Circuitly: A visual and constructive framework for teaching digital circuits

Lucas Castro
Institute of Computing - University of Campinas (UNICAMP)
Campinas, São Paulo - 13083-852
Email: lcbc.lucascastro@gmail.com
Rodolfo Azevedo
Institute of Computing - University of Campinas (UNICAMP)f
Campinas, São Paulo - 13083-852
Email: rodolfo@ic.unicamp.br

***Abstract*—This paper describes an interactive and student-friendly framework for teaching digital circuits and computer architecture topics. It aims to improve students learning process by providing a visual drag-and-drop circuit design editor, interactive simulation, signal monitoring and testbench tools - all integrated in a widely accessible application that runs in the browser. Circuitly does so in a programmatic way, to help students better understand the Hardware Description Languages they will encounter in the future.**

I. Introduction

To create a computer based on abstraction layers, we need to start in a very low level and design, layer by layer, new capabilities and functionalities. As example, we can start with a NAND gate - which abstracts a few transistors. It can be used to build other logical gates, such as ANDs, NOTs and ORs and all other combinatorial logic. After that, we need to create the concept of memorization and sequential circuits come into play. We keep evolving, developing larger blocks of logic until we have a set of hardware which compose the Von-Neumann architecture in which our computers are based today.

The hardware interface with software starts with the instruction set, where programs are expressed. Then we can create new functionalities with the lowest level drivers that configure the hardware and ensure it is up and running, followed by an operating system providing the essential resources to the applications - which may be tools, libraries or frameworks to allow higher level applications to do their jobs. This stack keeps growing, raising the abstraction level that allows us to solve the ever more complex problems we must face in computing. Students must be able to understand and learn how this stack is built over layers of abstractions, as well discussed by Jeff Kramer in his work *Is Abstraction the Key to Computing?* [1].

Shimon Schocken and Noam Nisan wrote a book proposing a constructive approach for teaching computing. The student starts with a NAND gate and iteratively builds larger components up to the point it is possible to write a Tetris game in a higher level language and run it in the custom processor developed along the course [2], [3]. This approach provides a wider vision for the student of how computing is built, how each topic relates to others and why we need the abstraction layers to solve complex problems.

Our work values and is inspired by Schocken and Nisan's *"From Nand to Tetris"* approach. We wanted to apply this gradual learning concept in a new user friendly framework, named Circuitly (www.circuitly.app) [1], aiming to improve some aspects in the students learning process when it comes to digital circuits and computer architecture. Circuitly allows students to visually design circuits with modular reusable building blocks as circuit components, visualize and interact with the circuit simulation, run automatic tests and create larger logical blocks from smaller ones. We also took special care so it could be widely accessible and platform independent, ensuring that no restrictive tools or environments would be a barrier for students or self learners who want to use it. Instead of designing with circuit diagram blocks, we opted to use a more text-like design, although based on blocks, to better engage the student with Hardware Description Languages that (s)he will use in the future.

This work aims to improve teaching digital circuits and computer architecture related topics by developing a student friendly framework which considers digital circuits as modular building blocks, provides interactive visual simulation, is available across multiple platforms and allows constructive bottom-up teaching approaches, such as the proposed by Schocken and Nisan [2]. In order to achieve this goal we used other existent tools, such as Blockly [6] which is a Javascript framework for developing visual programming languages and DigitalJS [7] which is a visual and interactive digital circuits simulator that supports SystemVerilog and runs on the browser.

II. Related Work

Considering the aspect of using a constructive hands-on approach for teaching digital circuits, *"From Nand to Tetris"* [2], [3], [8] is a very relevant work in which we were inspired. It is a consolidated course that guides the student's path to build an entire computing system starting with the NAND

[1]Circuitly is open source [4] and has a web page [5] with additional information about the project.

logic gate, including a full textbook, laboratories material, hardware simulator and other tools to be used by the student.

Regarding other tools and frameworks for digital circuits that run on the browser, an important example is EDA Playground [9] which has many prototyping features and supports many simulation backends, but it is not specifically designed for teaching. Falstad [10] is also a relevant tool, supporting interactive circuit simulation on browser, however it is focused on electronic circuits and has restrict support for logical circuits. Another relevant work is DigitalJS [7] which is a promising digital circuits simulation tool for teaching purposes that allows students to live interact with their simulated circuits and watch their behaviors. DigitalJS is one of the base tools used in this work, as detailed along the article.

Another main topic in this work is using visual programming languages for teaching. Although this is not (yet, hopefully) a common topic for digital circuits, there are many successful projects implementing this concept for software teaching, such as Scratch [11] and Snap! [12]. An important project in this regard is Blockly [6] which is a consolidated and extensible framework for preparing custom visual block based programming languages. "Blockly OpenCL" [13] is one example of a teaching software built upon Blockly framework, focusing on parallel programming concepts and architectures. Blockly is also an important framework used in this work as will be detailed in the following sections.

III. Circuitly Design and Implementation

A. Overview

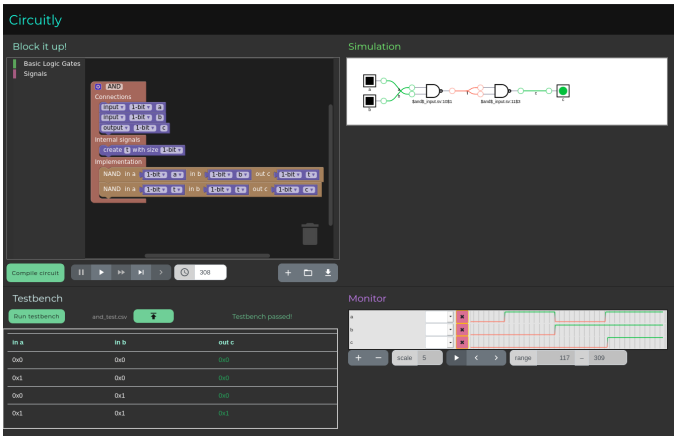


Fig. 1: Circuitly user interface Overview

Figure 1 shows an overview of Circuitly user interface. The framework has the following main features to help students design their circuits:

- **Multi-platform available**: This was a project requirement, we wanted it to be as much available and easy to run as possible. We describe how it was addressed in development in section III-B.
- **Blockly editor** (In Figure 1 as "Block it up!") The editor where students can design their circuits as modular blocks. We prefer a language-like style to better introduce Hardware Description Languages in a next stage. This is discussed at section III-C.
- **Circuit Simulation and Monitoring**: Visual simulation and signals monitoring for the designed circuit. This is presented at section III-D.
- **Testbench**: This is a utility that allows teachers to provide a series of test cases to validate the circuit implementation. Testbenches allow students to self-check their work and also instructors to autograde students' solutions. The testbench functionality is described at section III-E.

Figure 2 shows an overview of how Circuitly combines these features internally. This flow is better explained in the following sections.

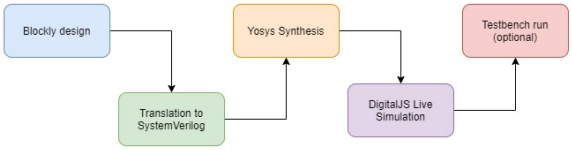


Fig. 2: Circuitly internal flow diagram.

B. Ensure wide and easy access in all platforms

The objective which most affects development is ensuring the framework is widely accessible and platform independent since it limits the tools we are allowed to use. Native applications would be a poor choice, it would not be easy to support the most used operating systems. We could make a Java based application since it would run in every machine which supports a JVM or a Web based application which is an application that runs on the browser. We decided for the latter (Web) to be really able to run everywhere and tried to remove the requirement to install software. It also runs on cellphones, although the visual interface would require improvements for better user experience in this platform.

We selected libraries and frameworks available for Web as the base infrastructure for this project. It is nodeJS based [14] and has many small dependencies, however the main features which are detailed in the following sections are built upon Blockly [6] and digitalJS [7].

C. Designing digital circuits as modular building blocks

The next requirement is to allow students to see circuits as building blocks. It should be natural that smaller blocks of logic could be used to build larger blocks. We also wanted that students could skip the process of learning a Hardware Description Language (HDL) in the first moment for two main reasons (we kept in mind that they will have to learn HDL in the future if they want to be hardware designers):

- Hardware description languages have some different programming concepts when compared to software programming languages since circuits are different from machine instructions which may confuse beginner students;
- Allowing students to design hardware as building blocks permit them to visually see and experiment with the circuit components instead of trusting a synthesis tool that transforms their code into a logical circuit. We believe that this closer contact will be very beneficial for learning;

We also did not want the student to design circuits as diagrams since:

- Designing larger circuits using only diagrams without the assistance of HDL description logic is not very scalable;
- We believe that a higher level of abstraction would be beneficial for learning the concepts behind larger circuits and understanding what are the "smaller building blocks" used to build it;
- We expect the student to have contact with diagrams and HDL in more advanced courses, the goal is to pave the way between their first contact and such courses;

Based on these constraints, we selected a block design that has a good visual appeal, is easy to use, and provides a clear path to future HDL. Blockly [6] has done a very good work in designing a customizable block based programming language and we decided to use it to build the Circuitly design editor.

1) Using Blockly to create a digital circuits editor:

Blockly [6] is, from Blockly developers own words: *"a library that adds a visual code editor to web and mobile apps. The Blockly editor uses interlocking, graphical blocks to represent code concepts like variables, logical expressions, loops, and more. It allows users to apply programming principles without having to worry about syntax or the intimidation of a blinking cursor on the command line."* [15].

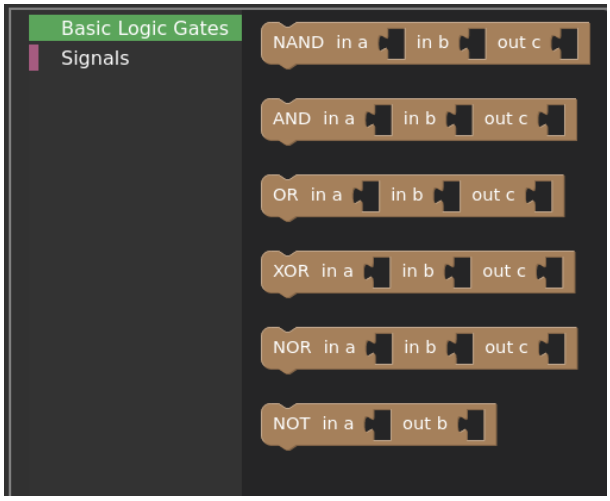
Blockly fits our purposes to create a block based code editor for digital circuits. Although Blockly does not support any HDL officially, it allows developers to create their own blocks which generate custom code - in other words, we can generate SystemVerilog from Blockly blocks. Blockly also provides a great set of customization options for custom blocks such as their shapes, colors, type restrictions and code generation instructions.

Figure 3 shows a set of custom blocks developed for Circuitly as example. These blocks are used in a drag-and-drop fashion. In Figure 3a there are examples of logical gates that the student can use in their designs and in 3b there are blocks related with the abstraction for "signals interconnection" created in Circuitly that allows users to interconnect the blocks in the design. This abstraction was based on the Verilog HDL syntax and gives the students the flexibility to create signals; attach them in the blocks inputs and outputs, thus connecting these blocks; assign signals to other signals; split, concatenate and partition signals.

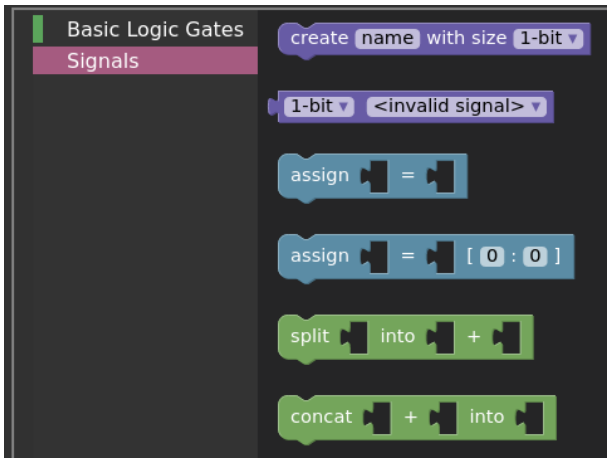
The custom blocks developed for Circuitly, such as those in Figure 3, generate valid SystemVerilog code. Users can use the high level block description to build their circuits and their work will be converted to SystemVerilog in the background. This is important so we can still use existing synthesis and simulation tools while providing the higher level design layer for students.

Figure 4 shows an example of a Blockly workspace with custom blocks and the SystemVerilog code generated from it.

2) Constructive development of more complex circuits:



(a)



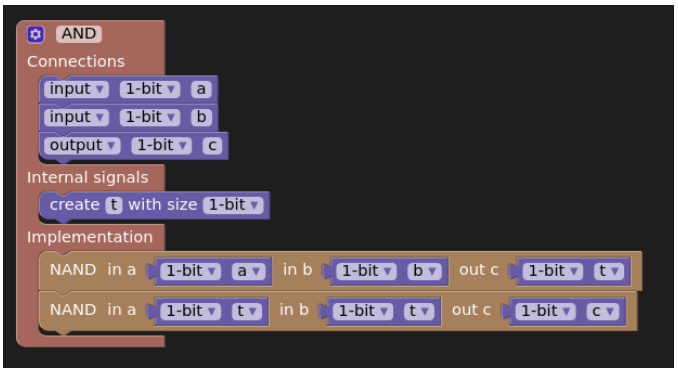
(b)

Fig. 3: Example of custom Blockly blocks for logic circuits design.

As discussed in the introduction, this work values the *"From Nand to Tetris"* [2], [3] approach and wants Circuitly to be compatible with it.

Blockly natively allows us to prepare workspaces - which are prepared environments where we determine what blocks students can use to do their work. It is straightforward to prepare consecutive laboratories asking students to build blocks that will be used in the next exercises using only the blocks they have already prepared previously. Following are two simple descriptions of laboratories to exemplify the concept:

- **AND Laboratory: Develop AND using only NAND gates**;
  It is a workspace where students only have access to NAND blocks to implement the AND behavior. This laboratory would not have any pre-requisites since NAND is the most primitive logic block;
- **DFF Laboratory: Develop Data Flip Flop (DFF) using only NOT and NAND gates**;
  It is a workspace where students only have access to NAND and NOT blocks to implement the DFF behavior. Note that this laboratory should be held after the "NOT




```
module AND(
    input a,
    input b,
    output c
);

    wire t;
    nand nand_0(t, a ,b);
    nand nand_1(c, t ,t);

endmodule
```

Fig. 4: Example of Blockly workspace with custom blocks and generated SystemVerilog code.

Laboratory" since the NOT block is required here.

Following *"From Nand to Tetris"* roadmap, this sequence of laboratories should constantly get more complex towards the point the student is able to build a minimal processor and run some code on it. This is better discussed at section V.

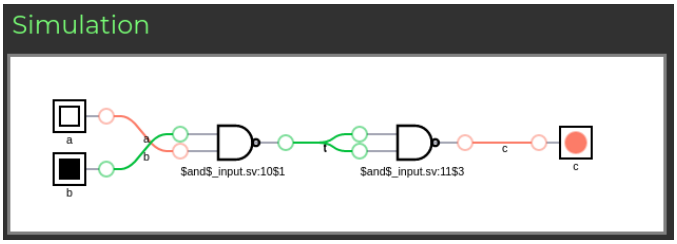
D. Interactive circuit visualization and monitoring

Blockly provides a very good framework to develop the editor students will use to build their logical circuits. The next step is synthesizing the SystemVerilog code we will generate from the custom blocks we developed for Circuitly and simulate it with a visual and interactive tool.

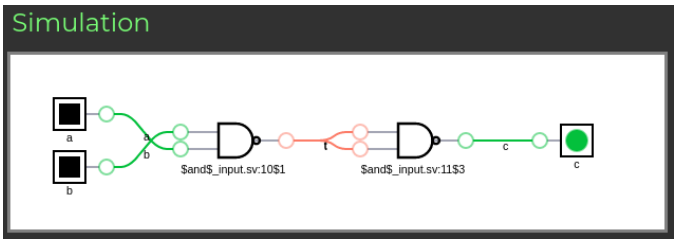
1) Simulation and Monitoring with DigitalJS:

Marek Materzok worked on "a visual circuit simulator tool designed for teaching students digital circuit design" named DigitalJS [7]. It is a Javascript framework that uses Yosys [16] to synthesize Verilog/SystemVerilog code and simulate the circuit on the browser. It has a friendly design, is easy for the students to use and interact with. It also has a monitor to watch signals and trigger event-based breakpoints what is very useful for better understanding the circuit and debug it.

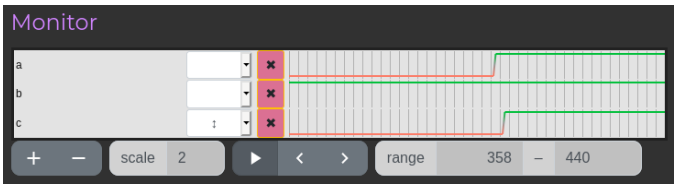
Figure 5 shows digitalJS simulation for the automatically generated SystemVerilog code shown in Figure 4. Users can toggle the input buttons and dynamically see how the circuit behaviors. Considering the circuit state in 5a, toggling input 'a' will change the circuit to the state shown in 5b. Users can also take a closer look in the signals transitions over time in the monitor, as shown in 5c.



(a)



(b)



(c)

Fig. 5: Simulation and monitoring example of the AND block shown on figure 4

E. Testbench

The possibility to run automatic tests is very useful for students to check their work. Circuitly has a "Testbench" section that accepts an input file describing a list of test cases with input signals configuration and expected values for the outputs. The test cases are run sequentially and indicates which tests succeeded and which failed.

This utility is available both in the user interface and as a runnable script, allowing teachers to automatically correct assignments.

IV. Using Circuitly

In section III we presented Circuitly main features using an AND block as example. Following is a more objective demonstration of building a logic circuit in Circuitly focused in how students would work with the platform.

A. Building Data Flip Flop (DFF) block demonstration

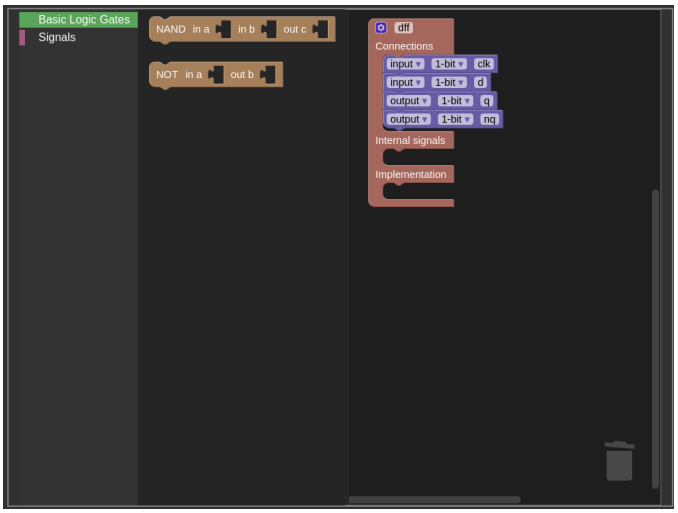
Firstly, it is expected that the laboratory specifies the desired module and the allowed building blocks. For this example, suppose our laboratory is:

Build a Data Flip Flop (DFF). The only blocks allowed are: NOT and NAND. DFF should contain the following connections:

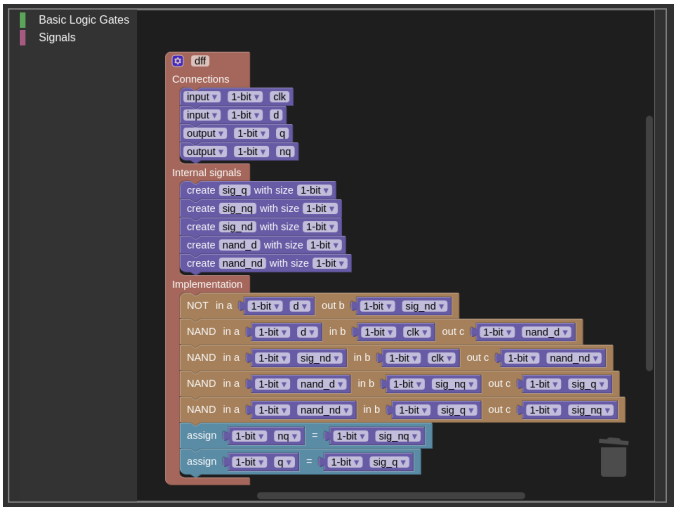
- ***input**: clk - Clock source;*
- ***input**: d - Data signal;*
- ***output**: q - Sampled data;*

- ***output***: *nq - Negated sampled data;*

This specification can be reinforced in the editor's workspace and toolbar by placing only the allowed blocks in the toolbar and preparing a 'module shell' with the expected connections in the workspace, as shown in Figure 6a (this is a feature built upon Blockly workspaces concept). Students can use the drag-and-drop editor, as discussed in section III-C, to implement DFF behavior. One possible result is shown in Figure 6b.



(a)

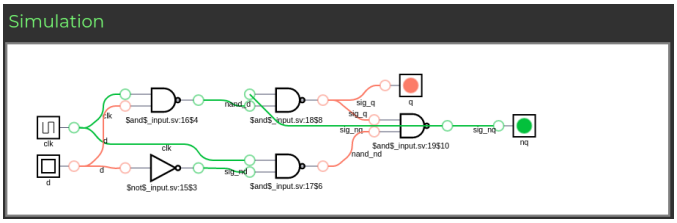


(b)

Fig. 6: Building a DFF using Circuitly Blockly based editor.

The user can *"Compile"* the design any time. If the design is valid, it will generate SystemVerilog code and synthesize it in the background, as shown in section 4. The interactive simulation starts, allowing the user to experiment with his/her work. Figure 7 shows possible states of the simulation for this circuit.

Lastly, students have to check if the circuit is valid accordingly to some predefined tests. Circuitly testbench supports input files in CSV format describing the input signals configurations and the expected output results. It runs each test case and generates a table displaying the circuit behavior in



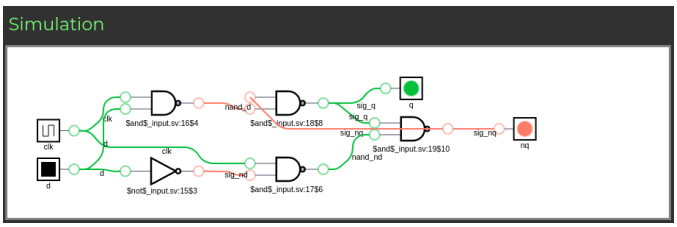
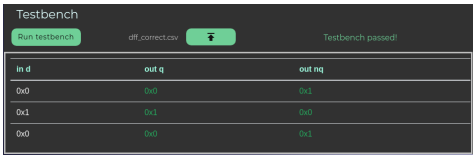


Fig. 7: Simulation examples for the module of figure 6

the test. Figure 8a shows a testbench input for this example and Figure 8b the corresponding testbench result.

d	q	nq
0	0	1
1	1	0
0	0	1

(a)



(b)

Fig. 8: Automatic testbench validation of the DFF shown in figure 6

During this process, students can save and load their work. If desired, it is possible to setup a submission platform to retrieve the generated modules and setup an environment to automatically run the testbench from command line to validate the students submissions - although this was not done yet and is marked as future work.

V. Future Work

A. Build a RISC-V processor using Circuitly

As discussed along the article, this work was inspired by Schocken and Nisan's *"From Nand to Tetris"* constructive teaching approach. We intend to create a similar set of laboratories using Circuitly, starting with a NAND block and creating new larger blocks until it is possible to build a RISC-V [17] processor - which is a significant change to Schocken and Nisan's work, since they decided to go for a custom architecture.

We intend to use RISC-V for the following reasons:

- RISC-V is a RISC (*Reduced Instruction Set Computer*) ISA, its smallest variant is RV32i [17] which has a very small set of instructions that can be easily implemented in the scope we want for Circuitly laboratories;
- It is used for real world applications;
- There is a growing community involved in RISC-V projects that can provide help or attract students to further study this ISA;

- It is an open architecture, there is no legal related restrictions preventing the development and usage of RISC-V processors for academic courses, which could be a barrier in proprietary ISAs;

B. Circuity Improvements

Although Circuitly already provides a working interface with all features described in the previous sections, there is still a lot of improvements to be made and features to be added. Following is a list of what we consider the most important improvements to be done:

- Study the possibility to use the Javascript build of Yosys, YosysJS [18], instead of the native Linux one. This would allow Circuitly to be completely client-side, requiring no server;
- Allow users to import SystemVerilog files and automatically create their blocks, allowing Circuitly to be compatible with existing SystemVerilog circuits;
- Allow users to import a "module" block and automatically create the corresponding "instantiation" block. Currently, it is necessary to manually create the "instantiation" block and insert it in the workspace toolbar through code;
- Study the possibility to ship Circuitly with an integrated submission platform or provide some API for external tools;
- Better errors handling and notifications for users;

Finally, we also intend to present the framework to students and listen to their feedback. The main goal of this work is to improve their learning, consequently their opinions and judgment is extremely valuable to improve Circuitly.

VI. Conclusion

We designed and implemented Circuitly, an interactive and student-friendly framework for teaching digital circuits. It manages to conciliate existing tools for synthesis and simulation with teaching methods and frameworks that are unexplored for digital circuits and computer architecture, such as block based programming with Blockly. It also brings an important feature of being openly available supporting many platforms as a Web app.

We consider Circuitly a promising framework for teaching digital circuits, although there are still many improvements and functionalities to be added in follow-up projects in order to make it truly student-friendly, easy to use for academic courses and able to provide a set of laboratories guiding the student's path to build a RISC-V processor from the ground-up.

VII. Acknowledgement

This work is supported by the Sao Paulo Research Foundation (FAPESP) (2013/08293-7), CAPES (2013/08293-7), and CNPq (438445/2018-0, 309794/2017-0).

References

[1] J. Kramer, "Is abstraction the key to computing?" *Commun. ACM*, vol. 50, no. 4, p. 36–42, Apr. 2007. [Online]. Available: https://doi.org/10.1145/1232743.1232745

[2] S. Schocken and N. Nisan, *The Elements of Computing Systems*. The MIT Press, Mar. 2005.

[3] S. Schocken, N. Nisan, and M. Armoni, "A synthesis course in hardware architecture, compilers, and software engineering," in *Proceedings of the 40th ACM Technical Symposium on Computer Science Education*, ser. SIGCSE '09. New York, NY, USA: Association for Computing Machinery, 2009, p. 443–447. [Online]. Available: https://doi.org/10.1145/1508865.1509021

[4] L. Castro, "Circuitly repository," https://github.com/lcbcFoo/circuitly, accessed July 12, 2020.

[5] L. Castro, "Circuitly," https://circuitly.app, accessed July 12, 2020.

[6] The Blockly Team, "Blockly - A JavaScript library for building visual programming editors." https://developers.google.com/blockly, accessed July 12, 2020.

[7] M. Materzok, "Digitaljs: A visual verilog simulator for teaching," in *Proceedings of the 8th Computer Science Education Research Conference*, ser. CSERC '19. New York, NY, USA: Association for Computing Machinery, 2019, p. 110–115. [Online]. Available: https://doi.org/10.1145/3375258.3375272

[8] Schocken, Shimon and Nisan, Noam, "From nand to tetris - building a modern computer from first principles," https://www.nand2tetris.org/, accessed July 12, 2020.

[9] Doulos, "Eda playground," https://www.edaplayground.com, accessed July 12, 2020.

[10] P. Falstad, "Falstad circuit simulator applet," https://www.falstad.com/circuit/, accessed July 12, 2020.

[11] MIT Scratch Team, "Scratch - create histories, games and animations. share with others around the world," https://scratch.mit.edu, accessed July 12, 2020.

[12] J. Mönig and B. Harvey, "Snap!" https://snap.berkeley.edu/, accessed July 12, 2020.

[13] J. Gomes, M. Pereira, A. Brito, and J. Ramos, "Um ambiente baseado em blocos para ensino de programação paralela com opencl," *International Journal of Computer Architecture Education*, vol. 5, no. 1, Dec. 2016. [Online]. Available: http://www2.sbc.org.br/ceacpad/ijcae/v5_n1_dec_2016/IJCAE_v5_n1_dez_2016_paper_7_vf.pdf

[14] "Nodejs," https://nodejs.org/en/, accessed July 12, 2020.

[15] The Blockly Team, "Blockly Guides - Introduction to Blockly," https://developers.google.com/blockly/guides/overview, accessed July 12, 2020.

[16] C. Wolf, "Yosys open synthesis suite," http://www.clifford.at/yosys/, accessed July 12, 2020.

[17] "The RISC-V Instruction Set Manual, Volume I: User-Level ISA, Document Version 2.2," editors Andrew Waterman and Krste Asanović, RISC-V Foundation, May 2017.

[18] C. Wolf, "Yosysjs," http://www.clifford.at/yosys/, accessed July 12, 2020.