Is it time to include High-Level Synthesis design in Digital System Education for Undergraduate Computer Engineers?

Isaac Nelson*
Miami Univiersity, OH, USA.
nelsonid@miamioh.edu

Ricardo Ferreira**, José Augusto Nacif**
Universidade Federal de Viçosa, Brazil.
ricardo, jnacif @ufv.br

Peter Jamieson*
Miami Univiersity, OH, USA.
jamiespa@miamioh.edu

2021 IEEE International Symposium on Circuits and Systems (ISCAS) | 978-1-7281-9201-7/20/$31.00 ©2021 IEEE | DOI: 10.1109/ISCAS51556.2021.9401774

***Abstract*—We ask the question, "should High-level Synthesis (HLS) design be part of an undergraduate computer engineering education while learning digital system design?".**

Current trends in industry include an increasing demand for engineers who can build FPGA systems. FPGAs, just like other chips, continue to improve in terms of complexity, speed, available resources, and new features. The design complexity for using an FPGA continues to grow and Hardware Description Languages (HDLs), though a step up in design efficiency compared to schematic design, is a low-level approach akin to assembly language for programmers, and HDLs limits the productivity of an engineer. HLS tools, such as Legup, Intel HLS, and Xilinx's Vivado attempt to provide designers with a higher-level design abstraction providing a means to describe their computation in high-level languages - such as C. As these tools become more mainstream in industry, when should education follow?

In this work, we explore how HLS tools might be used by an undergraduate by looking at exemplar designs, a simple RISC-V processor and a basic C loop, and implementing the design in both HDL and HLS. We then analyze the FPGA cost of each implementation. Next, we provide a philosophical discussion based on this experience on what the pros and cons of moving students to HLS design abstraction level are.

I. INTRODUCTION

FPGA design and the use of prototyping boards are common topics and pieces of hardware in higher-education engineering schools as a substitution for VLSI design. First, FPGAs allow complex designs to be implemented, tested, and designed in university undergraduate labs at a reasonable cost because FPGA companies produce their chips and CAD software for low-volume customizable applications - exactly the scenario in an undergraduate lab. FPGA prototyping boards and designing for them is a 'democratization' of Integrated Chip (IC) design similar to what microprocessors and the personal computer 'democratized' computing and programming. There is a demand for engineers who can create designs on FPGAs, and we, as educators and practitioners, need to constantly ask what aspects of design should be introduced and exposed in undergraduate education.

At this moment in time, we believe an important question is, should High-level Synthesis (HLS) design be part of an undergraduate computer engineering education with respect to what they learn in digital system design? HLS tools are a major efforts in the Electronic Design Automation (EDA) industry to allow software engineers to use programming languages to describe and implement hardware. In some circles this is known as "C-to-gates", and HLS is about using a high-level language that is a more traditional software programming language and automatically analyzing the program to create a low-level hardware implementation of the design on an FPGA or an ASIC. Presently, a large portion of EDA desgin is done via Hardware Description Languages (HDLs) such as Verilog HDL [1] and VHDL [2].

This paper examines this question from a pedagogical pros and cons perspective by leveraging the experience from an experiment we did in the past year. In particular, we had an undergraduate student implement a RISC-V processor [3] and a simple looped design in both HDL and C for an HLS flows to experience the two design methodologies. Additionally, because we target the same FPGA we can explore the one-off efficiency cost of moving to a higher-level design abstraction.

From this activity we present the basic ideas of the differing design approaches, an analytical comparison of the designs on the same target FPGA, and a pedagogical discussion of the pros and cons of the approach. Using this discussion, we then propose how HLS could be added to the undergraduate computer engineering curriculum in terms of what topics would be removed. In a constantly adapting profession, such as computer engineering, we believe exercises such as these are fundamental in experimentally questioning what aspects of innovation should be adopted into the curriculum.

II. BACKGROUND

Within computer engineering undergraduate education, a major focus is on building digital systems to perform computation for various applications and solutions. Within the suggested curriculum [4] released in coordination with both the IEEE and ACM in 2016, the "CEC-DIG" category (refering to digital systems) is one of the twelve major knowledge areas that a computer engineer should be exposed to, and digital system concepts and ideas are fundamental in all the other areas from embedded systems to information security.

The above document is an excellent reference for computer engineering undergraduate curricula, and for the CE-DIG 50 core hours, it is broken up into the following knowledge components as follows:

978-1-7281-9201-7/21/$31.00 ©2021 IEEE

- CE-DIG-1 - History [1 hr.]
- CE-DIG-2 - Tools and standards [2 hr.]
- CE-DIG-3 - Number systems and encodings [3 hr.]
- CE-DIG-4 - Boolean algebra applications [3 hr.]
- CE-DIG-5 - Basic logic circuits [6 hr.]
- CE-DIG-6 - Design of combinational circuits [8 hr.]
- CE-DIG-7 - Design of sequential circuits [8 hr.]
- CE-DIG-8 - Control and datapath design [9 hr.]
- CE-DIG-9 - Design with programmable logic [4 hr.]
- CE-DIG-10 - System design and constraints [5 hr.]

From a design language learning perspective, the document includes the prescribed use of HDLs including both Verilog HDL and VHDL, and there is a brief mention of schematic entry for low-level gate based systems.

We will make the assumption that all the above topics are of relevance to an undergraduate computer engineer. The question that we want to ask is when does the new design approach, HLS design, suggest that we spend less time on showing how to design these systems with HDL and schematics and shift to the designer efficient enabled with HLS?

Technological adoption into education is a constant debate among educators. The question of students using computers and calculators in the classroom for mathematics is one example of a these debates as to what is the student learning, and we, likely, had these same debates when an abacus, and paper and pencil were invented [5].

Over the last 20 to 25 years we have seen HDL languages and other ideas introduced into digital systems education. Takach and Moser [6] looked to improve digital systems design course, and Areibi [7] was one of the first authors to puclish work on using both HDL and programmable logic systems within these courses. Since then various approaches to improving students performance in this space have been attempted including a better understanding of HDL [8], computer architecture design with HDLs [9] [10], games [11] [12], and visualization techniques [13].

A. HLS for Digital System Design

In 2018, an article by Sakari *et al.* [14], asked the question is HLS ready in industrial design. This seems to make our question, from an educational perspective, a little presumptive as the technology seems to still be emerging in the quicker adopting industry domain. However, not only should academia stay abreast of what is happening in industry, but we should make similar endeavors with regards to these questions, particularly when there is a significant designer efficiency benefit.

Figure 1 shows a portion of Gajski-Kuhn Y-chart for VLSI design. The main difference between HDL languages and HLS is it shifts the abstraction more to behavioral and algorithmic descriptions and less structural components. HDLs allow for a range of structures to be described, and VHDL, in particular, has a very large range of what can be described and simulated - as this was the intention of the language. With an HLS language the designer moves away from describing the hardware implementation details and leaves this to the tools.

In the process of creating a hardware design from a higher level programming language a number of phases need to be executed including: Traditional front-end parsing, Allocation, Scheduling, Binding. These steps are needed since the target architecture is an open target [16] and the tool needs to determine what functional units will be created and who will use them throughout the execution of the computation.

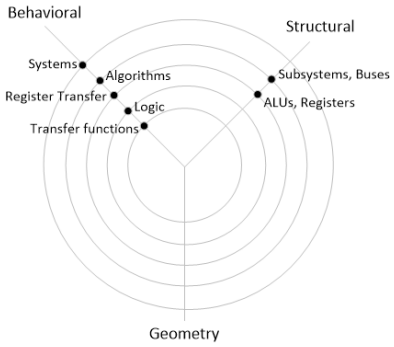

Fig. 1. Gajski and Kuhn Y-chart for VLSI design [15]

There are a vast number of HLS tools both in academia, for example, Legup [17] and Bambu [18], and in Industry Xilinx's Vivavo [19], Intel's HLS tool [20], and a commercialized version of Legup [17]. There are many more tools than these 4 and recent surveys by Nane *et al.* [21] and Huang *et al.* [22] includes more details about HLS and ongoing research questions.

This work uses both Bambu and Legup mainly because these tools can target any FPGAs, whereas the tools from Intel and Xilinx seem to be targeting specific FPGAs that are more expensive and less likely to be adopted in university labs.

III. HLS AND HDL IMPLEMENTATION OF SIMPLE DESIGNS

In this section, we provide a description of our small exemplar designs - RISC-V and a loop - and show how our two approaches perform on the FPGA via the two CAD flows (with HLS front-end(s) and one with straight HDL designs).

A. RISC-V processor architecture

To help us examine the question of including HLS as a portion of an undergraduate computer engineers digital system education, we will use a RISC-V processor architecture. We follow the suggestions provided in McGrew *et al.* work on how to design a RISC-V architecture as an undergraduate including tools to use to test and verify the system [10]. This provides us with a simple processor that we would expect most second year undergraduate students to be able to build in an HDL.

The first exercise is to build an HDL version of the processor. This process is a common design activity towards the end of a first year course on digital system design. Normally, students would be exposed to registers, control signals, and an Arithmetic Logic Unit (ALU) as bridging exercises between combinational circuit and sequential circuit design. From these exercises, it is not hard to extend the system a little further to create a simple processor. In our case, we use the a subset of

TABLE I
RISC-V RESULTS OF CAD FLOW ON INTEL FPGA FOR EACH FLOW

Design Entry	Logic Elements	Registers	Memory Bits	9x9 Multipliers	Fmax
Bambu	3472	1188	263168	6	102.32 MHz
Legup	2375	761	295936	6	75.71 MHz
VHDL	3747	1334	294912	6	98.61 MHz

the RISC-V architecture, implement the memory file on on-chip FPGA RAM, and piece together the ALU, register file, memory, and a controller.

We found it is a more challenging exercise to try and implement the same processor in C programming language such that it can be synthesized by both Legup's commercial version and Bambu for our target FPGA. To achieve this, we design the processor as a RISC-V emulator. This means that a controlling loop is used to fetch, decode, and execute instructions as fetched from a pre-initialized array (or memory). We, loosely, define this an the emulation of the RISC-V processor [23] and it is related to the wide hobby interest in emulation; however, it is interesting to ask are we emulating if we are writing in a form that the compiler then converts into hardware? As hardware designers creating an emulator in software is a strange experience.

We created implementations of the three systems, noting that the Bambu and Legup designs are very similar and are both written in C. All three can be accessed at BLIND.

To evaluate our circuits, we target the Cyclone IV EP4CE115F29C7 Intel FPGA that is available on Terasic's DE2-115 Development board. This is an older development board that is not that expensive (approximately 300 USD). We use Quartus 16.1 CAD flow to synthesize and map the design to this FPGA once it is in an HDL form suitable for this flow. For HLS we use the respective Legup or Bambu tool on Windows and Linux, respectively, and these tools similarly map the design using Quartus.

Table I shows the results of our designs for each of the flows. The first column describes each design entry format, columns 2 through 5 shows the resource consumption on the FPGA, and the last column shows the maximum frequency that the circuit can operate at.

In general, even though we can see differences in Table I for each of the flows, we do not see a clear benefit in terms of speed and area for the HDL version as might be expected. Instead, we see small differences that we believe may be eliminated if significant time was spent determining how each of the HLS flows perform their analysis and mapping of the software. For this type of design, the main result is that there is not significant differences between the three designs when we understand that a trade-off can be made for less area at a slower speed.

From a design efficiency perspective, we only have anecdotal results. It only took 2 weeks to create the C version of the RISC-V architecture compared to one month for the HDL version. There is a minor caveats with this result - the C version was created after the HDL version was built and the designer had additional experience both with the design and the FPGA flow while creating the HLS design version.

B. Looping Example

The previous exemplar design is not a great demonstration of the strength of HLS tools as the design, when written in C, represents and emulation of hardware as opposed to a programmers code description of an algorithm that is then transformed into a hardware implementation. In this section, we look at a simple loop construct and perform a similar experiment as above by comparing the design implemented for Legup versus an HDL implementation.

Algorithm 1 Simple Looping Benchmark

```
1: for (i = 0; i < N; i++) do
2:     c[i] = a[i] * b[i];
3:     sum += c[i];
4: end for
```

Algorithm 1 shows the C code we want to execute. It is very simple and was taken from the Legup 7.0 documentation as an example as there is no need to create a completely new example. Using the C version as the template, we implemented an HDL version of the same example with a finite state machine, but used the memory controller and wrapping logic created by the Legup flow.

Table II shows the results of each of the two instances. In both cases, the designs result in very similar FPGA resource usage and speed. The HDL one is slightly larger and slower. The real difference in this design problem is the design time where the HLS provides significant benefits compared to hand coding the finite state machines for the program. The C benchmark was a turnkey design in Legup and just took the time for the tool to synthesize. Converting the C code into Verilog took around thirty minutes of design and testing, and we note that this was for a very simple benchmark. We did not look into optimization for either flow.

IV. DISCUSSION

To start our discussion on whether to include HLS design as part of an undergraduates exposure to digital system designs, we will create a discussion of the pros and cons to the approach. In particular, we are interested in the questions of "designer efficiency", "system efficiency", "debugging capabilities", "accessibility for the learner", and "understanding the final hardware product". By looking at each of these areas, we begin to understand the trade-offs of this approach. Finally, we will conclude this section by showing that based on the discussion and the target industries and graduate schools for an institutes undergrads the answer to this question changes.

Designer Efficiency: the main reason to consider introducing HLS design at the undergraduate level is the potential

TABLE II
LOOP BENCHMARK RESULTS OF CAD FLOW ON INTEL FPGA FOR EACH FLOW

Design Entry	Logic Elements	Registers	Memory Bits	9x9 Multipliers	Fmax
Legup	64	47	0	0	291.29 MHz
Verilog	69	47	0	0	287.26 MHz

improvement on how quickly students can design their digital systems. In theory, designer efficiency is improved by using HLS assuming first, the user is familiar with C programming (or instead, will learn this competency as they learn HLS), and second, using the HLS tool and writing their design in C will in a reasonably efficient hardware implementation. Next, the user will need to familiarize themselves with the HLS tool, and more importantly, will need to learn how to create designs so that the tool can make efficient hardware implementations as they improve their own skill using the tool. Both of these needs will come at a time cost for the designer to learn, but once mastered should result in a significant designer efficiency improvement over HDL designs.

System Efficiency: already partially described earlier, the designer must learn to implement the design efficiently. Even by being knowledgeable with the HLS tool, we still expect some inefficiencies in the design. This is similar to the efficiency losses in using high-level languages that are compiled compared to assembly language due to automation and missed optimizations, the same will be true going from HDLs to HLS implementations. However, from our small exemplar applications, it is not evident that this will always be the case as a huge factor in this is the designers skills and experience, which an automated tool could hide.

Debugging Capabilities: when teaching digital systems and design with HDLs there is a disconnect between the realized hardware and what the designer is describing. Some teachers try to have students be able to recreate a schematic from Verilog or VHDL and there are a number of tools that can help in this visualization process (for example DigitalJS [13]). By using HLS tools we now have another level of indirection between the actual hardware and the design because the tool allocates/schedules/binds hardware components.

So understanding what is created from the design has a disconnect, and the next challenge is debugging functionality. There is some work on debugging HLS designs [24], but the level of indirection and the disconnect between simulation and synthesized hardware may be hard for a learner to understand. By using HLS tools, there is potential downside that designers will treat their resulting designs as more black-boxes than would typically be done with HDL designs, though using HDLs already introduces this challenge.

Accessibility for the learner: FPGAs and the software tools are relatively cheap and are available for undergraduates to access and work with giving them access to quality industrial CAD and parallel programming tools [25]. Both Xilinx and Intel's HLS tools, however, are targeted to very specific FPGA families that they release, which reduces accessibility making undergrads use simulation only. The Legup commercial tool allows most FPGAs to be targeted, but has a licensing costs as this company is attempting to create a viable business. Therefore, using HLS tools for undergraduate education does come with an additional financial cost.

Understanding the final hardware product: as we have already discussed, there is a trade-off in what the designer understands of the final hardware product produced from an HLS tool compared to their understanding of the hardware generated from HDL as their is a tighter one-to-one matching for the later. However, optimizations from a HDL design can obfuscate hardware designs even for HDL designs. As complexity of design increases, then there is a larger disconnect between product and design.

In general, teaching HLS design to undergrads means that we will, eventually, improve the designer efficiency at the cost of having less understanding of the final hardware artifact and incurring additional non-recurring engineering design costs. At this moment in time, we do not believe that HLS design should be taught to all undergraduates, but this question should be revisited in five years. There has been a shift in computer science and software engineering programs away from understanding software and its interaction with the CPU/computer. For some computer engineering programs we are also seeing a shift where embedded system design is more the goal as opposed to the lower level components [26]. For this shift learning more about how to use HLS to create quick hardware implementations of systems might make sense, and there exists research that is implementing HLS curriculum's [27].

Additionally, the question of what components of the existing curriculum could be shifted to spend on HLS design includes modifying "CE-DIG" 6 through 10. We expect that ideas of parallelism and acceleration will be further emphasized, focused on, and practiced with in computer engineering curriculum as a focus shifts to cloud computing. The choice of which tools including HLS, GPU coding, and cloud acceleration tools will need to be revisited and, likely, more time will need to be spent on this in the curriculum.

V. CONCLUSION

In this work, we examined the question of whether the improvements and the rise of HLS tools for FPGA and ASIC design means we should include this in computer engineers education. We investigated some exemplar low-level HLS versus HDL designs that might be done at a 2nd year level, and we observed and commented on the exercise and the state of these tools. Our conclusion is that HLS tools are on the cusp of being part of the undergraduate education curriculum, but as of now the trade-offs are still in favor of teaching HDL digital system design.

ACKNOWLEDGMENTS

This work was supported by CNPq, grant #436290/2018-9.

REFERENCES

[1] *Verilog Hardware Description Reference*, Open Verilog International, March 1993.

[2] *IEEE Standard VHDL Language Reference Manual*, IEEE, 1987.

[3] A. Waterman, Y. Lee, D. A. Patterson, and K. Asanovic, "The risc-v instruction set manual, volume i: Base user-level isa," *EECS Department, UC Berkeley, Tech. Rep. UCB/EECS-2011-62*, vol. 116, 2011.

[4] J. Impagliazzo*et.al*, "Curriculum guidelines for undergraduate degree programs in computer engineering," 2016. [Online]. Available: https://www.acm.org/binaries/content/assets/education/ce2016-final-report.pdf

[5] D. Klein, "A brief history of american k-12 mathematics education in the 20th century," *Mathematical cognition*, pp. 175–225, 2003.

[6] M. Takach and A. Moser, "Improving an introductory course on digital logic," in *Frontiers in Education Conference, 1995. Proceedings., 1995*, vol. 2, 1-4 1995, pp. 4b6.1 –4b6.2.

[7] S. Areibi, "A first course in digital design using vhdl and programmable logic," in *Frontiers in Education Conference, 2001. 31st Annual*, vol. 1, 2001, pp. TIC –19–23. [Online]. Available: http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.16.3048

[8] R. Rodriguez-Ponce and J. Rodriguez-Resendiz, "Integrating vhdl into an undergraduate digital design course," in *Teaching, Assessment and Learning for Engineering , 2013 IEEE International Conference on*. IEEE, 2013, pp. 172–177.

[9] N. Calazans and F. Moraes, "Integrating the teaching of computer organization and architecture with digital hardware design early in undergraduate courses," *Education, IEEE Transactions on*, vol. 44, no. 2, pp. 109 –119, may 2001.

[10] S. E. McGrew, Tyler and P. Jamieson, "Framework and tools for undergraduates designing risc-v processors on an fpga in computer architecture education," in *2019 International Conference on Computational Science and Computational Intelligence (CSCI)*. IEEE, 2019, pp. 778–781.

[11] P. Jamieson, L. Grace, B. Zhang, and N. Mizuno, "verilogtown-improving students learning hardware description language design-verilog-with a video game," in *2015 ASEE Annual Conference and Exposition*, 2017.

[12] L. Grace, P. Jamieson, N. Mizuno, and B. Zhang, "Verilogtown: Cars, crashes and hardware design," in *Proceedings of the 12th International Conference on Advances in Computer Entertainment Technology*, ser. ACE '15, 2015, pp. 39:1–39:3. [Online]. Available: http://doi.acm.org/10.1145/2832932.2832936

[13] M. Materzok, "Digitaljs: a visual verilog simulator for teaching," in *Proceedings of the 8th Computer Science Education Research Conference*, 2019, pp. 110–115.

[14] S. Lahti, P. Sjövall, J. Vanne, and T. D. Hämäläinen, "Are we there yet? a study on the state of high-level synthesis," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 5, pp. 898–911, 2018.

[15] D. Gajski, "Silicon compilers," 1987.

[16] G. D. Micheli, *Synthesis and optimization of digital circuits*. McGraw-Hill Higher Education, 1994.

[17] A. Canis, J. Choi, M. Aldham, V. Zhang, A. Kammoona, J. H. Anderson, S. Brown, and T. Czajkowski, "Legup: high-level synthesis for fpga-based processor/accelerator systems," in *Proceedings of the 19th ACM/SIGDA international symposium on Field programmable gate arrays*, 2011, pp. 33–36.

[18] C. Pilato and F. Ferrandi, "Bambu: A modular framework for the high level synthesis of memory-intensive applications," in *Int Conf on Field programmable Logic and Applications*, 2013, pp. 1–4.

[19] T. Feist, "Vivado design suite," *White Paper*, vol. 5, p. 30, 2012.

[20] M. Sussmann and T. Hill, "Intel HLS compiler: Fast design coding and hardware," in *White paper*, 2017.

[21] R. Nane, V.-M. Sima, C. Pilato, J. Choi, B. Fort, A. Canis, Y. T. Chen, H. Hsiao, S. Brown, F. Ferrandi *et al.*, "A survey and evaluation of fpga high-level synthesis tools," *IEEE Trans. on CAD of Integrated Circuits and Systems*, vol. 35, no. 10, pp. 1591–1604, 2015.

[22] L. Huang, D.-L. Li, K.-P. Wang, T. Gao, and A. Tavares, "A survey on performance optimization of high-level synthesis tools," *Journal of Computer Science and Technology*, vol. 35, pp. 697–720, 2020.

[23] V. Moya, "Study of the techniques for emulation programming," *Proyecto fin de carrera. Universidad Politécnica de Cataluña. España*, 2001.

[24] J. Goeders and S. J. Wilton, "Effective fpga debug for high-level synthesis generated circuits," in *2014 24th International Conference on Field Programmable Logic and Applications (FPL)*. IEEE, 2014, pp. 1–8.

[25] P. Jamieson, M. Herbordt, and M. Kinsy, "A case study: Undergraduate self-learning in hpc including openmp, mpi, opencl, and fpgas," in *Annual Conf. on Computational Science & Computational Intelligence (CSCI'19)*, 2019.

[26] P. Jamieson and J. Herdtner, "More missing the boat - arduino, raspberry pi, and small prototyping boards and engineering education needs them," in *Frontiers in Education Conference , 2015. 32614 2015. IEEE*. IEEE, 2015, pp. 1–6.

[27] R. C. Panicker, A. Kumar, and D. John, "Introducing fpga-based machine learning on the edge to undergraduate students," in *2020 IEEE Frontiers in Education Conference (FIE)*. IEEE, 2020, pp. 1–5.