Variations on "From Nand to Tetris" with Logisim and ARM

Ivaylo Ilinkin
iilinkin@gettysburg.edu
Gettysburg College
Gettysburg, PA, USA

ABSTRACT

This paper shares experience with using Logisim to complement the hardware specification approach and accompanying simulator for the projects in "From Nand to Tetris" (N2T; Shocken et al., SIGCSE'09). Student feedback indicates that Logisim enhances the visualization of the main concepts and can be used effectively in parallel with the framework developed in N2T. This is achieved via an external Logisim library implemented by the authors that extends the rich set of components available in Logisim with the specialized circuits required in N2T (e.g ALU, CPU, Screen, Keyboard). Upon completing the projects the students are able to execute the same machine code for a Pong-like game on their Logisim and N2T computers.

CCS CONCEPTS

• **Computer systems organization**;

KEYWORDS

Nand to Tetris, Logisim, hardware description language, ARM assembly

ACM Reference Format:
Ivaylo Ilinkin. 2023. Variations on "From Nand to Tetris" with Logisim and ARM. In *Proceedings of the 2023 Conference on Innovation and Technology in Computer Science Education V. 1 (ITiCSE 2023), July 8–12, 2023, Turku, Finland*. ACM, New York, NY, USA, 6 pages.
https://doi.org/10.1145/3587102.3588858

1 INTRODUCTION

From Nand to Tetris (N2T) is the title of a popular course based on the book *The Elements of Computing Systems* [9] whose conceptual framework is described in [12–14]. Carefully designed abstractions and support software guide the students through the stages of building a computer system from scratch (the *Hack Computer*) including both the hardware circuitry as well as the software layers of an operating system and programming environment. N2T is an ambitious project that takes the students on a tour of much of the fundamental knowledge in computers science. The first six chapters start from the elementary gates and progress through the complete design of a computer, illustrating along the way the concepts of machine and assembly programming by connecting them directly to the hardware specifications of the computer the students have built from scratch. The remaining six chapters lay the software foundation that would, in principle, enable the students to build the promised Tetris game. Here the students are guided towards building a virtual machine, a compiler for a high-level language, and an operating system layer. For instructors of computer organization the first six chapters will be of primary interest and this is the focus of the paper. For the complete details we refer the reader to [9, 12–14].

This paper shares experience with using the N2T approach as part of a one-semester course on computer organization. Since our course has additional demands (including introduction to C/C++) we cover only the first half of N2T, i.e. the hardware platform and assembly language programming, which comprise the first six chapters. The N2T treatment of assembly language is based on their artificial *Hack Assembly*, which we supplement with introduction to ARM Assembly. The projects described here are based on N2T with variations introduced in our course. The projects can be completed during one half to two thirds of a regular semester depending on instructor preference and organization.

2 CONTRIBUTION

The main contribution of this work is a discussion of parallel development of a hardware platform both in the simulation environment of N2T as well as in the digital logic simulator Logisim [4]. N2T offers a wide array of support software including a Hardware Simulator, but unfortunately the development of the *Hack Computer* is achieved exclusively via a *Hardware Description Language* (HDL) designed by the authors of N2T. Thus, both the construction of the hardware platform and the visualization of the signal propagation during circuit operation lack the tangible effect produced by a digital circuit simulator such as Logisim.

Both approaches to circuit design and implementation offer benefits and strengths. Students should be exposed to a high-level hardware specification notation and N2T meets this goal successfully. At the same time, our observation is that a digital logic simulator helps the students conceptualize the work by making the design and visualization more concrete. This observation is supported by course feedback discussed in Section 8.

Logisim already has a rich set of components including many of the ones used in N2T. Our second contribution is the development of a Logisim library for the missing components (e.g ALU, CPU, Screen, Keyboard) so that the students can specify the *Hack Computer* in HDL or design it as interconnected digital components in Logisim. Having a complete set of working components is important for two reasons: the pedagogical reason is to allow the students to make progress even if their implementation at an earlier stage was incomplete; the practical reason is to ensure that the simulation runs at an acceptable rate (the components provided

ITiCSE 2023, July 8–12, 2023, Turku, Finland

ACM ISBN 979-8-4007-0138-2/23/07…$15.00
https://doi.org/10.1145/3587102.3588858

for both N2T and Logisim are implemented differently from what the students are expected to submit).

Two other contributions of this work are the implementation of non-trivial and engaging programs for testing the *Hack Computer* and ideas for integrating ARM Assembly in the N2T projects.

The assignment write-ups, support library **nand2tetris**, and *Hack Assembly* programs (including any future updates) will be available through our course website.

3 RELATED WORK

Our approach is motivated by evidence established in prior research that the study of computer organization is greatly facilitated by good visualization tools. These tend to fall in two categories. *Hardware Emulators* offer a high-level view of a particular computer architecture with data path visualization during program execution. Some examples include the CPU Emulator from the N2T project [9], MarieSim [10], Emumaker86 [2], and the x86 simulator in [3]. *Logic Simulators* are designed for exploration at the lower level, starting with the basic gates and combining them into components of increasing complexity that can culminate in an implementation of a complete computer architecture. Some examples include Logisim [4], DLSim 3 [5], and JLS [11].

Stanley et al. [16, 17] report positive learning outcomes in teaching computer architecture using a logic simulator to realize a particular computer design. Schuurman [15] describes a sequence of assignments that lead to the implementation of a computer architecture and thus shares ideas with N2T and the approach described here.

The list of visualization software is not exhaustive but it shows the interest over the years in creating pedagogical tools to support teaching of computer organization concepts. We chose Logisim since it combines an intuitive interface, a good set of features and components, and a plug-in option for further extensions. Logisim's success as a pedagogical tool is evident through the open-source project *Logisim-evolution* [6] with contributions from a number of academic institutions.

4 HARDWARE PROJECTS

Projects 1, 2, 3, and 5 in N2T introduce the circuits and the design of the computer architecture. Upon completion of Project 5 the students have built a working computer on which they can execute programs provided in the N2T resources or developed as part of the software projects. In our course the students could run the programs both on the HDL-specified computer in the Hardware Simulator offered by N2T as well as on the computer built in Logisim.

This section provides a brief description of the hardware projects. The complete details can be found in the N2T book [9] and the accompanying website. Each project could be completed in one week, but in our course some projects are split since we add the additional requirement to implement the circuits in Logisim. Occasionally, the projects have both a hardware and a software portion. The presentation here follows the sequence and organization in our course as outlined in Table 1.

4.1 Project 1: Elementary logic gates

The requirement is to build `And`, `Or`, `Not`, `Xor` and 2-way multiplexers and demultiplexers `Mux`, `DMux` using only the built-in `Nand` gate. Each of the required gates is implemented in two variants of either 1-bit or 16-bit bus width. Here is the notation of N2T's custom HDL:

```
// chip with 1-bit bus width
CHIP And {
    IN a, b;  // input pins
    OUT out;  // output pins

    PARTS:
    Nand(a=a, b=b, out=nandOut);
    Not(in=nandOut, out=out);

}
```

```
// chip with 16-bit bus width
CHIP And16 {
    IN a[16], b[16];  // 16-bit input bus
    OUT out[16];      // 16-bit output bus

    PARTS:
    Nand16(a=a, b=b, out=nandOut);
    Not16(in=nandOut, out=out);

}
```

This assignment serves as an introduction to HDL and Logisim. It is somewhat repetitive which allows the students to get familiar with the HDL syntax and the editor options in Logisim. We also use it as an opportunity to familiarize the students with the file format for testing circuits by requiring submission of test cases (see Section 6). Instructors can choose the appropriate level of detail.

N2T also includes the requirement to design 4-way and 8-way `Mux`, `DMux`. We defer this to Project 2.

4.2 Project 2: Adders

The requirement is to build `HalfAdder` and `FullAdder` out of elementary gates and use these new chips to build `Add16`, for adding two 16-bit numbers, as well as `Inc16` for adding the constant 1 to a 16-bit number.

In our course here we also complete the 4-way and 8-way `Mux`, `DMux`. We follow the implementation suggestion in N2T to use a recursive approach, i.e. 4-way `Mux` is built out of two 2-way `Mux` and 8-way `Mux` is built out of two 4-way `Mux` (plus an additional 2-way `Mux` in each case). In Logisim the students use several copies of the built-in `Mux` gate by setting its properties appropriately. Similar ideas are used for `DMux`.

4.3 Project 3: ALU, Bit, 16-bit Register

Here we focus on the `ALU` which is part of Project 2 in N2T. The actual implementation of the `ALU` is straightforward, but its design and operation can be difficult to grasp initially, so moving this into its own project allows the students to focus on the main ideas.

We also set the stage for the implementation of the memory chips by implementing the basic building blocks `Bit` and `Register`. This starts the transition from combinational to sequential circuits but the transition is quite manageable and does not detract from the implementation of the ALU.

4.4 Project 4: RAM Chips; Assembly

The requirement is to build `RAM` chips that range in size from 8 (`RAM8`) to 16K (`RAM16K`) of 16-bit addressable units/registers. Here the recursive approach from Project 2 can be applied again, i.e. `RAM64` is built out of 8 `RAM8`, `RAM512` out of 8 `RAM64`, etc. Since the hardware portion of the project is straightforward to implement, we introduce ARM Assembly which roughly corresponds to Project 4 in N2T and is discussed in Section 7.

Table 1: The project sequence described in Sections 4 and 7.

Project 1 HW: Basic Gates	Project 2 HW: Adders, n-Way Mux,DMux	Project 3 HW: ALU, Bit, Register
Project 4 HW: RAM SW: ARM Intro, Arrays	Project 5 SW: ARM Functions, Recursion	Project 6 HW: CPU
Project 7 HW: Memory, Computer SW: Machine Code	Project 8 SW: ARM to Hack Translator MOV,CMP,ADD,JMP,...	Project 9 SW: ARM to Hack Translator LDR,STR,LDM,STM,...

4.5 Project 6: CPU

The requirement is to implement the CPU both in HDL and Logisim. The CPU is the most complicated of the chips implemented so far and is separated in its own project. It requires solid understanding of the function of the ALU, which is perhaps the next component in order of complexity. The CPU design of N2T includes two registers labeled A and D where D is used to store data values and A can be used both as data and as address register.

To simplify the implementation the students are advised to create two sub-circuits: the first sub-circuit controls where the value computed by the CPU should be stored, which could be any subset of A,D,M (M denotes computer memory); the second sub-circuit determines whether a branch/jump should take effect based on the instruction and the value computed by the ALU, i.e. implementation of JEQ,JLT,...,JMP. The details can be inferred from the CPU description in [9].

4.6 Project 7: Memory and Computer; Machine Code Programming

Upon completing this project the students have built a working computer. The two required hardware components — Memory and Computer — are quite straightforward to implement, and therefore, we also include a software portion on writing programs in *Hack Machine Code* and *Hack Assembly*, which is discussed in Section 7.

The Memory includes the 16K RAM built previously as well as memory-mapped Screen of size 8K and Keyboard (a single 16-bit word). This reminds the students of Project 4 in which smaller RAM modules were combined to create a larger RAM. Here Screen and Keyboard are viewed as memory modules and the only challenge compared to Project 4 is that the memory modules are of different sizes, which requires more careful use of Mux and DMux.

Somewhat anticlimactically the Computer is the simplest circuit to build. It is a direct copy of the diagram provided in [9]. The Computer is simply the CPU connected to a ROM and the Memory module. Programs are loaded in the ROM module and their execution produces a result in the Memory (e.g. the n-th Fibonacci number in cell 0 of the RAM, a drawing of a cat on the Screen, etc.).

To test the Computer implementation the students are given two programs created by the authors. The Cat program shows an animation of a running cat. The Pong program is a game where the goal is to move a paddle left/right via the Keyboard and hit a bouncing ball; successful hits are indicated as dots under the paddle.

Figure 1 show screenshots of the two programs running on the same computer implementation in Hardware Simulator and in Logisim. It was particularly rewarding for the students to see the simulation in Logisim.

(Note that an instructor provided Pong implementation does not take away the sense of accomplishment. Writing Pong in *Hack Assembly* would be a challenging and tedious undertaking for the students. The authors of N2T also provide an implementation of a Pong game in the final chapter written in their high-level language *Jack* which is the focus of the second half of N2T. Although this would be a more feasible project, it is wisely left for independent exploration.)

5 N2T LIBRARY FOR LOGISIM

This section describes the extension to Logisim that enables the parallel implementation of the N2T projects. The N2T support software contains default implementation of all circuits that are part of the hardware projects. Whenever a circuit implementation is not found in the project directory the Hardware Simulator uses a built-in implementation. The suggested workflow is to develop each project in a separate folder and only implement the required circuits. This ensures that progress is made even if a circuit from a previous project was not implemented correctly, since the Hardware Simulator will use the built-in implementation. Moreover, the built-in implementation is faster and should be preferred (particularly relevant for the RAM modules in Project 4).

Logisim has a rich collection of components including those used in N2T. Therefore, direct implementation in Logisim of the computer architecture proposed in N2T is feasible. Only a small set of N2T components are not available in Logisim: ALU, Inc, PC, CPU, Screen, and Keyboard. (Versions of Screen and Keyboard are available, but do not have the same interface as required by N2T). Fortunately, users of Logisim can extend it via external JAR libraries that contain Java implementations of the desired components. The Java implementations must conform to a set of design and implementation guidelines described on the Logisim website.

We created **nand2tetris.jar** which contains the missing components and can be used by instructors who would like to develop the N2T projects in Logisim. Figure 2 shows **nand2tetris.jar** loaded in Logisim with its components arranged on the canvas. This ensures that students can continue to make progress in each new stage in case there were difficulties in a previous assignment.

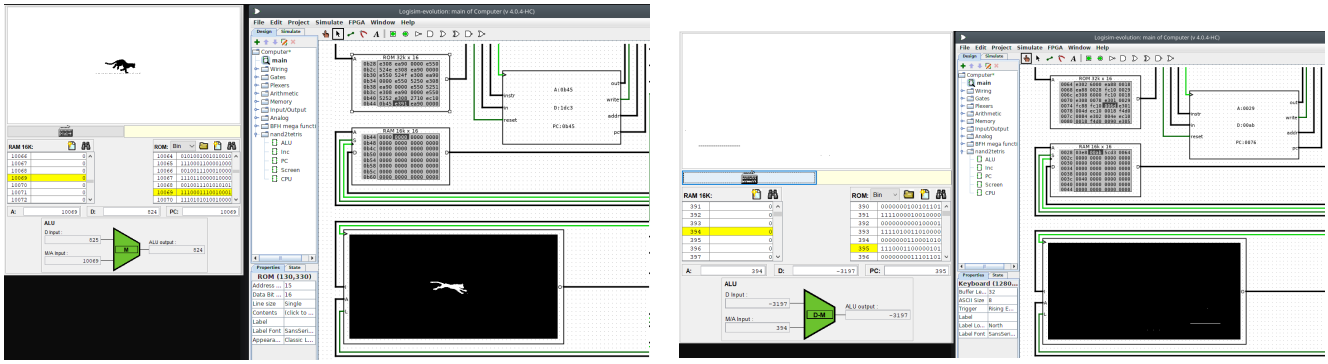

Figure 1: The `Cat` and `Pong` programs executed on Hardware Simulator and Logisim

6 CIRCUIT TESTING

This section includes a brief discussion on circuit testing, both for students during project development and for instructors during grading. The N2T support materials include test data for each circuit that is split into two files:

- **.tst** is a script that sets the circuit inputs and triggers signal propagation
- **.cmp** is a truth table that shows the expected values of the outputs for given values of the inputs; for sequential circuits the *clock* is part of the inputs

The **.tst** file is loaded in Hardware Simulator and after executing the script (step-by-step or full run) the Hardware Simulator compares the output with the expected results in the **.cmp** file. The Hardware Simulator also has command line mode that allows instructors to test one or several circuits from the command line.

Logisim has a similar testing feature. Test cases are specified in a format close to the N2T **.cmp** file and after execution Logisim generates a report that highlights the failed cases. The test can be run within Logisim or from the command line for grading. Unfortunately, the testing feature of Logisim does not work for sequential circuits. We plan to consider this in future work and share it with the community in order to facilitate testing of the projects.

7 SOFTWARE PROJECTS

In this section we describe the software projects used in the course. Some of the projects are motivated by N2T, while others are designed to meet the specific needs of our course.

7.1 Project 4 & 5: ARM Assembly

N2T introduces assembly programming via a language designed by the authors called *Hack Assembly*. Unfortunately, *Hack Assembly* is not representative of a typical assembly language, and therefore, we have opted to supplement the discussion on assembly languages with an introduction to ARM Assembly.

This introduction is split in two parts. Project 4 described in Section 4.4 contains the first part of ARM Assembly programming which is an implementation of *Bubble Sort*. Prior to this assignment the students have been introduced to the concepts of assembly programming including how to express conditions and repetition and

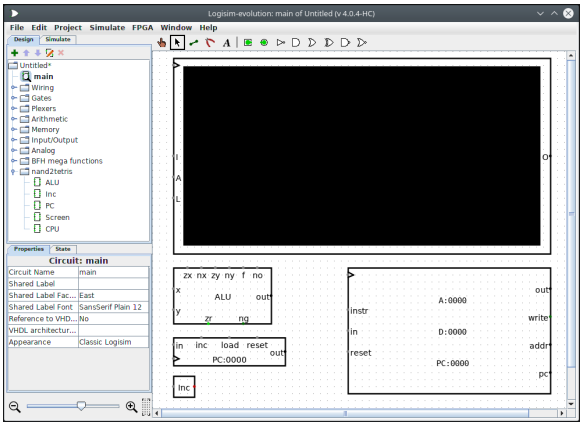

Figure 2: The `nand2tetris.jar` library created for our course to provide the missing N2T components. Currently, `Keyboard` is given as an actual circuit, but will eventually be incorporated in the library.

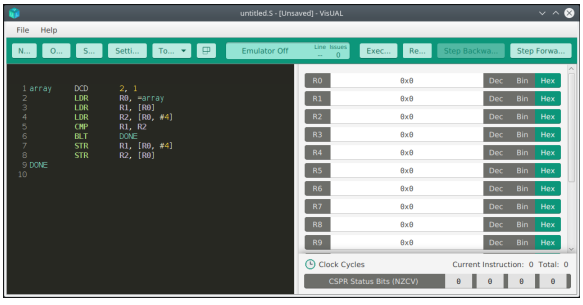

Figure 3: The VisUAL ARM emulator.

have seen examples that add the numbers from 1 to n or add all numbers in an array.

For the implementation of *Bubble Sort* the students are guided through a series of stages:

(1) carry out a single pass through the array exchanging every pair of elements that are out of order

(2) wrap the code from the previous step in a loop that runs n-1 times; this guarantees that the array is sorted
(3) modify the code from step 1 so that after a complete pass through the array is made, register `R10` contains either 0 or 1 to indicate whether a swap occurred
(4) modify the loop from step 2 to stop when a complete pass has left register `R10` with 0, i.e. no swap took place

The following week the students complete a software only project on ARM Assembly that covers the concepts of stack frames to support function calls, parameter passing, and recursion. As part of this assignment the students develop recursive implementations of computing the *n*-th Fibonacci number and *Binary Search*.

We use VisUAL [1], "A highly visual ARM emulator", which offers a beginner-friendly environment for writing ARM Assembly programs (Figure 3). It supports syntax highlighting, automatic indentation, single-step execution, and register and memory viewer. The set of ARM instructions supported by VisUAL is rich enough to serve as a solid introduction to ARM Assembly programming. The free textbook *Dive into Systems* [7, 8] has a good treatment of Assembly programming, in general, and ARM, in particular.

7.2 Project 7: Machine Code

Project 7 described in Section 4.6 contains a software portion on machine language based on *Hack Machine Code*. Unlike *Hack Assembly*, *Hack Machine Code* is representative of the main ideas that one might wish to present on machine language programming. This portion of the project asks for an iterative implementation of a program that computes the *n*-th Fibonacci number. Overall, the students seemed comfortable with understanding the instruction format and putting together instructions in binary form to carry out the various steps. Along the way the students were introduced to *Hack Assembly* which happens to be a good shorthand for machine language instructions.

Both Logisim and Hardware Simulator make it possible to load the program in `ROM` and trace through the execution one instruction at a time. Hardware Simulator can display the instructions in a variety of formats, including *binary*, *hexadecimal*, and *assembly*, while Logisim restricts the display to *hexadecimal*. However, observations from debugging sessions with students suggest that the display format of the instruction is less important then the intended effect. In other words, students have a sense of what changes are expected to happen in memory and in what order. If the intended effect is not observed after executing a particular instruction, it is examined more closely to determine which bits are set incorrectly (often due to a typo or accidental bit insertion/deletion).

7.3 Project 8 & 9: Assembler/Translator

Chapter 6 of N2T concludes with a project on writing an *Assembler*. The goal is to translate a program written in *Hack Assembly* to *Hack Machine Code*. We modified the project to be on writing a *Translator* from ARM Assembly to *Hack Assembly* (it would have been a trivial change to write *Hack Machine Code* directly).

The *Translator* was implemented in C++ and was split over two assignments that were completed at the end of the semester. The *Translator* supported all ARM instructions that were used as part of the software projects in Section 7.1. The only handout given to the students were functions to read a line from a file and to extract tokens from the line. The students maintained their own lookup tables for labels and identifiers for a two-pass translator.

The implementation had a simple structure: a main loop reading each ARM Assembly line that dispatched to the appropriate method based on the first token in the line (the ARM instruction). The first translator assignment focused on the basic instructions (e.g. `MOV`, `CMP`, `ADD`, `SUB`) as well as branch instructions and lookup tables. The second assignment added support for arrays and function calls (e.g. `LDR`, `STR`, `LDM`, `STM`, `DCD`, `BL`). The ARM registers were mapped to virtual registers at addresses 0-15 in the *Hack Computer*.

With this project the students could translate their ARM programs from Section 7.1 and execute them on the *Hack Computer* either in the Hardware Simulator or in Logisim.

8 FEEDBACK

The main focus of this work was to explore the effectiveness of a dual approach for implementing the N2T projects using both HDL and Logisim. We expected that the two approaches will complement each other and that the more concrete development and simulation environment of Logisim with its immediate visualization of the actual circuits and signal propagation will enhance learning and appreciation for the projects.

The results from a pilot survey are summarized in Tables 2 and 3 in this experience report (there were 14 responses out of 18 enrolled students). The feedback suggests that the dual approach was effective in teaching computer organization concepts. The responses give strong endorsement of building the computer in Logisim and the comments in the freeform question suggest that there is benefit in using both the high-level descriptive approach of HDL and the low-level concrete construction in Logisim.

Overall, it appeared that the students followed a development process that alternated between the two approaches. The preferred workflow was to create the initial implementation in Logisim since it provides a more concrete realization of the design. After initial testing the Logisim implementation was translated in HDL. Testing iterated between the two implementations which was helpful in illustrating the connection between the two different forms of expressing the design. The student feedback has given strong endorsement of the current approach and we plan to refine it in future iterations of the course.

9 CONCLUSION

This paper shared experience with teaching computer organization based on the course *From Nand to Tetris* [9, 12–14]. The main contributions of this work include:

- Ideas for integrating the N2T projects with Logisim.
- Support library **nand2tetris.jar** for missing components.
- Implementation in *Hack Assembly* of engaging non-trivial test programs `Cat` and `Pong`.
- Suggestions for integrating ARM Assembly programming with the *Hack* platform.

Out hypothesis was that the concrete visualization of digital circuits and signal propagation offered by a digital circuit simulator such as Logisim will enhance the understanding and the appeal of the computer organization topics presented in N2T. Student

Table 2: Survey Feedback: SD=Strongly Disagree, D=Disagree, N=Neutral, A=Agree, SA=Strongly Agree.

		SD	D	N	A	SA
1.	The Nand2Tetris projects were effective in teaching computer architecture concepts. (The question is about the projects on their own, separate from HDL and Logisim.)	1	0	0	7	6
2.	The use of Logisim enhanced my understanding of the Nand2Tetris projects and computer architecture concepts.	0	0	0	0	14
3.	The use of Logisim enhanced my understanding of how software and hardware interact.	0	0	0	5	9
4.	The use of Logisim motivated me to complete the projects.	0	0	0	2	12
5.	Logisim should be used in future iterations of this course.	0	0	0	1	13
6.	HDL and Hardware Simulator enhanced my understanding of the Nand2Tetris projects and computer architecture concepts.	1	0	4	6	3
7.	HDL and Hardware Simulator were helpful in completing the Nand2Tetris projects.	1	0	4	5	4

Table 3: Freeform Question: Please share your thoughts on the answers you gave to the previous questions.

- *I leanred a lot through the use of Logism and HDL. Being able to slowly move up in the complexity of the chips helped me get a better grasp of the how it work. Overall, pleased with Nand2Tetris with the use of logism and HDL*
- *I really enjoyed using Logisim and I thought it helped a lot towards deepening my understanding of the concepts. HDL and Hardware Simulator also helped, but less so, because I felt like I was just copying my Logisim project in a different format.*
- *I really enjoyed everything about Logisim/Hardware simulator. It helped me visualize all of the chips being made and allowed me to work with them to further conceptualize how they worked. Using the two different mediums also enhanced understanding by making me think about implementing the chip using different formats.*
- *Overall, the logisim was always easiest to complete at first for the Nand2Tetris projects. It gave a nice visual to understand what was happening the circuits and we could manually test different cases ourselves. This is return made it easier to do the Hardware HDL part.*
- *I really liked working with Logisim. I think its visual demonstration of bits and gates is very informative. For HDL, I would say it was less informative but it still helped me understand the ins and outs of Logisim.*

feedback from a pilot survey included in this experience report offers evidence that the approach has been successful in meeting the goals.

REFERENCES

[1] Salman Arif. 2015. VisUAL: A Highly Visual ARM Emulator. https://salmanarif.bitbucket.io/visual/index.html

[2] Michael Black and Nathaniel Waggoner. 2013. Emumaker86: A Hardware Simulator for Teaching CPU Design. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (Denver, Colorado, USA) *(SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 323–328. https://doi.org/10.1145/2445196.2445294

[3] Michael David Black and Priyadarshini Komala. 2011. A Full System X86 Simulator for Teaching Computer Organization. In *Proceedings of the 42nd ACM Technical Symposium on Computer Science Education* (Dallas, TX, USA) *(SIGCSE '11)*. Association for Computing Machinery, New York, NY, USA, 365–370. https://doi.org/10.1145/1953163.1953272

[4] Carl Burch. 2002. Logisim: A Graphical System for Logic Circuit Design and Simulation. *J. Educ. Resour. Comput.* 2, 1 (March 2002), 5–16. https://doi.org/10.1145/545197.545199

[5] John L. Donaldson, Richard M. Salter, Joe Kramer-Miller, Serguei Egorov, and Akshat Singhal. 2009. Illustrating CPU design concepts with DLSim 3. *2009 39th IEEE Frontiers in Education Conference* (2009), 1–6.

[6] Logisim-evolution. 2022. https://github.com/logisim-evolution

[7] Suzanne J. Mathews, Tia Newhall, and Kevin C. Webb. 2022. *Dive into Systems.* https://diveintosystems.org/

[8] Suzanne J. Matthews, Tia Newhall, and Kevin C. Webb. 2021. *Dive into Systems: A Free, Online Textbook for Introducing Computer Systems.* Association for Computing Machinery, New York, NY, USA. 1110–1116 pages. https://doi.org/10.1145/3408877.3432514

[9] Noam Nisan and Shimon Schocken. 2001. *The Elements of Computing Systems.* MIT Press, New York, NY. https://www.nand2tetris.org/

[10] Linda Null and Julia Lobur. 2003. MarieSim: The MARIE Computer Simulator. *J. Educ. Resour. Comput.* 3, 2 (June 2003), 1–es. https://doi.org/10.1145/982753.982754

[11] David A. Poplawski. 2007. A Pedagogically Targeted Logic Design and Simulation Tool. In *Proceedings of the 2007 Workshop on Computer Architecture Education* (San Diego, California) *(WCAE '07)*. Association for Computing Machinery, New York, NY, USA, 1–7. https://doi.org/10.1145/1275633.1275635

[12] Shimon Schocken. 2012. Taming Complexity in Large-Scale System Projects. In *Proceedings of the 43rd ACM Technical Symposium on Computer Science Education* (Raleigh, North Carolina, USA) *(SIGCSE '12)*. Association for Computing Machinery, New York, NY, USA, 409–414. https://doi.org/10.1145/2157136.2157259

[13] Shimon Schocken. 2018. Nand to Tetris: Building a Modern Computer System from First Principles (Abstract Only). In *Proceedings of the 49th ACM Technical Symposium on Computer Science Education* (Baltimore, Maryland, USA) *(SIGCSE '18)*. Association for Computing Machinery, New York, NY, USA, 1052. https://doi.org/10.1145/3159450.3162353

[14] Shimon Schocken, Noam Nisan, and Michal Armoni. 2009. A Synthesis Course in Hardware Architecture, Compilers, and Software Engineering. In *Proceedings of the 40th ACM Technical Symposium on Computer Science Education* (Chattanooga, TN, USA) *(SIGCSE '09)*. Association for Computing Machinery, New York, NY, USA, 443–447. https://doi.org/10.1145/1508865.1509021

[15] Derek C. Schuurman. 2013. Step-by-Step Design and Simulation of a Simple CPU Architecture. In *Proceeding of the 44th ACM Technical Symposium on Computer Science Education* (Denver, Colorado, USA) *(SIGCSE '13)*. Association for Computing Machinery, New York, NY, USA, 335–340. https://doi.org/10.1145/2445196.2445296

[16] Timothy Stanley, Nathan I. Kim, Yiu-Ming Chan, Ji Zheng, and Leslie Fife. 2009. Experiences in Teaching Computer Architecture. *J. Comput. Sci. Coll.* 24, 4 (apr 2009), 46–52.

[17] Timothy D. Stanley, Thanh Quach Xuan, Leslie Fife, and Don Colton. 2007. Simple Eight Bit, Emulated Computers for Illustrating Computer Architecture Concepts and Providing a Starting Point for Student Designs. In *Proceedings of the Ninth Australasian Conference on Computing Education - Volume 66* (Ballarat, Victoria, Australia) *(ACE '07)*. Australian Computer Society, Inc., AUS, 141–146.