Less is More: Developing complex designs using a minimal HDL subset in an introductory digital devices laboratory

Dr. Bryan A. Jones, Mississippi State University

Bryan A. Jones (S'00–M'00) 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 Assistant Professor with the 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 current research interests include robotics, real-time control-system implementation, rapid prototyping for real-time systems, and modeling and analysis of mechatronic systems

Ms. Jane Nicholson Moorhead, Mississippi State University
Dr. M. Jean Mohammadi-Aragh, Mississippi State University

Dr. M. Jean Mohammadi-Aragh is an assistant research professor with a joint appointment in the Bagley College of Engineering dean's office and the Department of Electrical and Computer Engineering at Mississippi State University. Through her role in the Hearin Engineering First-year Experiences (EFX) Program, she is assessing the college's current first-year engineering efforts, conducting rigorous engineering education research to improve first-year experiences, and promoting the adoption of evidence-based instructional practices. In addition to research in first year engineering, Dr. Mohammadi-Aragh investigates technology-supported classroom learning and using scientific visualization to improve understanding of complex phenomena. She earned her Ph.D. (2013) in Engineering Education from Virginia Tech, and both her M.S. (2004) and B.S. (2002) in Computer Engineering from Mississippi State. 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.

©American Society for Engineering Education, 2015

Less is More: Developing complex designs using a minimal HDL subset in an introductory digital devices laboratory

While the core knowledge composing a traditional introductory course on digital devices (Boolean algebra, finite state machines) has been stable for some time, several recent trends challenge educators. First, the move from older PLDs such as the classic 22V10 PAL to more complex FPGAs requires a transition from relatively simple hardware description languages (HDLs) such as PALASM or ABEL to significantly more complex modern HDLs such as Verilog or VHDL, which professional design engineers spend years to master. Second, the wide availability of high gate count FPGAs, which enables students to create amazingly complex designs involving datapath components such as adders and counters, requires the teaching of many more complex components in a one semester course. This two-fold increase in complexity requires a redesign of the traditional introductory digital design laboratory to enable students to create complex designs using a relatively simple design methodology.

Therefore, this paper proposes the use of a minimal subset of an HDL (Verilog, in this case) complemented by judicious use of schematic capture and its library of datapath components to enable students to create modern, register-transfer level (RTL) driven designs without requiring students to develop a deep expertise in an HDL. In addition, the use of assertions supported by newer HDLs provides students with immediate feedback on the correctness of their designs, a vital feature which enables them to succeed in the creation of complex designs.

1. Background

The rapid advances in the density of integrated circuits dramatically changes the content which should be covered in an introductory course on digital logic. In particular, the history of advances in programmable logic devices, which enable students in a laboratory context to implement a digital system, defines the size of the design which students can be expected to create. In the 1970s, programmable logic arrays (PLAs) allowed the implementation of arbitrary sum-of-products expressions, but were difficult to use. Programmable logic devices (PLDs) in the 1980s, such as the popular 22V10, provided improvements in ease of use and allowed the design of moderately complex state machines (those with up to 10 flip-flops); these began to find their way into the laboratory of a digital devices course. The 1990s saw the advent of complex programmable logic devices (CPLDs) and field-programmable gate arrays (FPGAs) and their introduction into the classroom. The 2000s and beyond witnessed the explosion of FPGAs, with densities and capabilities rising exponentially; current offerings from Xilix and Altera exceed 1 million flip-flops, with comparable combinational logic.

To harness this power, the tools used to define the logic implemented by these devices have likewise changed dramatically. Early tools introduced in the 1980s such as the PAL assembler (PALASM) or the Advanced Boolean Equation Language (ABEL) provided a means of specifying

each register's input in terms of combinational equations, suitable for synthesizing small state machines. However, these tools proved insufficient for engineers developing large application-specific integrated circuits (ASICs); designers then turned to hardware description languages (HDLs), Verilog and VHDL, to create larger designs. While these HDLs began as simulation-only languages, they later (in the 1990s) gained the ability to synthesize a design from the description. Finally, SystemC, introduced in the early 2000s, utilized C++ supplemented with HDL-specific classes to provide equivalent capabilities in a well-known programming language. With the widespread use of HDLs, support for simpler tools is waning rapidly; for example, neither Altera nor Xilinx support ABEL in current products.[1]

The advent of HDLs introduced significant additional complexity into the design process. These languages support structural modeling, in which every combinational gate is described textually; behavioral modeling, in which logical operators (and, or, not, etc.) imply gates; and support many features (arbitrary time delays, for example) which cannot be synthesized but which are critical for testing a design. A wide variety of verification strategies – VHDL-93 provides an `assert` statement, while Verilog requires either a third-party library or use of SystemVerilog – makes testing more difficult. Finally, the wide variety of language variants (considering just Verilog: Verilog-95,[2] Verilog-2001,[3,4] Verilog-2005,[5] SystemVerilog-2005,[6] SystemVerilog-2009,[7] SystemVerilog-2012[8]) and varying support for these versions among tool vendors further complicates their use for instructional purposes.

As a result, educators developing a digital devices laboratory component now face a bewildering variety of choices: what language (VHDL, Verilog, SystemC), what features of that language (structural, behavioral, simulation), what hardware vendor (Altera and Xilinx dominate the industry), and what verification approach? Textbooks authors likewise struggle to address these concerns: Vahid[9] covers both VHDL and Verilog, but places the coverage in a separate chapter at the end of the text. Wakerly's classic text[10] introduces Verilog, VHDL, and ABEL, then uses all three throughout his 800+ page book. Mano and Ciletti[11] provide the only book in this group with a set of lab experiments included; however, the experiments are based on wiring up 74x logic gates, which are then retrofitted to Verilog. The following insights were gained in the process of redesigning the laboratory component for ECE 3714, Digital Devices and Logic Design, offered as a required course typically taken in the first semester of the sophomore year at Mississippi State University for ECE and CS students.

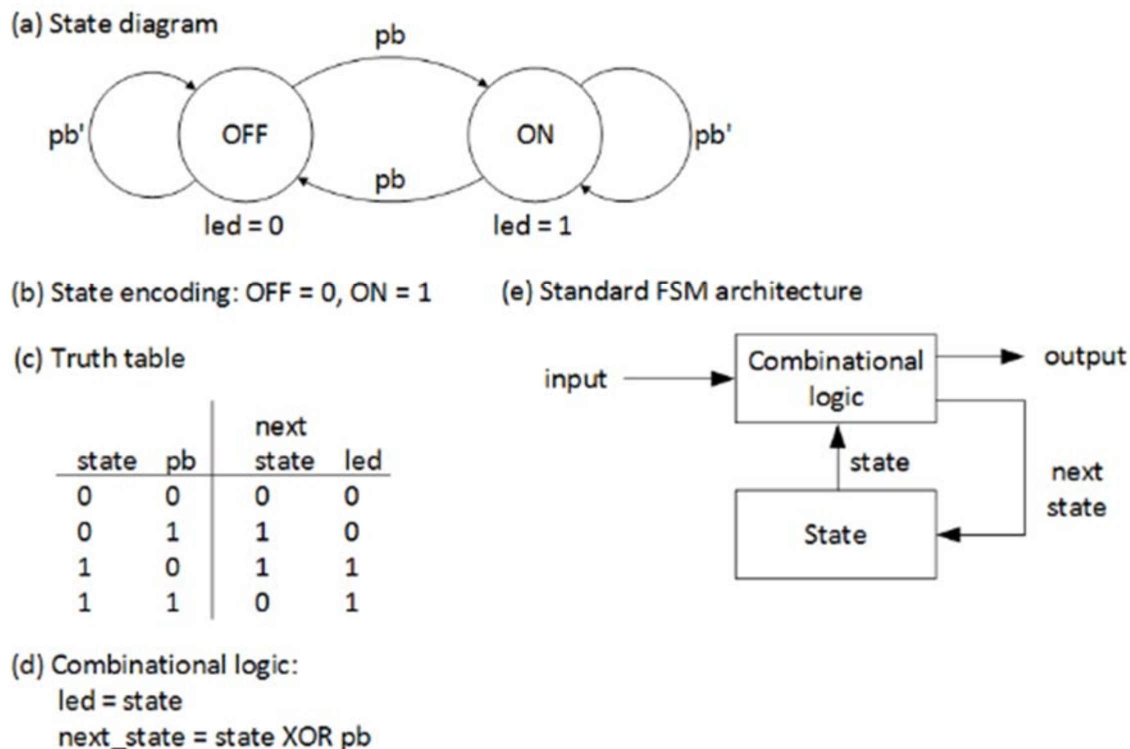
2. Approach

In Newstetter and Svinicki's[12] overview of engineering education learning theories, they highlight constructivism as the most prevalent learning theory in literature. From a constructivist perspective, learning is a process that involves the learner creating personal mental models for a given subject. Constructivism is the theoretical basis for most active learning pedagogy[13]. In a construc-

tivist classroom, instructors support students' efforts of active organization by providing clear connections between students' prior knowledge, classroom instruction, instructor demonstrations, and students' laboratory experiments.

2.1 Design

Applying constructivism to this effort provided our primary guiding principle: students should be better able to transfer their knowledge from the classroom to the laboratory when there is a clear visual connection between the two. By following a prescribed process methodology, a student should be able to easily transfer their designs from paper onto a programmable device. For example, consider a state machine which blinks its output LED when an input pushbutton is pressed. As shown in Figure 1, the design process begins with a state diagram (a), which, given a state encoding (b), implies a truth table (c) which leads to combinational logic (d), a part of the standard architecture of a finite state machine (FSM) (e). To implement this in a laboratory environment, several decisions must be made.



state	pb	next state	led
0	0	0	0
0	1	1	0
1	0	1	1
1	1	0	1

Figure 1: FSM design process.

First, should the design be implemented using discrete 74x logic gates? While the simplicity is beneficial, this approach has several drawbacks. First, the approach scales poorly when moving to larger datapath designs. For example, the design shown in Figure 5 would require a prohibitive

number of devices and wiring. Second, it provides no formal method for simulation and verification. Therefore, a viable laboratory component must make use of higher-density devices; FPGAs fill this role well.

Should the design then be exclusively implemented using an HDL? While this approach enables professional ASIC designers the ability to generate large, complex designs, a number of pitfalls make it a poor choice for pedagogical purposes as shown in Figure 2. In this figure, note that there is no visual similarity between this design artifact and the design process shown in Figure 1, except for the two `assign` statements. Subtleties that a student may not notice or may code incorrectly: (a) use of a synchronous reset, a better choice for many FPGAs; (b) The initial state (OFF), chosen when in reset; (c) the use of the non-blocking assignment operator <= when inferring a D flip-flop, in contrast to the use of the blocking assignment operator = in an `always` statement. In addition, the complexity of the language provides many pitfalls into which students may stumble – inferring a latch instead of a flip-flop, forgetting to a reset, or the many cryptic syntax errors which HDLs invariably produce. While knowledge of HDLs is essential, its use should be postponed until students have acquired the foundational skills presented in an introductory digital design course.

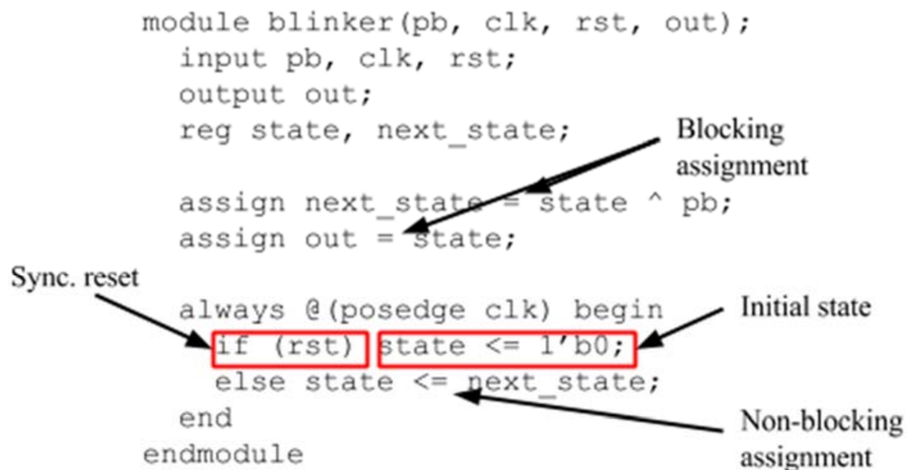


Figure 2: Verilog implementation of the design in Figure 1.

As a third alternative, should the design therefore be implemented using schematic capture? While this is feasible for small designs, schematics lose their expressive power and require significant time spent drawing and redrawing the diagram when too many gates are used, as shown in Figure 3.

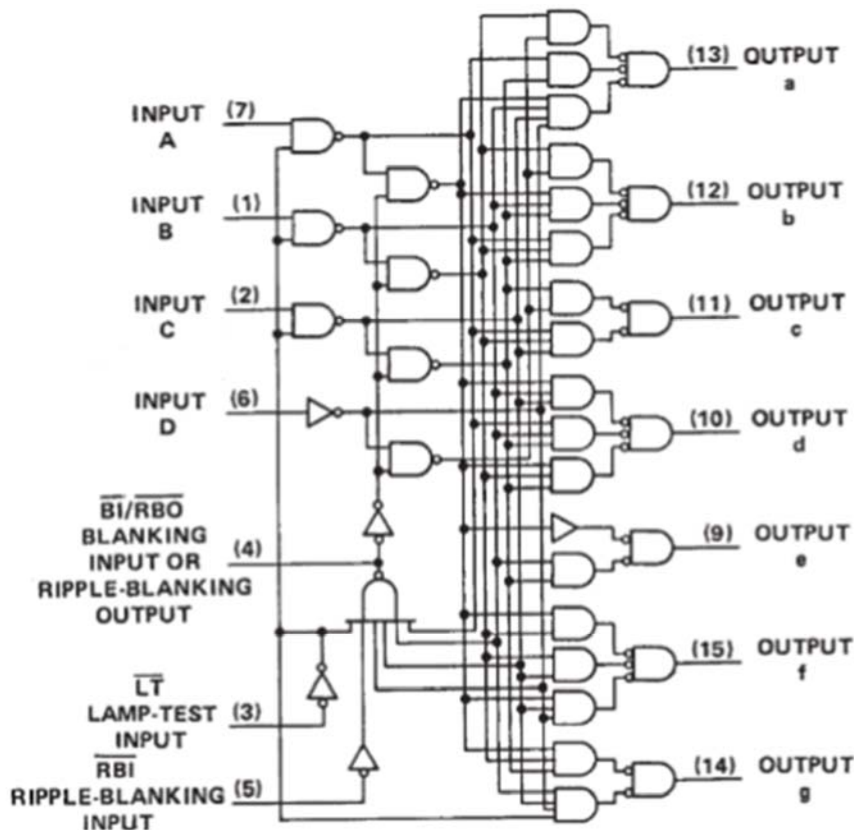


Figure 3: The schematic of a TI SN7448 BCD to seven-segment decoder/driver shows that schematics lose their expressive power as the number of gates grows.

Therefore, in order to both visually match the classroom content shown in Figure 1 while also supporting larger designs (see Figure 5), consider the hierarchical implementation shown in Figure 4. Note the strong visual similarity to elements of Figure 1 in both the block diagram (the top-level schematic) and the combinatorial equations (captured as Verilog in the `comb_blinker` module). Here, the combinational block contains a simple Verilog file containing only continuous `assign` statements, very much recalling the simplicity of ABEL. Rather than infer D flip-flops for the state block, the state is composed of a flip-flop taken from the vendor's libraries (Xilinx, in this case), guaranteeing exactly what will be synthesized in the design.

2.2 Verification

With a design complete, students must have some method to verify that their design is correct. Likewise, instructors need a formal method to verify that their students' designs are correct. As essential as verification is for circuit design, the wide variety of methods and varying language support for verification (such as the inclusion of assertions) as discussed in the background section makes this a non-trivial task, one that is often neglected. Therefore, consider the following common approaches to verification:

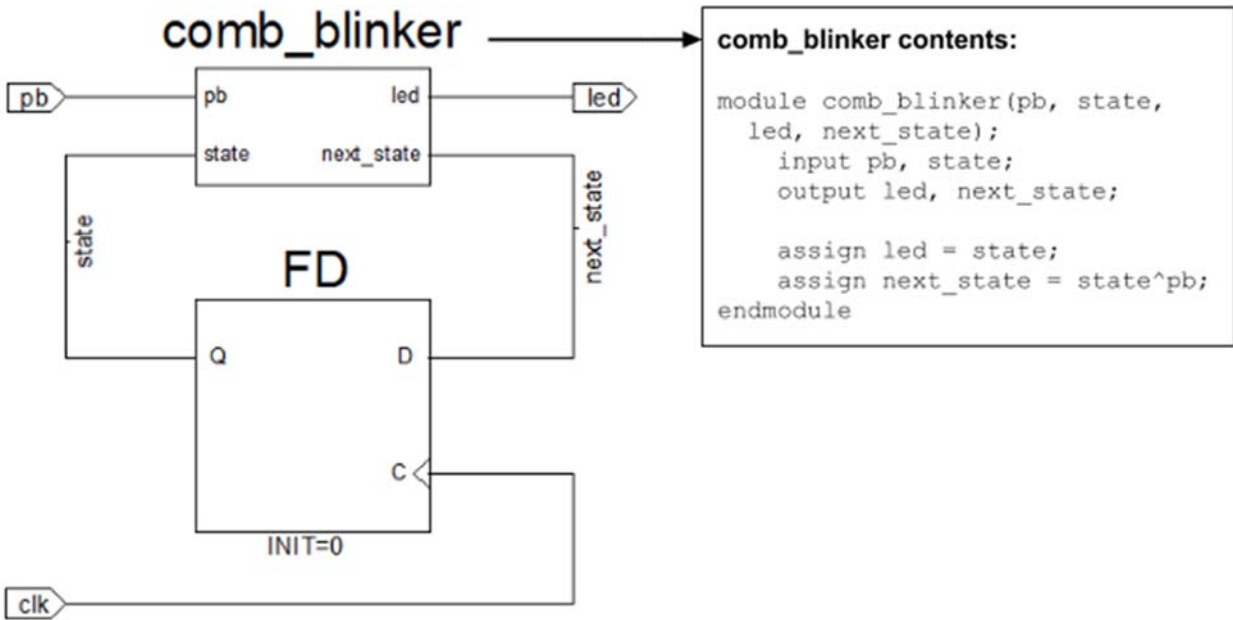


Figure 4: The implemented blinker design of Figure 1.

Manual: The most straightforward method of verifying a design is to manually test it using pushbuttons or switches as input coupled with LEDs for output on a physical board. For example, one current textbook omits formal testing from the laboratory experiments, instead requiring students to manually test their resulting design.[14] While this is certainly a valuable exercise and students should always demonstrate that their simulations do produce a correct real-world design, this approach is error-prone and typically involves a very small amount of testing. With this approach, when students find errors, they have no insight into where to begin debugging. Assertions, as discussed below, provide this valuable starting point.

Stimulus-only testbench: many common approaches simply provide a set of carefully selected inputs which exercises the device under test (DUT), such as the FSM given in Figure 1, using any particularly error-prone sequences. The user must then manually inspect the waveforms resulting from the simulation to verify that the outputs are indeed correct. As an example, Hwang's text[15] provides screen shots of simulation traces, but does not provide an automated method to check that these results are correct. Again, while simulation is helpful, this approach suffers from most of the problems with the manual methodology discussed previously.

Individual assertion testbench: Much more helpful is the approach given in Wakerly[16] and Vahid.[17] A series of `assert` statements (in VHDL or SystemC) or a series of `if (!assertion) $display` constructs (in Verilog) provide an automated method to check the correctness of a design by applying a stimulus then verifying that the resulting output is correct. However, this approach still requires a manual process: scrolling through the output to check for any failure messages. For small designs, this works well; for large designs, where there may be hundreds of tests

(as in Figure 5) and much additional textual output not related to test failures, this again becomes tedious, time-consuming, and error-prone.

Unit testing: To improve verification, consider test-driven development (TDD) methodology in the domain of software engineering, in which use of a TDD approach demonstrates improved productivity.[18] In particular, the xUnit testing framework central to TDD has been successfully implemented on many programming languages. These frameworks run a series of tests then report on the results. This last step is essential: it provides a quick summary, prominently indicating if all tests passed or if any failed. While SVUnit[19] provides a xUnit implementation for SystemVerilog, this paper presents a much simpler framework which runs on Verilog, since some toolchains (Xilinx ISE, which were required for the FPGA used in the digital devices laboratory accompanying this paper) don't support SystemVerilog. This framework, detailed in the following section, provides assertions, reports the line number on which any failures occurred, and provides a simple report with an all test passed indication or a failure message indicating the number of failed tests.

2.3 Datapath design

The approach presented in this paper scales well to larger designs. For example, a typical datapath design consists of a FSM to control datapath components (counters, adders, etc.); the block diagram of such a design can be directly represented in schematic capture by placing the required datapath components from a vendor's library, supplemented with the FSM architecture shown in the previous sections. The verification methodology likewise scales well, again presenting a simple go/no go status then providing insight into what tests failed, and therefore what areas of the design must be examined.

As an example, consider Figure 5, which challenges students to design a queue, based on information presented in section 5.8 of Vahid's excellent text.[20] This laboratory exercise challenges students to design a 4-bit, 16-entry queue. First, in (1) schematic capture allows students to employ complex library components (in this case, a dual-port RAM, counters, and a comparator) with minimal effort. Second, the detail of the controller block's design in (2) shows the layout of a simple FSM: D flip-flops are taken from the library, while combinational logic is implemented using Verilog's continuous assignment statements in (3). Finally, the test bench shown in the figure is provided to students, enabling them to verify the correctness of their design.

3. Implementation

Notice the progression of concepts in this diagram: first, combinational logic is introduced. Coupling this with D flip-flops then enables FSM design. Next, the use of datapath components as library symbols introduces students to RTL design. Finally, coupling that datapath with an FSM provides a full RTL-centric design experience, enabling students to create complex designs with simple tools. This flow was mirrored in the design of the laboratory for this class; see https://sites.google.com/site/ece3714digitaldevices/ for the lab experiments referred to in this section.

Introduction: In introductory labs students first familiarize themselves with the Xilinx ISE toolchain by creating a two-gate combinational circuit, simulating it, then programming it on an FPGA (the Digilent Basys 2 board).

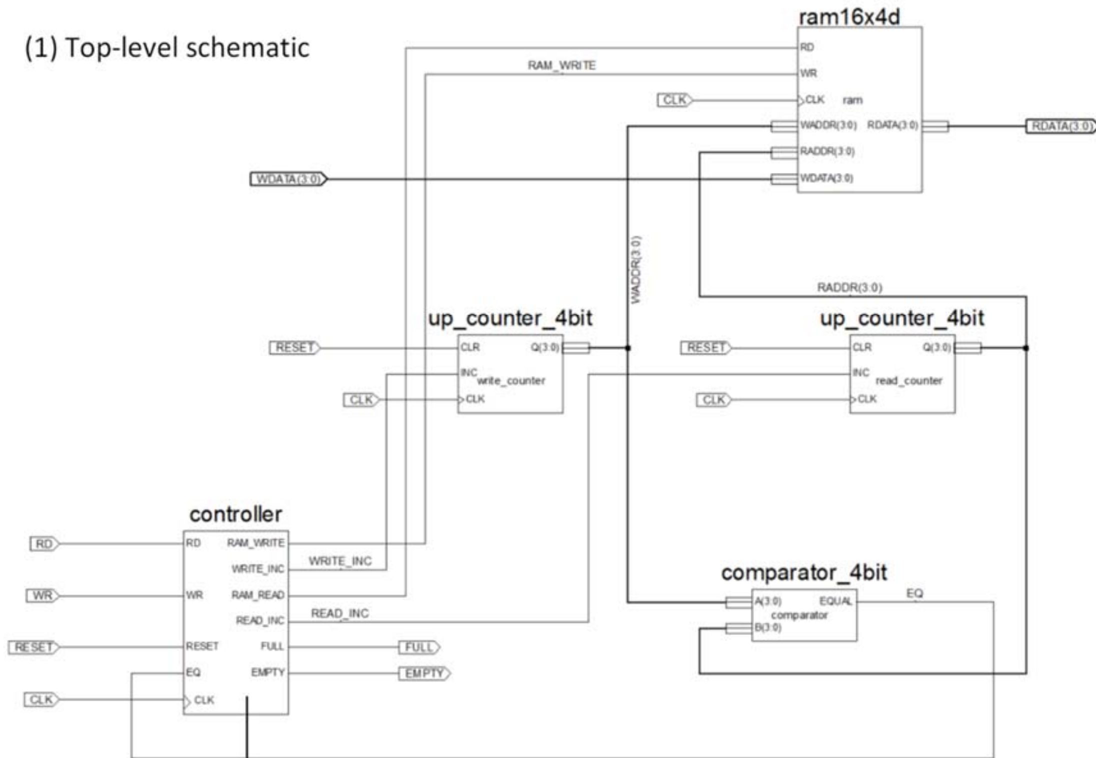
Combinational logic: In Lab 3, students design a 7-segment decoder by entering a series of assign statements in a Verilog source file; the top-level block diagram provided in the lab connects the decoder to switches as input and a 7-segment display as output. A test bench verifies correct operation; the design is then loaded to an FPGA for students to demonstrate to their TAs. Lab 4 introduces hierarchical design by using two seven-segment decoders with a mux to display digits in a multiple 7-segment display connected in a common-anode configuration. Hierarchical designs will then be used throughout the remainder of the labs. As always, designs must first pass a test bench in the simulator before being programmed to the FPGA.

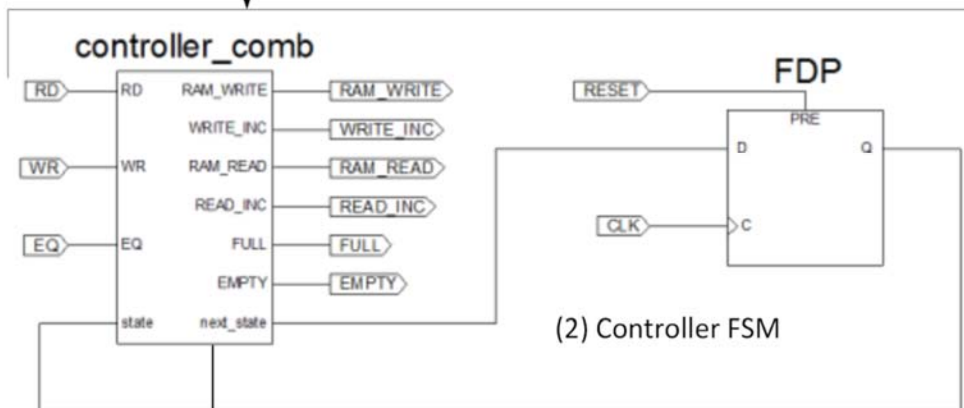
FSM design: Lab 6 challenges students to implement two small state machines, following the pattern given in this paper. Specifically, D flip-flops are placed using the schematic capture tool, then connected to combinational logic defined by Verilog `assert` statements to implement the FSM's truth table. Lab 7 requires the design of a door lock, a more complex state machine, using the same pattern. Test benches in simulation help students correct their implementation before verifying operation on an FPGA.

Datapath design: Labs 8 and 9 require the use of datapath components (adders, muxes, etc.) in creating an adder/subtractor and an ALU to familiarize students with these components and how to incorporate them into a design. Next, Lab 10 takes students through a simple datapath design by connecting an FSM with datapath components to create a simple stopwatch. Finally, Lab 11 (see Figure 5) requires students to design a queue, again by connecting datapath components with an FSM. Again, test bench simulation and implementation on an FPGA assist students in the design process.

These changes, implemented in the Fall 2013 semester, support the following stated course objectives:

- To develop the students' ability to analyze, design and build using combinational logic design, sequential logic design (controllers), datapath components, and register transfer level (RTL) design.
- Students should be able to demonstrate proficiency and comprehension in basic schematic design and HDL methodologies.
- Students should be able to develop original circuits and program an FPGA.






```
assign FULL = EQ&~state;
assign EMPTY = EQ&state;
assign RAM_WRITE = WR&~FULL;
assign WRITE_INC = RAM_WRITE;
assign RAM_READ = RD&~EMPTY;
assign READ_INC = RAM_READ;
assign next_state = READ_INC |
  state&~READ_INC&~WRITE_INC;
```

Test bench

```
// A read when empty does nothing.
test_name = "Read when empty";
`RESET_DUT;
RD = 1;
`TEST_RUN_RESET;
`ASSERT(FULL == 0);
`ASSERT(EMPTY == 1);
```

(3) Conbinational logic (Verilog)

Figure 5: A queue, which shows the implementation of a complex datapath design containing a FSM controller which directs the operation of datapath components (a dual-port RAM, counters, comparators).

4. Results and conclusions

In conclusion, this straightforward approach of implementing Boolean algebra (combinational logic) using a Verilog assign statement, then employing simple hierarchical design techniques and making use of library blocks for datapath components provides instructors with a flexible toolkit, helping students master the complexity of modern digital design with the simplicity of a few carefully-chosen tools.

Past versions of the laboratory component for this class relied on 74x logic ICs; however, this approach prevented the use of more advanced lab assignments, such as that shown in Figure 5. This approach was discarded in favor of an HDL-only approach. However, students found the use of HDLs too abstract; they were unable to correlate HDL code with in-class material. Labs soon became templates in which students only filled in combinational assignments. Compared with these previous revisions, anecdotal evidence collected based on the approach discussed in this paper shows that providing schematic capture for the top level design and memory or feedback-based components of the design allowed deeper coverage in both lab and lecture of FSM, and opened the way to challenging students with datapath designs at the end of the semester. By utilizing continuous assignment statements that can be mapped directly to the combinational logic, students can see a small but understandable portion of an HDL without being completely overwhelmed by the abstraction HDLs allow. Specifically, by allowing schematic capture to be used for the flip-flops and memory or feedback portions of the design, students can focus on how these components provide the desired functionality in the overall design without becoming lost with the abstractions of the HDLs.

References

[1] See http://www.xilinx.com/support/answers/32354.html, http://www.altera.com/support/kdb/solutions/rd12082003_6734.html. Retrieved 29-Jan-2015.

[2] “IEEE Standard Hardware Description Language Based on the Verilog(R) Hardware Description Language," *IEEE Std 1364-1995*, pp.1,688, Oct. 14 1996.

[3]

[4] “IEEE Standard Verilog Hardware Description Language," *IEEE Std 1364-2001*, pp. 1,856, 2001.

[5] “IEEE Standard for Verilog Hardware Description Language," *IEEE Std 1364-2005* (Revision of IEEE Std 1364-2001), pp.1,560, 2006.

[6] “IEC Standard for Systemverilog - Unified Hardware Design, Specification, and Verification Language (Adoption of IEEE Std 1800-2005)," *IEC 62530:2007 (E)*, pp.1,668, 2007.

[7] “IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language," *IEEE STD 1800-2009*, pp. 1285, Dec.11, 2009.

[8] “IEEE Standard for SystemVerilog–Unified Hardware Design, Specification, and Verification Language," *IEEE Std 1800-2012 (Revision of IEEE Std 1800-2009)*, pp.1315, Feb. 21, 2013.

[9] Vahid, F., “Digital Design with RTL Design, VHDL, and Verilog”, 2nd ed., Wiley, 592 p., 2010.

[10] Wakerly, J., “Digital Design: Principles and Practices,” 4th ed., Pearson Education, 895 p., 2005.

[11] Mani, M. Morris and Ciletti, Michael D., “Digital design with an introduction to the Verilog HDL,” Pearson, 576 p., 5th ed., Jan. 2012.
[12] Newstetter, W.C., & Svinicki, M.D. (2014). Learning theories for engineering education practice. In A. Johri & B. M. Olds (Eds.), Cambridge Handbook of Engineering Education Research (pp. 29-45), New York: Cambridge University Press.
[13] Prince, M. J. and Felder, R. M. (2006), Inductive Teaching and Learning Methods: Definitions, Comparisons, and Research Bases. Journal of Engineering Education, 95: 123–138.
[14] Mani, M. Morris and Ciletti, Michael D., “Digital design with an introduction to the Verilog HDL,” Pearson, 576 p., 5th ed., Jan. 2012.
[15] Hwang, E. O., “Digital Logic and Microprocessor Design with VHDL,” Thompson, 608 p., 2006.
[16] Wakerly, J., “Digital Design: Principles and Practices,” 4th ed., Pearson Education, 895 p., 2005.
[17] Vahid, F., “Digital Design with RTL Design, VHDL, and Verilog”, 2nd ed., Wiley, 592 p., 2010.
[18] Erdogmus, Hakan; Morisio, Torchiano. "On the Effectiveness of Test-first Approach to Programming." *Proceedings of the IEEE Transactions on Software Engineering*, 31(1). January 2005.
[19] http://www.agilesoc.com/open-source-projects/svunit/, retrieved 31-Jan-2015.
[20] Vahid, F., “Digital Design with RTL Design, VHDL, and Verilog”, 2nd ed., Wiley, 592 p., 2010.