their colleagues. You will need the modified version of LogicCircuit (which can be found here), and a number of additional builtInChips (which can be found here). 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 (if you are using Inworks laptops, this version of LogicCircuit is already installed). 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 generated by this code is targeted to the Xilinx Vivado Synthesis Suite, version 2018.3. I have used this code to successfully implement a VHDL version of the N2T CPU using this tool suite (implementing the N2T Computer requires additional platform-specific code). There are many issues related to the correct use of these tools with actual hardware that cannot be easily addressed within LogicCircuit.

I have also written a free standing Nand2Tetris HDL to VHDL translator, but if you are reading this note, you probably don’t care.

Specific Recommendations

1. Do not use underscore (“_”) in names for parts or pins. (N2T HDL compatibility issue).
2. Make sure part pin names match builtIn pin names (from the book) if you want to use builtIn parts with the N2T Hardware Simulator.

3. Do not use the LogicCircuit “Button”, “Sensor”, “LED”, “7-seg”, “LED Matrix”, “Buzzer”, “Probe”, or “Tristate” when designing for HDL translation. These parts have no HDL or VHDL equivalent.

4. Make sure your LogicCircuit project can successfully power-on before exporting to HDL. 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.

5. There is a maximum limit of 100 parts per visible circuit project. If you are approaching this limit, your design is too dense to be readable.

6. In general, HDL does not care (or want to know) about unused inputs and outputs. VHDL does, 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.

7. 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.

8. 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.
9. 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 back to Part 2.

This arrangement is perfectly valid design, however, this circuit, represented as drawn in HDL, will 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.

10. 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, the situation is better, since we can simply use signals and concurrent assignment statements to achieve the desired result.

Therefore, for HDL, we will 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 [firstWireLowRange-lastWireHighRange].”

---

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.
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]”

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.

Both SaveAsHDL and SaveAsVHDL 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 fail silently on any circuits that cannot be translated (e.g., ones that have a “Button”, “Sensor”, “LED”, “7-seg”, “LED Matrix”, “Buzzer”, “Probe”, or “Tristate” present). Such circuits are readily identified by their zero size.