A Literate Programming Approach for Hardware Description Language Instruction

Dr. J.W. Bruce, Tennessee Technological University

J.W. Bruce is with the Department of Electrical & Computer Engineering at Tennessee Technological University in Cookeville, Tennessee USA

Dr. Bryan A. Jones, Mississippi State University

Bryan A. Jones received the B.S.E.E. and M.S. degrees in electrical engineering from Rice University, Houston, TX, in 1995 and 2002, respectively, and the Ph.D. degree in electrical engineering from Clemson University, Clemson, SC, in 2005. He is currently an Associate Professor at Mississippi State University, Mississippi State, MS.

From 1996 to 2000, he was a Hardware Design Engineer with Compaq, where he specialized in board layout for high-availability redundant array of independent disks (RAID) controllers. His research interests include engineering education, robotics, and literate programming.

Dr. Mahnas Jean Mohammadi-Aragh, Mississippi State University

Dr. Jean Mohammadi-Aragh is an assistant professor in the Department of Electrical and Computer Engineering at Mississippi State University. Dr. Mohammadi-Aragh investigates the use of digital systems to measure and support engineering education, specifically through learning analytics and the pedagogical uses of digital systems. She also investigates fundamental questions critical to improving undergraduate engineering degree pathways. . She earned her Ph.D. in Engineering Education from Virginia Tech. In 2013, Dr. Mohammadi-Aragh was honored as a promising new engineering education researcher when she was selected as an ASEE Educational Research and Methods Division Apprentice Faculty.

A Literate Programming Approach for Hardware Description Language Instruction

INTRODUCTION

Digital devices are ubiquitous in modern life. Over the last several decades, nearly all aspects of society are dependent on digital devices from entertainment devices to convenience devices to economy-critical and life-critical devices. A big reason for the proliferation of digital devices into every part of our lives is that digital systems have increasing capabilities at shrinking costs [1]. This seemingly contradictory march has often been characterized by Moore's Law, named after Gordon Moore, co-founder of Fairchild Semiconductor and CEO of Intel.

A critical challenge to continue this progress is management of digital circuit complexity. The days of hand-tuned digital circuits designed by single engineer are long gone. Modern digital circuits are far too complex for a single person to grasp and understand. To aid the modern digital circuit designer, hardware description languages (HDLs) such as ABEL [2], VHDL (now described by IEEE Standard 1076), Verilog (now described by IEEE Standard 1364), and others have been introduced. These HDLs can be used to describe both behavior and structure of a digital circuit. Therefore, HDLs are documentation tools that possess characteristics similar to traditional computer programming languages. The introduction and use of modern HDLs, predominantly Verilog and VHDL, are a hallmark of modern computer engineering curricula [3]. Because of the concurrent nature of digital devices, these HDLs allow for description of concurrent operations, similar to programming languages like Haskell and Ada. Most traditional engineering students have no exposure to such languages and often find HDLs challenging at first. Confusion can be compounded when learning Verilog, as Verilog is syntactically similar to the sequential procedural C programming commonly taught to engineering students.

Donald Knuth proposed literate programming (LP) [5] in 1984. LP is an approach to programming computers in which the programmer (author) composes the program in a form that is readable by humans. In short, the program should be like literature. This "literate program" is an essay that contains both explanation for human readers and executable statements for compilers. A quick glance at the current state of computer programming paradigms will reveal Knuth's efforts in LP were not widely adopted. It has been suggested that the failure of adoption of LP is due, in part, to the lack of sophistication of early LP tools [7].

In this research, the authors employ a modern software tool for LP in a senior-level/introductory-level course on digital systems design. The course reviews and revisits digital logic design topics from an introductory course, while adding complexity and incorporating practical aspects not covered in an prerequisite digital devices course. All student design output is captured in VHDL for simulation, verification, and synthesis to a modern FPGA. Student attitudes toward the LP were collected via a survey. Student performance on VHDL assignments and in the course appear to be similar to previous offerings of the course without LP.

BACKGROUND

Much time and effort has been spent detailing the challenges of teaching traditional computer programming languages. In spite of the pressing need for capable, creative, and above all competent programmers, educators struggle to effectively train students in these essential computing skills. McKraken's comprehensive examination of first-year CS students [4] reports that only approximately 20% of the surveyed students could solve programming problems expected by their instructors. In addition, the importance of programming continues to grow. Not only are CS and ECE students expected to master the art of programming, but student mastery of domain-specific languages such as MATLAB, R, Maple, Mathematica, and one or more HDLs is now required to be successful in all engineering disciplines.

Composing one's thoughts in a computer language – be it a traditional programming language or a HDL – has much in common with writing human languages. The syntax, structure, and vocabulary are different, and often more limited, but many of the same mental pathways are exercised. Many approaches to teaching computer languages seem to completely disregard writing when composing in a computer program. For example, computer programming textbooks present a program's idea in well-crafted prose, then instructors coach students in developing flow charts, UML diagrams, and design documents. But, in the end, all this of this writing is completely cast aside when the actual program is written, resulting in a mono-spaced mess only a compiler could love, as shown in Figure 1.

We assert that good writing leads to good thinking, and good thinking leads to good programs. In order to improve education of HDLs, writing must be reintroduced into writing HDL descriptions. This approach follows the philosophy of Literate Programming (LP) proposed by Donald Knuth in 1984.

Literate Programming

Knuth coined the term literate programming (LP) [5], in which an author composes an essay containing both explanation for readers and executable statements for compilers. LP mandates that a document consisting of intermingled code and prose be the creative output, and differs radically from the traditional software development model. Traditional software development views comments as an optional accompaniment to the code, and makes no attempt to connect the scattered comments into a coherent whole. The premise behind LP is that *you do not document a program*. Instead, LP has *you write a document that contains a program* [6].

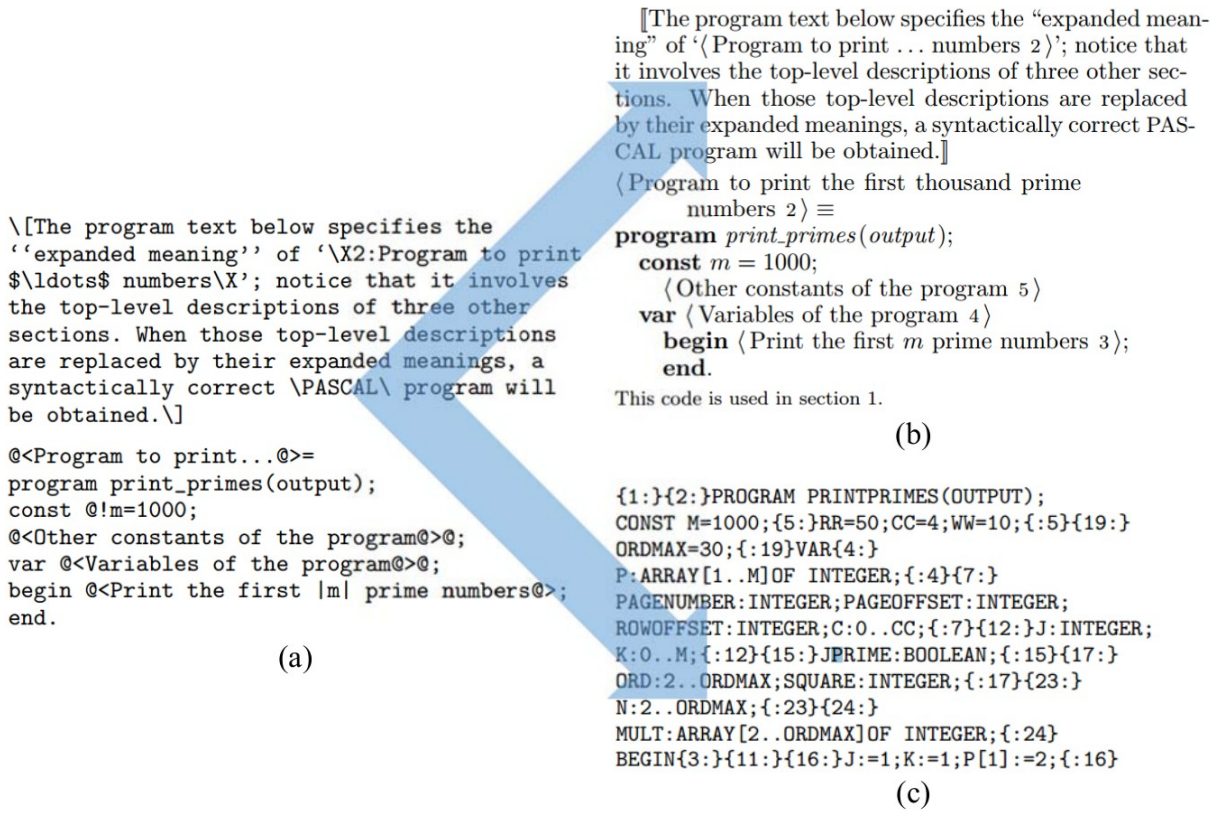
Using LP, educators can provide their students with an executable text – high-quality prose with detailed explanations, figures, diagrams, hyperlinks, etc. Simultaneously, students can read, learn, compile, execute, and explore ideas. Knuth's LP paradigm is also consistent with cognitive load theory [12], which states that keeping related concepts close, temporally or spatially, can improve the ability of students to grasp difficult ideas [13][14].

```
-------------------------------------------------------------------------
-- Title: Parallel to Serial Converter (PAR2SER)
-- Project: ASEE 2019
-------------------------------------------------------------------------
-- File: par2ser.vhd
-- Author:   <foo@bar.engr.com>
-------------------------------------------------------------------------
-- Description : -- Implements a simple 8-bit parallel to serial conv
-------------------------------------------------------------------------
-- Modification history : -- 2019/02/19 : created
-------------------------------------------------------------------------
library ieee;
use ieee.std_logic_1164.all;
entity PAR2SER is
     port (DIN : in std_logic_vector (7 downto 0); -- input register
     MODE : in std_logic_vector (1 downto 0);      -- mode selection
     CLK, RESET : in std_logic;                    -- clock and reset
     SDOUT : out std_logic);                       -- output data
end PAR2SER;

-- purpose: Implement main architecture of PAR2SER
architecture BEHAVIOR of PAR2SER is
     signal IDATA : std_logic_vector(7 downto 0);  -- internal data
begin  -- BEHAVIOR
     -- purpose: Main process
     process (CLK, RESET)
     begin  -- process
         -- activities triggered by asynchronous reset (active high)
         if RESET = '1' then
             SDOUT <= '0';
             IDATA <= "00000000";
         -- activities triggered by rising edge of clock
         elsif CLK'event and CLK = '1' then
             case MODE is
                 when "00" =>            -- no operation
                     null;
                 when "01" =>            -- load operation
                     IDATA <= DIN;
                 when "10" =>            -- shift left
                     SDOUT <= IDATA(7);
                     for mloop in 6 downto 0 loop
                         IDATA(mloop+1) <= IDATA(mloop);
                     end loop;  -- mloop
                 when others =>          -- no operation otherwise
                     Null;
               end case;
             end if;
     end process;
 end BEHAVIOR;
```

Figure 1: Representative VHDL code

Knuth's initial literate programming implementation, called WEB, incorporated ideas of LP along with his work in digital typography in TeX. WEB used macros and preprocessor directives to create a meta-language that would render a WEB file into two separate outputs: a human-readable document and source code for compilation. First, the WEB source document (see Figure 2a) consists of a series of cryptic formatting instructions, some of which allow the inclusion of Pascal code, the only programming language supported by WEB. These cryptic TeX-based formatting commands raised a high barrier of entry and discouraged adoption. Second, Knuth's choice of an additional source document means that the formatted human-readable document (in Figure 2b) and the source code (in Figure 2c) had to be created by LP tools, WEB's utilities *tangle* and *weave*, respectively. Neither rendered text or source code could be edited directly. Making minor edits to the formatted document in Figure 2b required finding the text in the source document in Figure 2a which produced it, editing the offending lines, then regenerate the formatted document to verify that the correct edit was performed. Likewise, modifying the source code in Figure 2c requires a similarly laborious process. Minor textual edits become major chores. Finally, traditional development tools such as debuggers and profilers are extremely difficult to deploy for WEB documents and their associated programs.



```
\[The program text below specifies the
''expanded meaning'' of '\X2:Program to print
$\ldots$ numbers\X'; notice that it involves
the top-level descriptions of three other
sections. When those top-level descriptions
are replaced by their expanded meanings, a
syntactically correct \PASCAL\ program will
be obtained.\]

@<Program to print...@>=
program print_primes(output);
const @!m=1000;
@<Other constants of the program@>@;
var @<Variables of the program@>@;
begin @<Print the first |m| prime numbers@>;
end.
```

(a)

⟦The program text below specifies the "expanded meaning" of ⟨Program to print ... numbers 2⟩; notice that it involves the top-level descriptions of three other sections. When those top-level descriptions are replaced by their expanded meanings, a syntactically correct PASCAL program will be obtained.⟧

⟨Program to print the first thousand prime numbers 2⟩ ≡

```
program print_primes(output);
  const m = 1000;
    ⟨Other constants of the program 5⟩
  var ⟨Variables of the program 4⟩
    begin ⟨Print the first m prime numbers 3⟩;
    end.
```

This code is used in section 1.

(b)

```
{1:}{2:}PROGRAM PRINTPRIMES(OUTPUT);
CONST M=1000;{5:}RR=50;CC=4;WW=10;{:5}{19:}
ORDMAX=30;{:19}VAR{4:}
P:ARRAY[1..M]OF INTEGER;{:4}{7:}
PAGENUMBER:INTEGER;PAGEOFFSET:INTEGER;
ROWOFFSET:INTEGER;C:0..CC;{:7}{12:}J:INTEGER;
K:0..M;{:12}{15:}JPRIME:BOOLEAN;{:15}{17:}
ORD:2..ORDMAX;SQUARE:INTEGER;{:17}{23:}
N:2..ORDMAX;{:23}{24:}
MULT:ARRAY[2..ORDMAX]OF INTEGER;{:24}
BEGIN{3:}{11:}{16:}J:=1;K:=1;P[1]:=2;{:16}
```

(c)

Figure 2: Knuth's WEB system for LP transforms the input source document in (a) to the formatted output in (b) and the source code in (c) as illustrated by the large arrows.[5]

Later LP implementations addressed the first problem in Knuth's approach: weaknesses in language support and formatting. Some variants support additional programming languages: CWEB (for C), FWEB (Fortran, C, and C++), xmLP (XML), pyWeb, FunnelWeb, nuweb, and noweb (any language). Other tools provided simpler formatting syntax: nuweb uses LaTeX; noweb a simpler set of literate programming directives and also allows use of LaTeX; pyWeb uses restructured text. Pieterse's survey reviews other literate programming approaches [7]. However, the second problem exists by Knuth's design: choosing a source document from which LP tools produce both source code and a formatted document prevents direct modification of either the source code or the formatted document, isolating authors from the writing they must do. For these reasons, no LP tool has gained widespread acceptance in the programming community or for sustained pedagogical use. This last point is substantiated by noting that most education-focused research using LP tools took place in the 1990s. Efforts in this area include using LP tools to grade homework submissions [8], teach programming [9], or write better comments [10]. The work in [9] and [10] report some success, but both cite student complaints about the difficulty of using LP tools.

More recently, documentation generated directly from code has became widespread. Documentation generators, such as Doxygen and JavaDoc, produce a document directly from formatted comments within source code, thus overcoming WEB's second problem. These document generation tools are widely used by programmers, with thousands of programs employing these tools or variants. While tools like Doxygen, JavaDoc, and others, were inspired by LP principles [11], they only document the program's external interface such as the application programming interface, or API. Today's popular document generators doe not easily allow elaboration about the internal workings of a program – the primary goal of LP.

CodeChat – a modern LP approach

One of the authors of this paper has developed a modern tool – CodeChat tool [15] – that attempts to address both of WEB's failings while still incorporating the main premise of LP. The CodeChat tool combines the strengths of both documentation generators and LP tools. CodeChat builds a formatted document directly from source code, using human-readable ReStructuredText [16] as markup contain in the language's native comments. Figure 3 shows the user view of CodeChat.

CodeChat contains a synchronization mechanism which matches source code with the corresponding web output, making editing of either straightforward. CodeChat users simply edit a programming language source file in the left pane. The user places very intuitive ReStructuredText markup in the language's comment lines. The CodeChat tool parses the source code file, locates the markup, and renders the program and annotations into human-readable form in real-time into the right-hand pane. The rendered output can be generated in many different formats, including HTML, TeX, LaTeX, and DocBook. This single-view paradigm is understood by today's computer users within a few moments of use. CodeChat provides an easy-to-use and viable platform for LP techniques in programming, and is an ideal tool for conducting research into the use of LP in programming education pedagogy. CodeChat supports LP in more than 200 programming and scripting language "right out of the box."

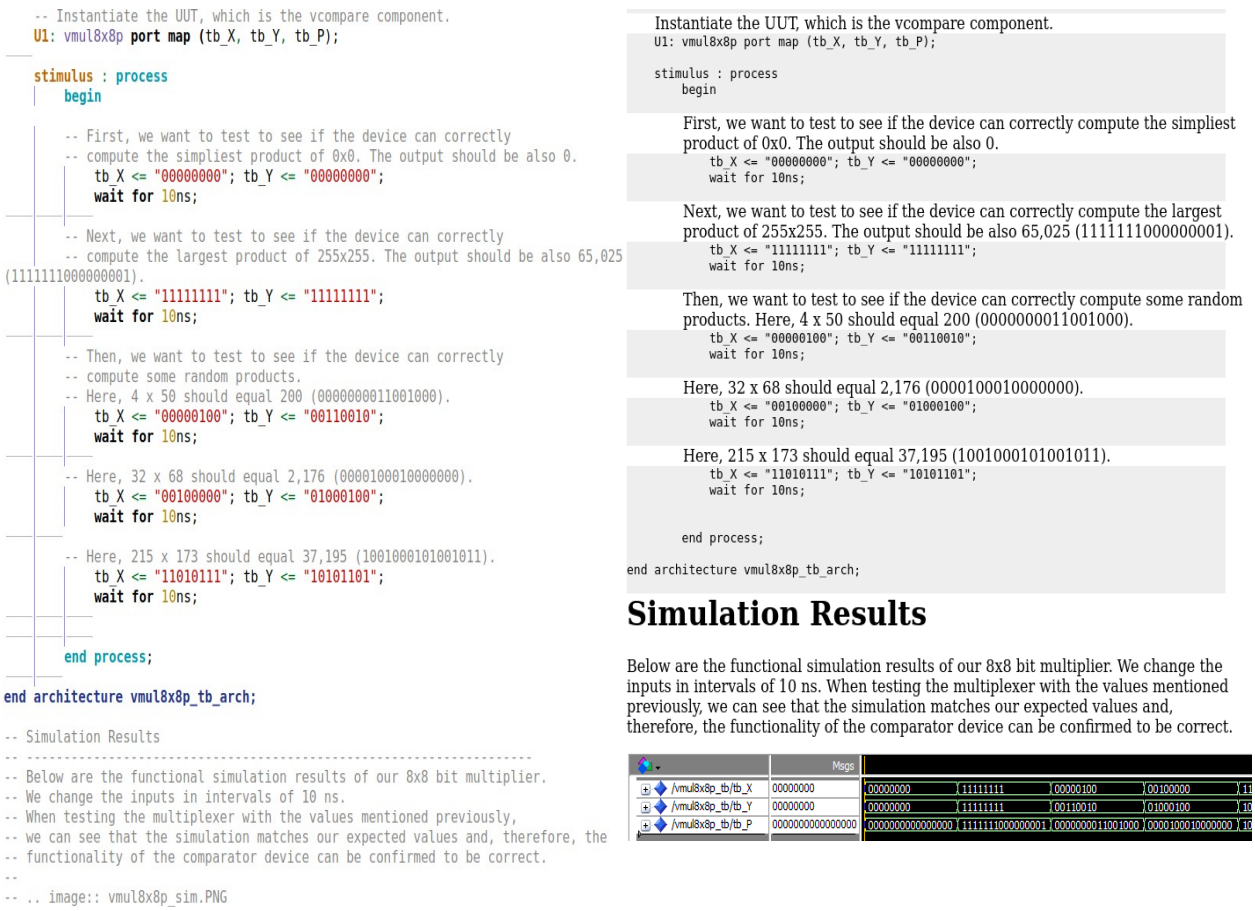


Figure 3: CodeChat, the LP tool used in this study, transforms traditional computer programming source files (on the left) into a variety of more human-readable forms such as web-browser rendered HTML (on the right). Transformation is synchronized to editing and performed in real-time.

LP for HDL education

The introduction and use of modern HDLs, predominantly Verilog and VHDL, are a hallmark of modern computer engineering curricula [3]. Because digital devices operate concurrently, HDLs allow for description of concurrent operations, similar to programming languages like Haskell and Ada. The complexity of hardware descriptions coupled with HDL similarities to sequential programming languages has led the authors to propose the idea of introducing LP into HDL education. Using LP to improve HDL education has not been widely investigated. One early attempt involved an approach very similar to Knuth's WEB approach [18][19] using a Prolog logic program to generate a human-readable form and a Verilog HDL file. The main thrust of this effort was to capture the formal operational semantics of the description and animate the

behavior of parallelisms. The approach taken in [18] and [19] will clearly suffer from the failings of Knuth's efforts with regard to a maintainability, tool-chain integration, and user familiarity.

The goal of LP is to get the students to view their HDL descriptions as being a product for human consumption instead of tool-chain consumption. Furthermore, the more expressive nature of human language and the student's experience with human language should allow the students to more clearly express the behavior and interactions of the digital hardware. Using LP in HDL education also reinforces the spirit of why HDLs were created in the first place – to describe hardware behavior. An HDL description should describe – in as human-readable terms as possible – what the hardware design "looks like" or "how it behaves". The HDL description is, first and foremost, an essay written to fellow designers describing digital hardware. This description also happens to be in an "executable form" thanks to the HDL tool chain.

APPROACH

The lead author teaches a split-level (senior and introductory graduate) course in digital system design. The course reviews and revisits digital logic design topics from an introductory course, while adding complexity and incorporating practical aspects not covered in the earlier course. All of the coursework is captured in VHDL. The course concludes with a small design project. The first half of the course focuses on instruction in VHDL syntax, and is accompanied by lab periods in which students write descriptions – mostly structural – of simple hardware designs in VHDL. Students compose appropriate test benches to exercise and validate their designs. The second half of the course delves into timing and system design issues along with a more behavioral approach to hardware description. Weekly lab assignments have the student design ever-larger components that may be used in their design project.

This paper examined a cohort of 36 students enrolled one section of the course during a recent fall semester. The cohort was composed of ten electrical engineering majors who took the class as an elective, and twenty-six computer engineering majors for whom the course is required. The course offering described here was similar to previous semesters with nearly identical topical coverage and pacing using the same text, lecture materials, and lab facilities. Due to faculty staffing changes, the previous iterations of the course were taught by another faculty member who did not employ LP methods. The course is offered once per academic year in a single section prohibiting a convenient control group.

The course was composed of lecture periods (twice a week; one hour each) and a weekly lab session (a single two hour session each week) with a graduate teaching assistant to help the students. Lecture periods were sprinkled with collaborative active exercises. Student were encouraged to work in teams on the lecture active exercises and on lab tasks. The instructor often had to expressly direct the collaborative effort as most students would choose to work alone if given the choice. The complexity of VHDL and the overhead of tool chain processes largely prohibited the use of the industrial-strength synthesis tools in-class for the active exercises. Active exercises were mostly centered around the design approach for a particular problem, with students writing "pseudo-code" VHDL. Homework and lab activities consisted of students writing VHDL descriptions annotated with explanations in the CodeChat tool. The addition of LP annotations to their VHDL was the only significant change to VHDL coding assignments

compared to previous semesters. In previous semesters, students wrote formal lab reports to describe their design approach and results. The CodeChat tool allowed students to include hyperlinks, figures, tables, equations, FSM diagrams, timing waveforms, etc. directly in their HDL descriptions. See Figure 3. Since the code itself was descriptive, and each design's test bench could be annotated with results obtained from the VHDL tools. Students were not required to submit a formal lab report. Each design task had a deliverable of VHDL files that were to be liberally annotated with descriptions, explanations, figures, timing diagrams, and observations.

DATA & DISCUSSION

It is desired to ascertain whether LP had an impact on the students' ability to successfully learn VHDL to capture digital systems behavior. The class population was composed of computer engineering and electrical engineering majors. The course is required for the former, and serves as a technical elective for the latter. While computer engineering and electrical engineering are very similar, the computer science and programming skills and experience of the computer engineering majors usually is much more substantial than those of electrical engineering students. Does this difference in background affect a student's view of LP as it is used in a digital systems design course?

An ideal, but often difficult to deploy, approach to testing the efficacy of LP in the digital systems design course would be formal experiment with control and treatment groups of students taught under similar educational conditions during the same academic period. This was not possible do to existing scheduling and faculty workloads. The experience described here did not employ a control group. Only one section of the course is offered annually. Another approach is for the instructor to create control and treatment groups within the same lecture and lab population. This approach is problematic as students would feel that they are being asked to do assignments that are viewed as more difficult, challenging, or otherwise detrimental to their grade. Control and treatment groups could be formed by student self-selection, but groups of comparable sizes, abilities, and motivation are very suspect with this scheme. Given the circumstances, a decision was made to apply the treatment to the entire cohort.

Comparison of cohorts in previous versions of the same course is always difficult. Every semester is different with two distinct groups of students are involved. In general, the same course in a different semester means students are evaluated under different conditions with different exams, etc. The cohort examined here completed the digital systems design course with exam, project, and course scores reasonably identical to scores in the previous versions. The instructor's observation was that the students in the class described here demonstrated abilities and knowledge equal to the students taught without LP in previous semesters. Other faculty have reported their sense of the students abilities in the topics and skills from the course to be similar.

At the conclusion of the course, students were asked to respond to a self-assessment concerning a wide variety of course activities. Survey questions were Likert-scale. Table 1 show the survey question relevant to this paper.

	Survey Statement	***Choices***
Q1	*I would rate my knowledge of HDL and digital systems design knowledge at the start of the course*	*Poor* (0), *Fair* (1), *Satisfactory* (2), *Very Good* (3), *Excellent* (4)
Q2	*I would rate my knowledge of HDL and digital systems design knowledge at the conclusion of the course*	*Poor* (0), *Fair* (1), *Satisfactory* (2), *Very Good* (3), *Excellent* (4)
Q3	*Writing detailed and descriptive comments in my HDL descriptions helped me to learn.*	*Strongly disagree* (-2), *Disagree* (1), *Neither disagree or agree* (0), *Agree* (+1), *Strongly agree* (+2)
Q4	*I would have rather had traditional Q&A questions instead of writing documented HDL descriptions for course homework*	*Strongly disagree* (-2), *Disagree* (1), *Neither disagree or agree* (0), *Agree* (+1), *Strongly agree* (+2)
Q5	*My digital design efforts in this class would have been better summarized with a formal report/paper*	*Strongly disagree* (-2), *Disagree* (1), *Neither disagree or agree* (0), *Agree* (+1), *Strongly agree* (+2)
Q6	*Putting my design ideas into words helps me to see errors in my design and improves my overall output*	*Strongly disagree* (-2), *Disagree* (1), *Neither disagree or agree* (0), *Agree* (+1), *Strongly agree* (+2)

Table 1: Relevant student survey questions administered after the course

Two first two questions asked students were asked to rank their knowledge and abilities of digital design concepts and hardware description languages. Students could choose a response from the following list: “poor, “fair”, “satisfactory”, “very good”, or “excellent”. The responses were mapped to the values zero (poor) through four (excellent). The questions had the student rate their knowledge of digital system design and HDL at the start of the course (Q1) and at the end of the course (Q2). Table 2 gives the statistics of the two questions self-assessed abilities by major.

Knowledge of DSD and HDL	*CmpE*	*EE*	*Entire cohort*
Q1 (mean, std. deviation, median)	0.29, 0.59, 0	0.00, 0.00, 0	0.21, 0.51, 1
Q2 (mean, std. deviation, median)	2.12, 1.22, 2	1.86, 1.07, 2	2.04, 1.16, 2

Table 2: Self-reported knowledge of digital system design and HDL before (Q1) and after (Q2) the course.

A small number (four out of sixteen) of computer engineers reported have some knowledge of the subject before the course started, while electrical engineering majors all reported knowing effectively nothing about the course topics. At the end of the course, most students of both majors reported their knowledge as being “satisfactory” or better. The difference between a student’s response (Q2-Q1) would indicate the course affect on their knowledge. Computer engineering and electrical engineering majors reported a mean delta of 1.82 and 1.86, respectively, between Q1 and Q2.

Students were also asked to rate how well they agreed with several statements about the effect that LP contributed to their learning. Students were given the choice of “strongly disagree”, “disagree”, “neutral”, “agree” and “strongly agree”. In these survey questions, these choices were mapped to values -2 (strongly disagree) to +2 (strongly agree). The numeric values chosen allow that the sign of the response grossly indicate whether the response is negative, neutral, or positive. Students were allowed to also choose “not applicable”. N/A responses were discarded.

Q3 asked a student about the efficacy of the LP approach used in the course. They were asked how much they agreed to the statement “Writing detailed and descriptive comments in my HDL descriptions helped me to learn”. Q4 asked a student if they would have preferred the traditional homework set of problems to work instead of VHDL assignments using the LP approach. Specifically, students responded to the question “I would have rather had traditional Q&A questions instead of writing documented HDL description for course homework”. Q5 was posed to judge whether the student simply wanted to avoid “writing”. Students were asked if they would have preferred usual formal lab report required in the prior course offerings and in many of the prerequisite lab courses. Q5 asked a student how much they agreed with “My digital design efforts in this class would have been better summarized with a formal report/paper”. Later in the student survey, students were asked to give their opinion on the LP paradigm again. In Q6, the survey question was worded so as not to invoke the actual lab experience with the tools, the teaching assistant, the assignments, etc, but simply about the idea of using writing to make better HDL descriptions. Students were asked how much they agreed with the statement “Putting my design ideas into words helps me to see errors in my design and improves my overall output”. Table 3 give the data summary of the survey results. No student marked “Not Applicable” in their response of Q3-Q6.

LP and writing in HDL	*CmpE*	*EE*	*Entire cohort*
Q3 (mean, std. deviation, median)	-0.41, 1.33, 0	0.00, 0.00, 0	-0.29, 1.27, 0
Q4 (mean, std. deviation, median)	-0.18, 1.19, 0	0.29, 1.25, 0	-0.21, 1.18, 0
Q5 (mean, std. deviation, median)	-0.65, 1.17, -1	-1.29, 0.76, -1	-0.83, 1.09, -1
Q6 (mean, std. deviation, median)	-0.06, 1.09, 0	0.29, 0.95, 1	0.04, 1.04, 0

Table 3: Self-reported knowledge of digital system design and HDL before (Q1) and after (Q2) the course.

The results from Q3 indicate that computer engineers were slightly more negative than neutral about the LP approach. Electrical engineers were exactly neutral as a group. Of course, individual students reported “strongly agree” and “agree”, but there were roughly equal numbers of students reporting “strongly disagree” and “disagree”. The CodeChat tool was created by one of the authors. The tool is not as polished as commercial or mature open-source software applications. The tool itself works as advertised and is quite stable. Students never reported dissatisfaction with tool functionality. However, installation of CodeChat tool as problematic. To aid in CodeChat development, the CodeChat tool uses a wide variety of open-source libraries and supporting frameworks. Several of these software packages had varying compatibility issues with student laptops. Students were quite vocal in expressing their frustration with the

installation process at the beginning of the semester. The students' unhappiness was compounded as installation on one student's laptop would be smooth and flawless, and another student with an identical computer would experience installation problems. Ultimately, several installation problems were a result of students not following the detailed installation instruction provided by the instructor. Eventually, the instructor was able to assist every student and get the tool operational. However, this initial negative experience likely colored the student's impression of the CodeChat tool and the LP paradigm.

In Q4, computer engineers were indicated a slight preference for homework based on LP-infused VHDL over a set of traditional homework problems. Electrical engineers indicated an even strong preference for LP and VHDL, although neither group was overwhelming in their preference.

Not surprisingly, Q5 responses indicate that both majors are more emphatic about preferring the LP approach to VHDL compared to writing a formal lab report. As with Q4, the preference among the electrical engineers was stronger than the computer engineers.

Q6 was worded to try to elicit the student response about LP without ascribing it to the course and lab experience in support of Q3. Both majors were effectively neutral in their views, with electrical engineering majors being slightly positive on average, and computer engineers as a group were nearly neutral. Note that response in Q6 are a bit more positive than the very similar question Q3. Students appear to recognize that writing and natural language may help them express their designs, but the LP approach or negative feelings due to CodeChat installation drive their opinions a bit more negative when the writing involves commenting their HDL.

A Mann-Whitney U test was chosen as a non-parametric test to compare the two populations [20]. Table 4 shows the results of the Mann-Whitney test. No significant differences between the two populations are observed on questions Q1-Q6.

	Survey Statement	***U value***	***P value***
Q1	*I would rate my knowledge of HDL and digital systems design knowledge at the start of the course*	45.5	0.374
Q2	*I would rate my knowledge of HDL and digital systems design knowledge at the conclusion of the course*	52	0.634
Q3	*Writing detailed and descriptive comments in my HDL descriptions helped me to learn.*	71	0.465
Q4	*I would have rather had traditional Q&A questions instead of writing documented HDL descriptions for course homework*	56	0.824
Q5	*My digital design efforts in this class would have been better summarized with a formal report/paper*	41	0.240
Q6	*Putting my design ideas into words helps me to see errors in my design and improves my overall output*	71	0.465

Table 4: Results from Mann-Whitney U test. Threshold value for data is 28.

Other studies have reported similar student satisfaction results when introduced to the LP paradigm. Students were somewhat ambivalent about the effectiveness of LP to improve their HDL in the digital design course. One theory is the computer engineers are more comfortable with the topics covered in the course and with programming languages and documentation than the electrical engineers. The design tasks in the course were not overly complex, so computer engineers may have felt the LP paradigm was simply too much overhead and work for the reasonably simple technical HDL that followed. Electrical engineers are less comfortable in the course described here, and the LP approach may allowed them to better express themselves. If this is a valid view, one might expect that student enthusiasm to increase as assignments and designs get more complicated. Such a result would also agree with literature that computer programmers tend to comment more heavily in complicated code, or code that they do not initially understand [21], [22]. In essence, developers tend to elaborate in more descriptive and detailed comments when the coding is difficult. LP would provide the developer with a more suitable mechanism for human-digestible descriptions.

Students, especially the computer engineering majors who have more experience with computing languages, may have also suffered from the mistaken impression that any computer activity time spent not creating “working code” is a waste of time. With more experience, they will realize that documentation of computer code or VHDL descriptions leads to a much higher maintainability and, ultimately, shorter overall development cycles and lower costs [21], [22].

Anecdotal observations by the instructor and other faculty are that student comprehension and performance seemed largely unchanged compared to previous semesters. The next step would be to form a controlled experiment wherein one group will use traditional editors or IDEs to develop largely undocumented HDL descriptions. A treatment group would use tools and LP approach described here. Incorporation of formal product metrics such as defect production and development time would also give more objective measures to the quality of student output. Finally, student performance on other course measures or later academic outcomes might indicate a difference.

CONCLUSIONS

The authors taught a senior-level course on digital systems design required by their computer engineering program. A number of electrical engineering students were enrolled in the course as an elective. In the course, digital logic design topics from an introductory course are reviewed and explored again in detail using VHDL. The authors introduced Knuth’s literate programming paradigm, which treats a program as an essay, intermingling VHDL code with explanation in a very human-readable and friendly document. his stands in contrast to traditional programming pedagogy where difficult-to-understand code is isolated from its explanation. The hope is that LP would aid student learning, and improve the quality of the student output. Situations in scheduling prohibited a true control and treatment group approach for this study. The study as described here indicates that student learning and academic performance appears unchanged from more “traditional” teaching approaches. Computer engineering students reported being neutral to very slightly negative about use of LP, with electrical engineering students reporting a slightly positive view of the approach. The authors have reason to believe that the logistics of the LP tool installation biased students toward the negative. Arguably, the population size was small

and the study was limited. It is hoped that other researchers will apply LP to HDL education at their institutions. Additional study is required with more students. The instructor's experience using LP to teach HDL should also create a more positive and effective learning environment in future attempts.

REFERENCES

[1] Moore, Gordon E. "Cramming more components onto integrated circuits". Electronics, April 1965.
[2] Lee, Kyu Y., et al. "A High-Level Design Language for Programmable Logic Devices". VLSI Design. Vol. 6, no. (6): 50–62. (June 1985)
[3] ACM/IEEE, Computer Engineering Curricula 2016: Curriculum Guidelines for Undergraduate Degree Programs in Computer Engineering, Dec. 2016.
[4] McKracken, M. et al. "A Multi-national, multi-institutional study of assessment of programming skills of first-year CS Students", SIGCSE Bulletin, vol. 33, no. 4, pp. 125-180, Dec. 2001.
[5] Knuth, D. Literate Programming, CSLI. p. 99, 1984.
[6] Skaller, M. J., in a Charming Python interview. Retrieved from http://gnosis.cx/publish/programming/charming_py-thon_8.html.
[7] Pieterse, V., Derrick G. K., & Boake, A. "A case for contemporary literate programming", SAICSIT, 75, 2-9, Stellenbosch, Western Cape, South Africa, 2004.
[8] Hurst, A. J. "Literate programming as an aid to marking student assignments", Proceedings of the 1st Australasian conference on Computer Science education, 280-286, 1996.
[9] Childs, B., Dunn, D., & Lively, W. "Teaching CS/1 Courses in a Literate Manner," Proceedings of the TeX Users Group Conference, St. Petersburg, Florida, July 24-28, Volume 16, No. 3, p. 300-309, 1995.
[10] Shum, S. & Cook, C. "Using Literate Programming to Teach Good Programming Practices," Proceedings of the twenty-fifth SIGCSE symposium on Computer Science education, p. 66-70, 1994.
[11] See http://rant.gulbrandsen.priv.no/udoc/history.
[12] Sweller, J. "Cognitive load during problem solving: Effects on learning," Cognitive science, 12(2), 257-285, 1988.
[13] Sweller, J., & Chandler, P., "Why some material is difficult to learn," Cognition and instruction, 12(3), 185-233, 1994.
[14] Sweller, J., Van Merriënboer, J. J., & Paas, F. G., "Cognitive architecture and instructional design," Educational Psychology Review, 10(3), 251-296, 1998.
[15] B. A. Jones, M. Jean Mohammadi-Aragh, A. K. Barton, D. Reese, and H. Pan, "Writing-to-Learn-to-Program: Examining the Need for a New Genre in Programming Pedagogy," Proceedings of the 122nd ASEE Annual Conference and Exposition, Seattle, WA, USA, June 2015.
[16] See http://docutils.sourceforge.net/rst.html
[17] Jones, B. A., & Mohammadi-Aragh, M. J. "Employing Literate Programming Instruction in a Microprocessors Course", Proceedings of 2016 ASEE Annual Conference & Exposition, New Orleans, Louisiana, June 2016.
[18] Bowen, J.P. "Combining Operational Semantics, Logic Programming and Literate Programming in the Specification and Animation of the Verilog Hardware Description Language." IFM (2000).
[19] Bowen, J.P., Jifeng, H., and Qiwen, X. "An animatable operational semantics of the Verilog hardware description language" Proc. Third International Conference on Formal Engineering Methods, Sept. 2000.
[20] de Winter, J.C.F. and D. Dodou, "Five-Point Likert Items: t test versus Mann-Whitney-Wilcoxon", Practical Assessment, Research and Evaluation, v. 15 n.11, 2010.
[21] R.S. Pressman, *Software Engineering: A Practitioner's Approach 5/e*, Mc-Graw Hill, 2001.
[22] S. McConnell, *Code Complete, 2/e*, Microsoft Press, 2004.