Paper ID #17853

verilogTown - Improving Students Learning Hardware Description Language Design - Verilog - with a Video Game

Dr. Peter Jamieson, Miami University

Dr. Jamieson is an associate professor in the Electrical and Computer Engineering department at Miami University. His research focuses on Education, Games, and FPGAs.

verilogTown - Improving Students Learning Hardware Description Language Design - Verilog - with a Video Game

Abstract

In this work, we present our game, verilogTown, as an aid to students learning Verilog. The reason for such a game comes from our experiences teaching digital system design where we observed a challenge for second year students learning to design with the Verilog hardware description language (HDL). In this work, we speculate why it is hard to learn an HDL, claiming that like learning all languages, the students do not play/use the language enough to develop an understanding of them (including Verilog). A student's typical process of learning Verilog includes class examples and assignments, labs, and a project, but like learning more traditional programming languages, until a learner spends significant time using a language to build something, these experiences only result in a basic understanding. verilogTown was created to provide students with a medium to play with Verilog in an engaging and safe environment. The hope is that instead of simply completing a working design for a class, students will be challenged by the game puzzles and will spend significantly more time with Verilog design. Our comparison on exam performance in 2014 and 2015 without and with verilogTown suggests that our game impacts student Verilog understanding.

Introduction

Over six years of teaching second year students digital system design, the major challenge for these students is how to describe hardware in a hardware description language (HDL). In our curriculum, the Verilog HDL is used to describe synthesizable designs mapped to an FPGA (noting that using VHDL is very similar and challenging). Students, at this stage, have experience with imperative programming, calculus, and physics, each of which are challenging areas, and each pose problems that are, arguably, at the level of simple digital system design. However, in a single semester course going from logic to digital arithmetic and control there a large amount of material to gain expertise in. Additionally, we challenge students to understand the design of common digital structures at both the schematic level and HDL level. The result is some understanding of using an HDL to design a complex system, but we would like students to have better understanding of Verilog, and for the experience to be less jarring in their learning process. Some teachers might argue that either there is too much material to learn in a short time period or that the material is challenging and hard to learn and simply is a difficult learning experience. We, however, are experimenting to see if other methods might make this learning process better.

In particular, we have designed a video game that includes Verilog HDL as the design language

used to control the world. This idea was inspired by other video games such as CodeHero (`primerlabs.com/codehero`) which mixes first person shooter games with a gun that can be programmed in Javascript to manipulate the world, and vimAdventure (`vim-adventures.com/`) which mixes an adventure game with learning the vim editor controls. verilogTown, is a puzzle game where cars are preset to travel a city of roads at specific entry and exit points at a given time. The cars are intelligent enough to pay attention to traffic signals (go, go left, go right, go forward, or stop), to not crash into a car going the same direction in front of them, and to calculate the shortest path to their, respective, exit. The puzzle is to write Verilog designs controlling each traffic light such that the cars safely traverse to their destination (no crashes or routing to a wrong destination) and all cars get through the city in a minimum amount of time.

The contributions for this paper are the creation of such a game and the improvement students showed in a small sample study of classes taught in 2014 and 2015. Additionally, this work is developed as open source software written in Java, and we hope that this might be useful to other researchers including our games simple Verilog compiler and simulator.

In this paper, we describe our video game, the Verilog simulation environment, and a lab/assignment we have designed for instructors to modify/use for their classes. In the background 2, we briefly examine the state of video games in education, and provide research results as well as our own speculation of why language design is a challenge to learn. We then describe, in section 3 the game and resources we include, Verilog simulation, and a lab we have created for others. In section 4 we describe how we evaluated the impact of our game, and in section 5 we discuss the current anecdotal user experiences with the game, and finally, we conclude the paper and describe future work.

Background

In the background we will focus on three main focal points: games for learning, digital design education, and learning design languages. In particular, our goal is to expose researchers to these broad fields and and speculate why languages are hard to learn. We argue that an HDL language is even harder to learn than programming languages for an undergraduate student.

Games for Learning

Games, which includes both in person tabletop games (board games, card games, party games) and video games, are activities with three distinctive features[1] (one of many definitions):

1. A game has a set of rules
2. A game provide feedback to the players
3. A game is voluntary

We claim that higher education includes the same set of defining features, and early education (K-12) includes only features 1 and 2. And yet, the striking difference between games and

education is that most people become highly engaged with games, but not so much with most of their education. In particular, video games have the ability to highly engage a large portion of the population, and people have speculated that this is the case because the game design (through the machines control) can present a user with a challenge that is just above their skill level and evoke a flow state. The flow state is "In such a state a person feels fully alive and in control, because he or she can direct the flow of reciprocal information that unites person and environment in an interactive system"[2].

Another important aspect of games is the idea of failure[3]. In education, it is very hard to fail, and in many cases within the education system, failure is a disaster, especially in the high stakes assessment we typically employ[4]. Because of this, failure has become something to be avoided or mitigated with the exception of people playing games; for example, for those of us who played Mario (`mario.nintendo.com/`) , Tetris (`tetris.com/`), Pacman (`pacman.com/`), or Angry Birds (`www.angrybirds.com/`), failure is an expectation in learning to succeed. Video games present failure as a low stakes assessment or feedback point that shows you have not learned a solution or skill to complete the task. Note, not many people ride a bike, solve a differential equation, or read a sentence perfectly the first time they make an attempt at these various skills and problems.

Because of the qualities and relationship between games and learning, many people have tested games as learning environments to improve student performance. It is still undetermined if creating games for learning is a worthwhile effort[5]. Below, we list four recent attempts by researchers in technical subjects just to show the range of ideas of using games for learning:

- Sol and Stephens created a game to teach statistics[6]
- Navarro *et. al.* built a game to help students learn radio communication[7]
- Lyon *et. al.* created, Little Newton, to teach physics[8]
- Cheng *et. al.* build an educational game for learning immunology[9]

This is a very small sample of games that focus on learning in technical fields.

In general, the field of games used for other purposes than just entertainment continues to grow, and this paper can not provide a full introduction to the subject. We direct the interested reader to Susi *et. al.*[10] as a place to start learning more about this field. For more directed questions of games and entertainment, Mitchell and Savill-Smith's work[11] provides a survey of early 2000 game research, and surveys of the impact on learning with video games All *et. al.*[12] and Boyle *et. al.*[13] provide a more present view of the lay of the land. Finally, our work provides a lense[5] for educators on what to look at as the educational benefits of a game before pursuing the challenging developmental efforts needed to create such digital games.

Research in Digital Design Education

Researchers and educators have questioned how to improve the learning outcomes and successes of students learning digital design, which itself is a massive industrial and academic field. Learning and education fit into the research domain of the social sciences, which means that the

complexity of humans and our experiments will rarely show clear measurable results and linear relationships that can be explained by a researchers intervention. Still, this field exposes ideas on how we may improve the learning experience.

Digital design taught to undergraduates tends to include an understanding of transistors as digital switches, transistors organized into logic gates, and structuring logic gates into more complex functions such as arithmetic units, memories, and finite state machines. There are a number of topics in this process including optimization for speed, area, power, and ease of design. The last of these leads us to exploring schematic design versus HDL design (and possibly high-level synthesis techniques). These topics can be spread out over three to four courses, but can also have been taught in one to two courses depending on a universities resources and curriculum. Finally, most courses in this domain are accompanied with practical lab design, normally, through simulation in software such as Multisim[14], wiring designs on bread boards, or, more recently, on programmable prototyping boards (with FPGAs) provided by companies such as Altera[15] and Xilinx[16] and their respective university programs.

In terms of research on improving digital system education, a number of papers have been presented. Takach and Moser[17] were one of the early contributors for using simulation and PLDs (Programmable Logic Devices) in their teaching of digital systems. PLDs, in the teaching domain, have been replaced with FPGA prototyping boards and Zhu *et. al.*[18] and Wang[19] describe their experiences with these as well as continuing work[20]. Ariebi discussed similar innovation in these courses as the first to discuss HDLs integrated in a digital design course[21], which was followed by others such as[22]. At, roughly, the same time Calazans and Moraes discussed how hardware and computer architecture can be integrated[23], and their approach was to implement the complex design of a processor in a simulation tool and then run an algorithm such as bubble sort on a student designed processor. Amaral *et. al.* looked at creating a course for computer science students in digital design and used a series of prepared labs (including prepared hardware modules) that allowed students to progress to complex systems in a single term[24].

Learning Design Languages

Design languages include both programming languages and HDLs, where we argue that the HDLs have more similarity to parallel programming than most early taught programming languages, which are sequential and imperative. Teaching programming languages is a common topic at education conferences and the research literature is significantly large. Mayer[25] provides a recent book on the topic and the many paths people have taken in the field of learning programming languages. Topics in this area greatly vary from programming in K-12 education[26] to specifics of object-oriented programming[27].

HDLs as a taught skill has a much more limited research base. Duckworth[28] presents the experiences he has with incorporating VHDL and FPGAs as design tool and target for students. In particular, he notes that students have particular problems with getting the system to work, using simulation as a design step, and understanding that an HDL is not a programming language. Ebeling and French attempt to help students by creating an HDL, Abstract Verilog, with "well-defined, clean parallel execution semantics"[29]. Vemuru *et. al.* propose a spiral model of

teaching where HDL pedagogy is intertwined with topics and is slowly built up with complexity over time[30]. As pointed out by Kumar *et. al.*[31] the HDL and tool flow is an industrial tool used by professional engineers that should be included in undergraduate education.

From our own experience integrating Verilog into a second year digital system course, students have a tough time with Verilog due to its C or Java like syntax (which they model in their mind as sequential executing language). Our approach is to have students implement designs as either schematics or HDL and then have the student interpret their design into the other format so that they begin to understand how the two forms are the same, but the advantage of using design languages is it allows for more rapid design and re-usability. Still, the topic consistently is reported as one of the hardest parts of the course, and as above, we suspect the parallel nature of the computation is a major challenge for students.

verilogTown

Here, we describe our game verilogTown[32] including resources we provide, the design of the Verilog simulator, and a lab we have created that others can use in their courses.

What is the game, editor, and level editor

Our motivation to create a video game for Verilog was to help students by spending more time playing with the HDL instead of simply using it to complete a lab. Our hypothesis is that spending more time playing and using a language will help students improve their understanding and skill with the language.

The goal in a verilogTown level is to get the vehicles through the town without crashing and as fast as possible. Figure 1 shows a sample level map where cars are driving around and responding to traffic control at intersections. To achieve this goal, players write the Verilog that controls the traffic signals at every four-way and three-way intersection. This means that the student/player needs to write a Verilog module for each of the six intersections seen in Figure 1.

The possible traffic controls (output of each of the six modules) includes stop (st), go , turn left (tl), turn right (tr), and go straight (gs) where the “go” signal allows the vehicle to follow its current shortest path to destination, and the other signals, potentially, force a car to follow another path. This allows the player to either specify where a car goes as per a defined solution. Once a student/player has written their Verilog modules, the level is started and car crashes can occur at intersections when two (or more) cars are turning; cars will not run into a car in front of them going in the same direction. Cars also calculate and re-calculate the shortest path to their desired exit point.

As stated earlier, the goal of the game is to get all of the cars through the city without any crashes and in the minimum amount of time. Levels are designed as a puzzle so that the Verilog modules written for each intersection can be designed to change paths or timing of the cars to improve the overall traffic flow. A player can edit the HDL that controls the traffic signals each intersection by

Figure 1: A verilogTown level with cars driving around with the traffic signals

clicking it and opening up an editor. From the perspective of the Verilog HDL each intersection has the following inputs and outputs (*with their corresponding Verilog*):

- a system clock - this clock signal (*input clk*) is a 25Hz clock.
- a system reset - this reset signal (*input rst*) is a system-wide reset low signal that is mainly used by the simulator.
- input intersection sensors - there are 8 input sensors (*input [7:0] sensor_light*) in a 4-way intersection. Four sensors are positioned at the car entry points to the traffic signal, and four are in the intersection. Each of these sensors is a binary signal that shows if a car is currently on that sensor.
- user placed sensors - depending on the level (maximum of 30 - *input [29:0] general_sensors*) , there are a set number of user placed sensors (in the car lanes) that every traffic signal on the map can read and use for predictive algorithms of incoming traffic. Figure 1 shows 3 of these sensors labeled 4, 5, and 6
- output signals - each car going in a direction (**N**orth, **E**ast, **S**outh, **W**est) has a corresponding traffic signal that can send one of the five signals (go, st, tl, tr, gf) - *output [2:0]outN; output [2:0]outE; output [2:0]outS; output [2:0]outW;*
- debug port - a debug port (*output [29:0] debug_port*) in an output which users can send out information during simulation about internal signals.

The player needs to solve/improve their control for a respective level by modifying the supplied combinational and sequential Verilog templates to get all cars through the map. This is achieved

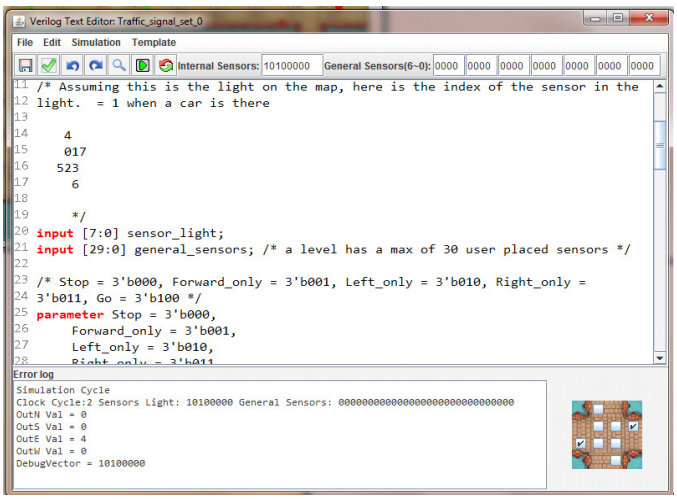

Figure 2: The editor where Verilog can be edited for a traffic signal

by clicking on a traffic intersection and modifying the Verilog design associated with it. Figure 2 shows the editor that in addition to being a medium for writing and compiling the Verilog can be used to simulate the behavior of the control Verilog depending on cars being at the traffic light sensors and the general sensors.

Once the player has created their design solution, they simply start the game simulation to see how they perform. This simulation can be paused and restarted to observe car behavior to come up with new and better traffic control solutions.

Game Simulator

The development language used to create verilogTown and its resources is Java, and even though we have experience creating open-source Verilog compilers[33][34], the simulation environment and compiler are completely original creations. In particular, instead of using Bison[35] and Flex[36] tools to create a parser and then creating an Abstract Syntax Tree, we instead used ANTLR 4.0 as the parser/data-structure[37], which is an excellent tool, but takes a significantly different approach to language parsing compared to our previous experiences. Interested users should read the previously cited book to understand ANTLR 4.

Since the Verilog for the game and the editor is only used for simulation, the Verilog simulation is much simpler than creating a functional netlist. Basically, each Verilog file (representing the control for each traffic light) is compiled at simulation time during which we create a symbol table, which is used to store the state of the Verilog (including output and input vectors). When a cycle is simulated, the input vectors (which are the sensors) are entered into the simulator and the syntax tree is walked through to update the respective values in the symbol table, which eventually propagates to the output vectors. The output vectors can then be read by either the editor or game simulation to update their respective states and that of the game world. Since each traffic signal is independent from one another we can simulate each traffic signal independently, and from a performance perspective compilation is done before the game simulation is started

(cars start entering and moving on the map), which means during game simulation the syntax trees only need to be traversed to update values. For sequential circuits the simulation is run 50 times per second (one simulation for clock updates and one simulation for signal propagation) and for the levels we have designed the simulation performance is real-time from the players perspective on a typical PC.

We do not support all the Verilog syntax, and the reason for this is the game has been designed for beginner users. Our approach to teaching Verilog is to keep the syntax as simple synthesizable syntax, and we include that syntax capability in the game. Our website lists the supported Verilog and key missing structures (`http://www.users.miamioh.edu/jamiespa/verilogTown/support.html`).

A Lab or Assignment with the game

Our goal is to have students play with the Verilog language more than they would in a standard course where they normally use the language to create a few designs for each lab. However, students are not likely to simply pick up the game and start playing it even though they have a hard time with learning the language. To help introduce students to the game, we have created a lab/assignment that has students create solutions for three of the game levels. This lab is available in word format at:
`http://www.users.miamioh.edu/jamiespa/verilogTown/lab.html`.

We have provided this lab on our website so that instructors can include this in their labs or as a separate assignment. We recommend that this be used in one of two ways. First, students can have their first introduction to Verilog HDL by playing the game and referencing our website. This is a self-learning approach that will require students to spend significant time to understand both Verilog and our game, but because of the provided templates this process is not that difficult and will provide students with templates that they can use to start to design Verilog with. The Verilog can then be used in class to illustrate the basics of the language, but we suggest that instructors spend some time on showing how Verilog is converted to schematics to make the connection between description and synthesizable logic (as Part 2 in our lab tests). The second approach is to include this activity after the students have some familiarity with Verilog as practice with changing designs for a desired response. In our study below, students used the second approach where they played the game after being exposed to Verilog.

Also, note that part 2 of this lab is about understanding how Verilog is translated into schematics. This can be removed if the students are not yet at a stage where they have this understanding. The sequential template is a finite state machine that may or may not be understood by the students, and this aspect can also be removed or dealt with appropriately.

Results

Our hypothesis is that by including verilogTown as part of the learning experience our students will improve their design ability using Verilog. To test this hypothesis, we taught a course with

and without verilogTown. In 2014, we taught our digital design course to 60 students without verilogTown, and in 2015, we taught the same course to 34 students with one lab replaced by the one supplied above that includes the use of verilogTown (class size difference was due to a new faculty member). Otherwise, both courses were identical in what was taught and how the courses were assessed.

Our assessment of Verilog performance was based on the second exam, which includes a question on designing a finite state machine (FSM) in Verilog, and the project. The second exam has a question worth 60% of the exam grade where the other 40% is for designing an FSM in schematics. Overall exam 2 is 10% of a student's grade. The project is worth 20% of the students grade, and there is a high emphasis placed on getting something working, but the design must be more difficult than the final lab. The reason labs are not used is they are graded based on mastery where if a student completes the lab with a successful demonstration, they get full credit for the lab.

In terms of a fair comparison between these groups, our experiment has a number of flaws. For example, the two groups are of significant different sizes due to the teaching arrangement at our institution. Note, however, the class average in 2014 was 73.2% and in 2015 was 72.9%, which suggests the class performance as a whole is similar. Additionally, we did not perform any standardized statistical tests as our results will suggest some benefit to our approach, but we believe a larger and more properly prepared experiment should be done in this domain.

Table 1: Results for exam 2

Year	Number of students	Exam 2 average	Exam 2 stdev	Project average	Project stdev
2014	60	56.8	21.7	74.2	18.6
2015	31	66.2	20.0	76	19.9

Table 1 shows the results of the two classes where the arithmetic average and standard deviation out of 100 are reported for each year on exam 2 and the major project. Typically, most of the points are lost on the Verilog question in this exam, and in year 2015 that included the use of verilogTown the percentage on the exam went up by about 9%. In our experience, this type of improvement is significant.

The project comparison in Table 1 shows no major difference in ability between the two years. This does not surprise us since students have a significant amount of time to get their projects working, which is a major portion of the grade where only about 30% of the project grade is to complexity of design. These projects, however, are significant achievements for the students.

We did not survey the students on how their experience was with the game since there is nothing to compare this against. Anecdotally, the students in 2015 liked the inclusion of the game where a discussion with one student found out that the sequential templates included in verilogTown were key for him to understand sequential logic and FSMs. There were also a few students that we had to tell to stop playing the game as there were more important tasks at hand.

Conclusion

In this work, we presented verilogTown, which is a game that uses Verilog HDL as the language to control traffic signals in a city. This game is an open source game that tries to give students a platform to play around with simple Verilog to help them get a better understanding of the design language. We included a number of resources with the game that allow students and educators to expand the game in any direction they feel is beneficial for their needs. For example, we include a level editor so that learners and educators can create additional levels beyond the 12 levels that come with the base game. This might include understanding deadlock and live-lock, and it is even conceivable that the game could be used to implement a communication protocol, or even a processor.

The question remains, is verilogTown an effective tool that meets our proposed goals. Based on a comparison of exam 2 results in 2014 and 2015 with no other significant course change other than the inclusion of verilogTown, we would conclude yes. The 2015 group performed on average 9% better than the previous year, and exam 2 is our most significant Verilog design assessment. When comparing grades for projects, there is no significant grade difference between the two groups, and we explained that this is because of the emphasis on working designs and the significant amount of time given to the project.

The future of this work is with one particular question: Can video games be used to teach an entire intro to digital design course and is it as effective? verilogTown is a step in this direction in proving that there is a benefit to games and teaching HDLs.

The website `www.users.miamioh.edu/jamiespa/verilogTown/` provides many additional details about the game and has links that will guide researchers to the source code, downloadable files, and other resources for the game and learning Verilog.

References

[1] J. McGonigal. *Reality Is Broken: Why Games Make Us Better and How They Can Change the World*. Penguin Press, 2011.

[2] Mihaly Csikszentmihalyi. *Flow*. Springer, 2014.

[3] Tim Harford. *Adapt: Why success always starts with failure*. Macmillan, 2011.

[4] Alfie Kohn. *Punished by rewards: The trouble with gold stars, incentive plans, A's, praise, and other bribes*. Houghton Mifflin Harcourt, 1999.

[5] P. Jamieson and L. Grace. A framework to help analyze if creating a game to teach a learning objective is worth the work. In *2016 IEEE Frontiers in Education Conference (FIE)*, pages 1–7, 2016.

[6] Sol Nte and Richard Stephens. Videogame aesthetics and e-learning: A retro-looking computer game to explain the normal distribution in statistics teaching. In *The 2nd European Conference on Games Based Learning*, pages 341–348, 2008.

[7] A. Navarro, J.V. Pradilla, S. Londono, P. Madrinan, I. Abadia, J.C. Alonso, and A.X. Gonzalez. Test: Serious games for radio communications learning. In *Frontiers in Education Conference, 2013 IEEE*, pages 517–522, Oct 2013.

[8] Natalie Lyon, Josep Valls, Caroline Guevara, Ning Shao, Junyu Zhu, and Jichen Zhu. Little newton: an educational physics game. In *Proceedings of the first ACM SIGCHI annual symposium on Computer-human interaction in play*, pages 351–354. ACM, 2014.

[9] Meng-Tzu Cheng, TzuFen Su, Wei-Yu Huang, and Jhih-Hao Chen. An educational game for learning human immunology: What do students learn and how do they perceive? *British Journal of Educational Technology*, 45 (5):820–833, 2014.

[10] Tarja Susi, Mikael Johannesson, and Per Backlund. Serious games: An overview. 2007.

[11] Alice Mitchell and Carol Savill-Smith. The use of computer and video games for learning: A review of the literature. 2004.

[12] Anissa All, Elena Patricia Nuñez Castellar, and Jan Van Looy. A systematic literature review of methodology used to measure effectiveness in digital game-based learning. In *European Conference on Games Based Learning*, page 607. Academic Conferences International Limited, 2013.

[13] Elizabeth A Boyle, Thomas Hainey, Thomas M Connolly, Grant Gray, Jeffrey Earp, Michela Ott, Theodore Lim, Manuel Ninaus, Claudia Ribeiro, and João Pereira. An update to the systematic literature review of empirical evidence of the impacts and outcomes of computer games and serious games. *Computers & Education*, 94:178–192, 2016.

[14] K Noga and M Radwanski. The teaching of digital techniques with multisim 2001. In *9th Baltic Region Seminar on Engineering Education, Monash Engineering Educations Series, Gdynia Maritime University*, pages 17–20, 2005.

[15] AlteraU. Altera University Program at https://www.altera.com/support/training/university/overview.html. 2010.

[16] XilinxU. Xilinx University Program at http://www.xilinx.com/support/university.html. 2010.

[17] M.D. Takach and A.T. Moser. Improving an introductory course on digital logic. In *Frontiers in Education Conference, 1995. Proceedings., 1995*, volume 2, pages 4b6.1 –4b6.2, 1-4 1995.

[18] Yi Zhu, T. Weng, and Chung-Kuan Cheng. Enhancing learning effectiveness in digital design courses through the use of programmable logic boards. *Education, IEEE Transactions on*, 52(1):151 –156, feb. 2009. URL `http://ieeexplore.ieee.org/xpls/abs_all.jsp?arnumber=4696063&tag=1`.

[19] Guoping Wang. Bridging the gap between textbook and real applications: A teaching methodology in digital electronics education. *Computer Applications in Engineering Education*, 19(2):268–279, 2011.

[20] P. Jamieson. Early project based learning improvements via a star trek engineering room game framework, and competition. In *Frontiers in Education Conference (FIE), 2011*, pages S4H–1 –S4H–5, oct. 2011. URL `http://www.users.muohio.edu/jamiespa/html_papers/fie_11.pdf`.

[21] S. Areibi. A first course in digital design using vhdl and programmable logic. In *Frontiers in Education Conference, 2001. 31st Annual*, volume 1, pages TIC –19–23, 2001. URL `http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.16.3048`.

[22] Rafael Rodriguez-Ponce and Juvenal Rodriguez-Resendiz. Integrating vhdl into an undergraduate digital design course. In *Teaching, Assessment and Learning for Engineering (TALE), 2013 IEEE International Conference on*, pages 172–177. IEEE, 2013.

[23] N.L.V. Calazans and F.G. Moraes. Integrating the teaching of computer organization and architecture with digital hardware design early in undergraduate courses. *Education, IEEE Transactions on*, 44(2):109 –119, may 2001.

[24] J.N. Amaral, P. Berube, and P. Mehta. Teaching digital design to computing science students in a single academic term. *Education, IEEE Transactions on*, 48(1):127 – 132, feb. 2005.

[25] Richard E Mayer. *Teaching and learning computer programming: Multiple research perspectives*. Routledge, 2013.

[26] Peter Hubwieser. Computer science education in secondary schools – the introduction of a new compulsory subject. *Trans. Comput. Educ.*, 12(4):16:1–16:41, November 2012. URL http://doi.acm.org/10.1145/2382564.2382568.

[27] John Lewis. Myths about object-orientation and its pedagogy. In *Proceedings of the Thirty-first SIGCSE Technical Symposium on Computer Science Education*, pages 245–249, 2000. URL http://doi.acm.org/10.1145/330908.331863.

[28] R.J. Duckworth. Embedded system design with fpga using hdl (lessons learned and pitfalls to be avoided). In *Microelectronic Systems Education, 2005. (MSE '05). Proceedings. 2005 IEEE International Conference on*, pages 35–36, June 2005.

[29] Carl Ebeling and Brian French. Abstract verilog: A hardware description language for novice students. In *Microelectronics Systems Education, IEEE International Conference on/Multimedia Software Engineering, International Symposium on*, pages 105–106, 2007. URL http://www.computer.org/portal/web/csdl/doi/10.1109/MSE.2007.16.

[30] Srinivasa Vemuru, Sami Khorbotly, and Firas Hassan. A spiral learning approach to hardware description languages. In *Circuits and Systems (ISCAS), 2013 IEEE International Symposium on*, pages 2759–2762. IEEE, 2013.

[31] Akash Kumar, Rajesh C Panicker, and Ashraf Kassim. Enhancing vhdl learning through a light-weight integrated environment for development and automated checking. In *Teaching, Assessment and Learning for Engineering (TALE), 2013 IEEE International Conference on*, pages 570–575. IEEE, 2013.

[32] Lindsay Grace, Peter Jamieson, Naoki Mizuno, and Boyu Zhang. Verilogtown: Cars, crashes and hardware design. In *Proceedings of the 12th International Conference on Advances in Computer Entertainment Technology*, ACE '15, pages 39:1–39:3, 2015. URL http://doi.acm.org/10.1145/2832932.2832936.

[33] Peter Jamieson and Jonathan Rose. A Verilog RTL Synthesis Tool for Heterogeneous FPGAs. In *Field-Programmable Logic and Applications*, pages 305–310, 2005. URL http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.77.3289.

[34] Peter Jamieson, Kenneth B. Kent, Farnaz Gharibian, and Lesley Shannon. Odin II - An Open-source Verilog HDL Synthesis tool for CAD Research. In *Proceedings of the IEEE Symposium on Field-Programmable Custom Computing Machines*, pages 149–156, 2010. URL http://www.computer.org/portal/web/csdl/doi/10.1109/FCCM.2010.31.

[35] GNU. Bison - GNU parser generator, 2009. URL \url{http://www.gnu.org/software/bison/}.

[36] Vern Paxson. The Lex & Yacc Page, 2009. URL \url{http://dinosaur.compilertools.net/flex/index.html}.

[37] Terence Parr. The definitive antlr 4 reference. 2013.