VisiBoole: VISIBLE DIGITAL LOGIC EDUCATION

John J. Devore
Department of Electrical and Computer Engineering
Kansas State University
jdevore@ksu.edu

Abstract

A novel software tool, called VisiBoole, provides a color-coded interactive display of simulated Boolean values. The Boolean values being displayed are named Boolean variables. They include both independent (input) and dependent (output) variables. Dependent variables are defined via Boolean equations. Their values are dynamically calculated from their defining equation based on the current values of the variables they are dependent on. Thus, VisiBoole provides a simulation and visualization for what amounts to an extremely simple hardware description language (HDL). The major aspects of that simplification is supporting only two-valued variables and two types of statements. Because of the two-value restriction, variables do not need to be declared – their role is determined by the context of their use. The two statement types are a variable-list statement and a Boolean assignment statement. Circuits expressed in assignment statements may be combinational or sequential in nature. Sequential circuits are created whenever a flip-flop is implied in the assignment. This is specified by appending a .d suffix to the output variable name on the left hand side of the assignment statement. This turns the variable into a state variable and makes it an independent variable – only its D input is dependent on other variables. Inputs (independent variables except state variables) must appear at least once in a variable-list statement. Both independent and dependent variables may appear as often in any variable-list statement and in as many such statements as is desired. Statements may appear in any order without affecting the results. This contributes to the simplicity of using the tool, and is also a teaching point. This point being that a set of HDL equations is different from a set of software language assignment statements. The HDL equations are evaluated in parallel whereas the software set are evaluated in sequence – top to bottom. VisiBoole has two modes – edit and run. They exist as tabs on the main display. Designs are created in edit mode and exercised (tested) in run mode.

The name, VisiBoole, was intentionally patterned after VisiCalc to suggest its spreadsheet-like characteristic. In **run** mode, VisiBoole always displays the current value of each variable (including their occurrences in the right hand side of equations) in color. Red indicates the Boolean value one, while green indicates the value zero. Input variables (variables that never appear on the left-hand side of an equation) and state variables (D-flip-flop names) can be toggled (interactively) by clicking on any occurrence of them on the display. After any such click, the program recalculates (simulates) the value of each dependent combinational variable

Proceedings of the 2014 ASEE Gulf-Southwest Conference
Organized by Tulane University, New Orleans, Louisiana

and next state value of each registered variable to update its color on the display. Thus, the VisiBoole program provides a spreadsheet-like interaction between the user and the displayed equations. As well as having the ability to toggle any input or current state value, there is also a TICK button that when clicked simulates the action of a clock in a sequential circuit. It causes all registered variables to take on their next state. Again this change of values propagates throughout the set of equations.

A prototype program was developed and a trial use begun in a senior computer-engineering design course. The favorable reaction of students convinced us that we should expand our use of this tool and to develop it further for distribution throughout the educational community. We will provide this program free of charge to interested educators. It is important for anyone reading this paper to be using a color copy. It will be very difficult to properly comprehend the software without seeing the included figures in color. Our apologies to any reader that has red-green color vision difficulties. We plan to provide the ability to configure colors in the described tool and even to substitute other font characteristics for color to allow use by totally color-blind users. However, our proof-of-concept program which was used to produce the figures in this document is only capable of using a red-green color-coded display.

Motivation for Creating VisiBoole

Instructors of digital-logic continually strive to find ways to help each student experience a digital-logic "eureka" moment, preferably early in their studies. Techniques taught to beginning students are straightforward. They include creating a truth table for a given function (combinational or sequential), transforming it into a sum-of-minterms or product-of-maxterms form, and using a Karnaugh map to produce a simplified sum-of-products or product-of-sums Boolean algebra equation[1]. However, even students that learn these techniques easily often have difficulty developing an in-depth understanding of the results they produce. This lack of in-depth understanding makes it difficult for even good students with merely operational understanding of Boolean logic to become fluent in hardware description languages. They still struggle with creating sets of Boolean equations to describe complex digital systems. Less capable students who find it difficult to perform even the basic operations usually never develop any HDL skill, and therefore choose to pursue an area of specialization not involving digital design. The authors believe that it is the parallel nature of multiple-output digital logic systems (even sequential systems that involve many flip-flops where the next state of each is produced in parallel) that causes much of the difficulty. The software tool, described herein, provides a mechanism designed to greatly enhance understanding of Boolean algebra equations, especially a related set of equations. These equations can be for either combinational or sequential circuits (nonregistered or registered variables).

Boolean algebra is a topic in many fields of study and educational disciplines. VisiBoole could

be a tremendous aid in courses that include the topic of Boolean algebra or of binary systems in general. Most digital logic simulations run from a script providing a set of inputs and their corresponding expected outputs. These handle small to very large designs, but offer virtually no insight into the logic being tested. Existing "teaching" simulators animate a circuit diagram (using logic gate symbols and connecting wires) of the design. These diagrams are time-consuming to create, and often become hard to follow because it is difficult to avoid some haphazard placement of components or complex routing of connections except on the simplest of designs. VisiBoole can easily display designs that are much more than an order of magnitude more complex than can circuit-diagram-based simulators without the set becoming difficult to comprehend.

Overview of VisiBoole Software

This tool helps reveal the inner working of Boolean logic designs similar to the way the Visible Body program helps reveal the inner working of the human body. VisiBoole displays the current value of each variable in every equation as a color. Red indicates a value of one, while green indicates a value of zero. One can also think of red as True and green as False. The program then calculates (simulates) the value of each dependent variable to display via its color as well. The program provides a spreadsheet-like interaction between the user and the displayed equations. Each independent or state variable can be "clicked" to toggle its value and that change will cause a reevaluation of all the dependent variables and next-state values. There is also a TICK button that simulates the action of a clock in a sequential circuit. It causes all registered (state) variables to take on their next state. Again this change of values propagates throughout the set of equations. A prototype program has been developed and a trial use begun in a senior-level design course. The success of that experimental use and the favorable reaction of other educators that have been shown the program (including several suggestions for enhancements) indicate that this tool deserves development and distribution throughout the educational community. Boolean algebra is a topic in many fields of study and educational disciplines. The tool described herein, after proper development, could be a tremendous aid in courses that include the topic of Boolean algebra or of binary systems in general. This could be true over a great range of courses and educational levels. Examples can be created that could be useful in teaching binary number concepts to middle school students. The examples presented are more appropriate to various levels of computer engineering courses.

It should be emphasized that there are no existing tools (programs) that take this approach to simulation and display or that can come even close to displaying on a single screen the easy-to-follow complete working details of very complex digital systems. Several existing tools perform digital simulation, but cannot help the student visualize the operation of very complex systems. They include VHDL simulators, NI MultiSim, and a handful of logic-symbol-based visual simulators where the interconnecting wires are color coded to show current logic values. VHDL

simulations are test-vector based and show nothing about why the outputs are the value that they are. NI MultiSim and other schematic-based simulators require tedious circuit entry and the circuit must incorporate switches connected to Vcc and ground for inputs. MultiSim requires placing probes in the diagram everywhere that outputs or intermediate values are to be monitored. There are three big deficiencies of these systems when the goal is to help students develop in-depth understanding of digital systems so they can advance to being able to produce significant digital designs. First, schematic entry is slow and tedious. One must select components, place them, and connect them with wires. Second, a single screen schematic can not show a very complex system in detail. Finally, they do not provide a direct path to HDLs that can allow the student to reproduce a complex design on an actual chip such as an FPGA so that she can watch the design run on hardware after the bugs have been eliminated during simulation.

Devore and Hardin used an active display of control unit equations for teaching computer hardware concepts[2]. It differed from VisiBoole in two very important ways. The program they used incorporated a fixed set of equations – it showed a single specific design. There was no way to change any of those equations let alone create a whole new set without modifying their program. Also, that program did not provide a way to make arbitrary changes to the values of any of the Boolean variables in the equations. Students could only view them going through a preset sequence of values. It was a pre-Windows (DOS) application running on a computer with a monochrome display without a mouse. The binary values of variables were shown by using black or white background (normal or inverse characters). Nevertheless, it was reported that student inspection of the sequence of displays of the set of equations that the program produced did aid student understanding. We site such an old article only to emphasize that it is the only article we could find that reports any functionality even remotely similar to the VisiBoole program.

The VisiBoole software provides an interactive display of a set of input variables and a set of Boolean algebra equations specifying intermediate variables, outputs variables, or the next state of state variables. Each equation can be for a combinational logic or sequential logic variable. The program currently runs on a Windows-based PC. VisiBoole provides a spreadsheet-like approach to the simulation of, and thus provides a visual display of the functioning of those Boolean algebra equations. The input files to the prototype program consist of statements that are either a list of input variables or standard-looking equation-based hardware design language (HDL) statements. The authors of this paper believe that the equation-based approach for an HDL underscores the parallel nature of hardware implementations, and that a comprehension of this parallelism is fundamental to true understanding of digital systems. Design languages like Verilog[3-4] offer an ease of implementation, and potentially fewer flaws in logic during the debugging phase, than do more purely equation-based languages[5]. However, in our opinion, they hide so much of the parallel nature of hardware that they are poor choices for a first exposure to

HDLs. VisiBoole animates a collection of HDL equations. In effect, every variable name becomes a cell in an active evaluation matrix. Font color is used to show the current binary value of each variable. Currently **red** indicates a value of one (or True or active) and **green** indicates a value of zero (or False or inactive). Currently only sum-of-product expressions are supported. AND operations are implied – a product term is simply a horizontal list of variables separated by blanks. A plus sign is used as the OR operator. AND has precedence over OR so parentheses are not required (nor allowed). An over-bar is used to show a complemented literal in **run** mode. Only single variables can be complemented. These conventions and restrictions all contribute to an uncluttered display that makes it easy to follow, and thus understand, the logic of the equations. Fixed-width fonts on the display allow successive lines of variables to be aligned in meaningful ways. Use of over-lining shows which literals are complemented without disturbing alignment. The display is dynamic in the sense that any occurrence of any input variable or state variable can be clicked (via a mouse-style or touch-pad input device) at any time anywhere on the display to toggle its value. This change is first propagated to all occurrences of that variable then to all variables dependent on it. An inspection of the display for any given set of input values and state variable values can reveal exactly why each dependent variable and next state-bit value is **one** (**true**) or **zero** (**false**). Since each design equation is a SOP equation its value will be **true** only if at least one product term is **true**. A product term is **true** only if all of the literals it contains are **true**. The color-coded value make it very easy to determine which product terms are **true**, and for those that are **false** to determine exactly why they are **false** (i.e. which literal(s) is(are) **false**). Performing this inspection for all combinations of inputs for simple sets of equations, or an interesting subset of those combinations for complex sets, can lead to an in-depth understanding of any specific design and of HDL described designs in general. As stated earlier, both combinational and sequential logic are supported in VisiBoole. Combinational assignment equations have only a variable name to the left of an equal sign. Sequential logic assignments use a variable name with a ".d" sufix. When formatted for the display, the variable name and the ".d" are separated by a space. The variable is colored with the "current" value of that state variable, while the ".d" is colored to reflect its "next" value – the value it will be assigned when a clock TICK occurs. The prototype software supports only D-type flipflops, but other types would be easy to add. A single TICK button provides the clock to all registered variables. The updating of registered values because of a TICK is followed by those new values propagating throughout the set of equations. The behavior of sequential circuits can be viewed by repeatedly clicking the TICK button. Thus, a better or faster understanding of such circuits can be achieved by inspecting the changes that occur and that are about to occur after each successive TICK. In addition to a list of input variables and design equations, formatted fields can be created to show groups of Boolean variables formatted as binary, decimal, or hexadecimal values. Such fields on the display are useful for demonstrating unsigned and 2's complement binary number. They are also useful for interpreting the inputs and results of arithmetic circuits and for showing state-machine state values. Additionally, if such a field is comprised only of input and state variables one can click it (instead of individual

variable names) to increment the field value. This provides a very easy (and fast) way of trying all possible values of a set of inputs.

The formatting field feature can also be used to help explain binary number interpretations as is shown in Fig. 0 below. Students running that file can click on the various bits of ***a*** and see how it affects the numeric value interpretation of that set of bits.

The sample screen-shot shown in Fig. 1 helps give a better feel for the use of the display. It implements an 8-bit ripple-carry adder. The circuit it represents contains 59 (2- or 3-input) AND gates and 15 (2-, 3-, or 4-input) OR gates. Entering this design as a detailed schematic would have probably required at least 30-60 minutes work and it would have been difficult to avoid confusion in the placement of all the gates and the routing of all the wires. Further, the schematic would have to contain 16 switches to provide inputs and 16 probes to monitor only the sum and carry values. It would be difficult to make it fit on a single sheet of paper while showing all the gates in detail. The VisiBoole design took less than 3 minutes to enter and fits easily on one quarter of a computer screen. The edit-mode file consists of 6 nonblank lines. Four are the variable-list statements that list the bits of the two 8-bit values being added, the carry bits, and the sum bits. Various formatted versions of these bit fields are also specified on those lines. The other two lines specify the expression for the carry bits and the sum bits. They are effectively vector versions of a standard full-adder circuit.

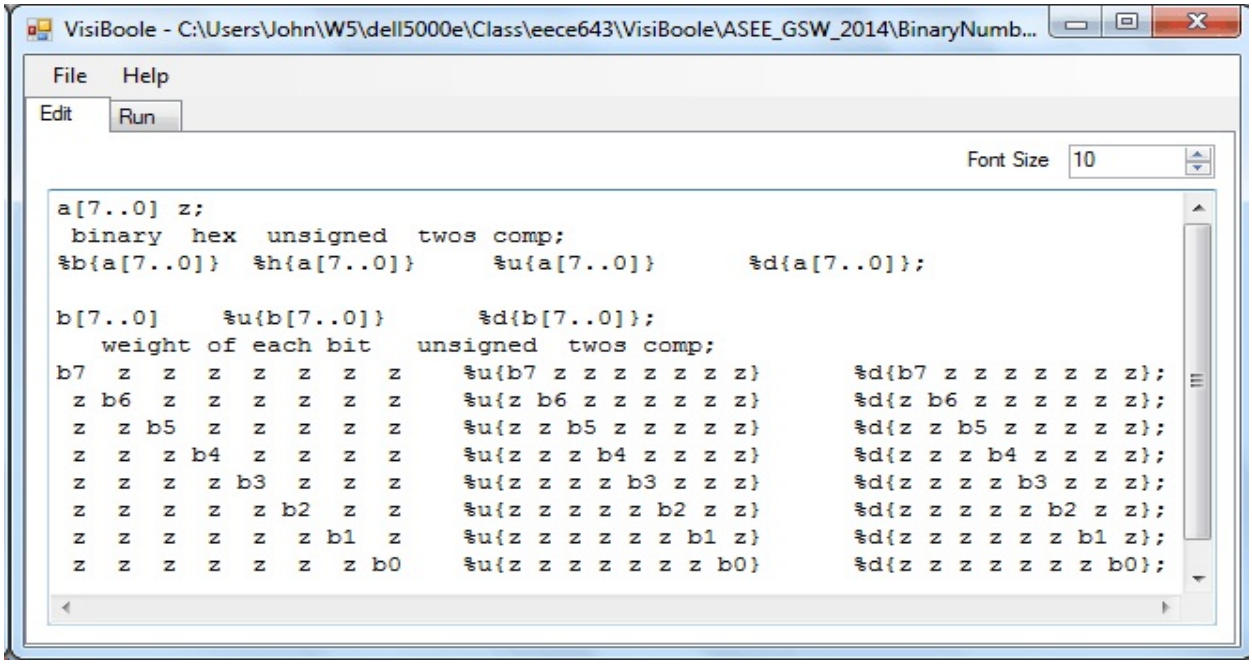

Figure 0a. VisiBoole input file used to demonstrate binary numbers.

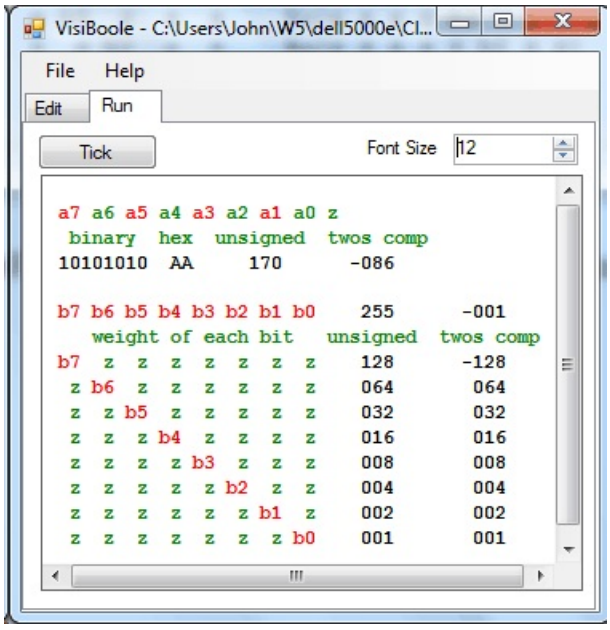

Figure 0b. VisiBoole RUN mode for above input file.

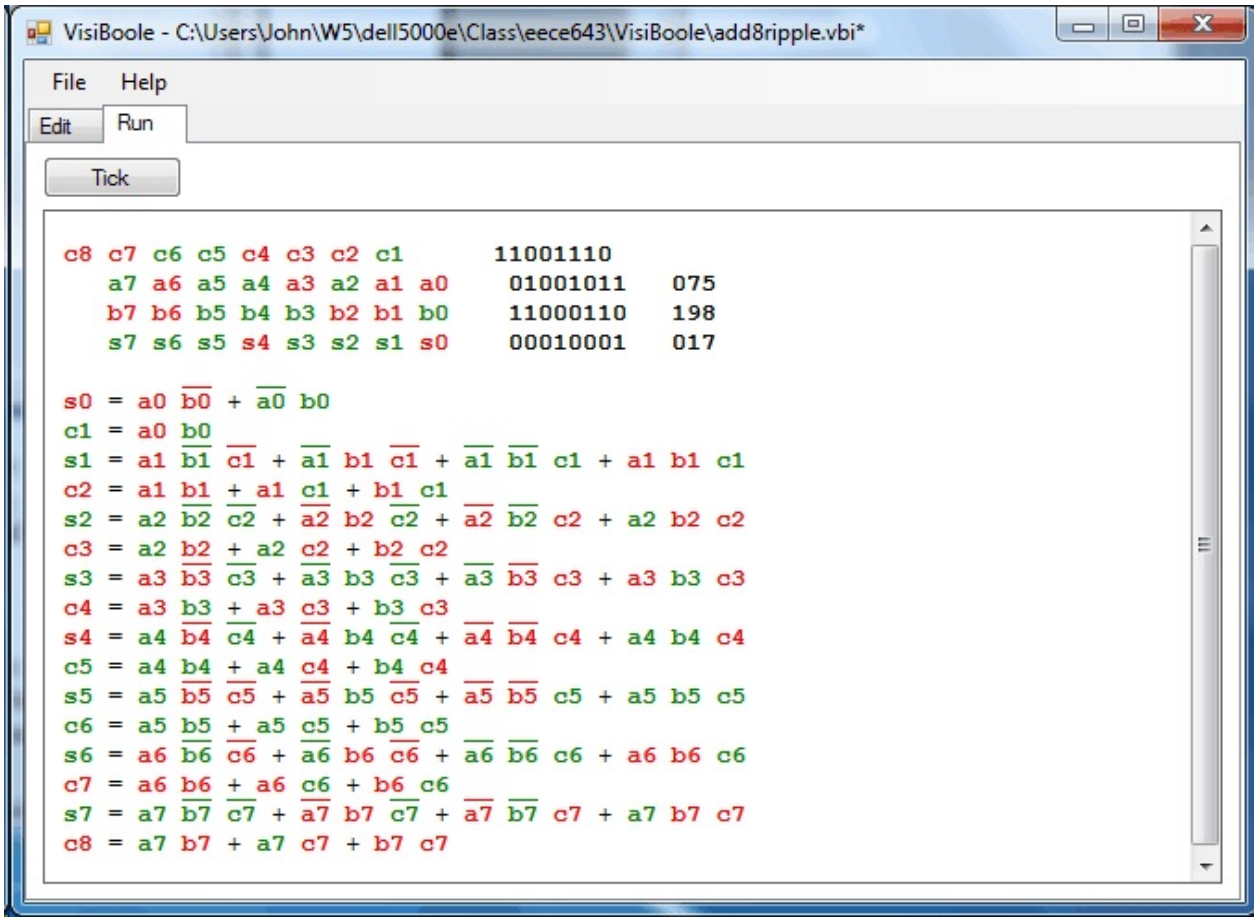

Figure 1. Sample VisiBoole Display Showing an 8-bit Ripple-Carry Adder
Red indicates a value of **1**(or true or active), **green** a **0** (or false or inactive),
the green values show the fields to their left as binary and unsigned decimal values.

Example VisiBoole Uses

Three more examples are presented and are ordered from basic to complex. The first example is appropriate for an entry-level college course involving Boolean logic. It shows a four-input function whose output truth table is to be 0 0 0 d 1 1 1 1 0 0 0 d 0 0 1 d, where d indicates a don't-care specification. The function is shown first as a full-detail sum-of-minterms expression. Then the needed minterms and maxterms are produced, by name, for showing the function as an abbreviated (use of intermediate functions – specifically the minterms and maxterms we defined) sum-of-minterms and also as a product-of-maxterms form. Finally, a simplified sum-of-products form is shown such as could be produced from a Karnaugh-map simplification. The student would be able to explore all the different combinations of inputs. Fig. 2a-d show a snap-

shot of four interesting sets of inputs. They demonstrate the fact that the shortest sum-of-minterms form always produces 0 for the don't-care case, while the shortest product-of-maxterm form always produces 1 for the don't-care case, and that a simplified expression may sometimes produce a 0 and sometimes a 1 for don't-cares. Testing all 16 values would demonstrate that all forms of the equation agree for the thirteen cases that were specified as 0's or 1's.

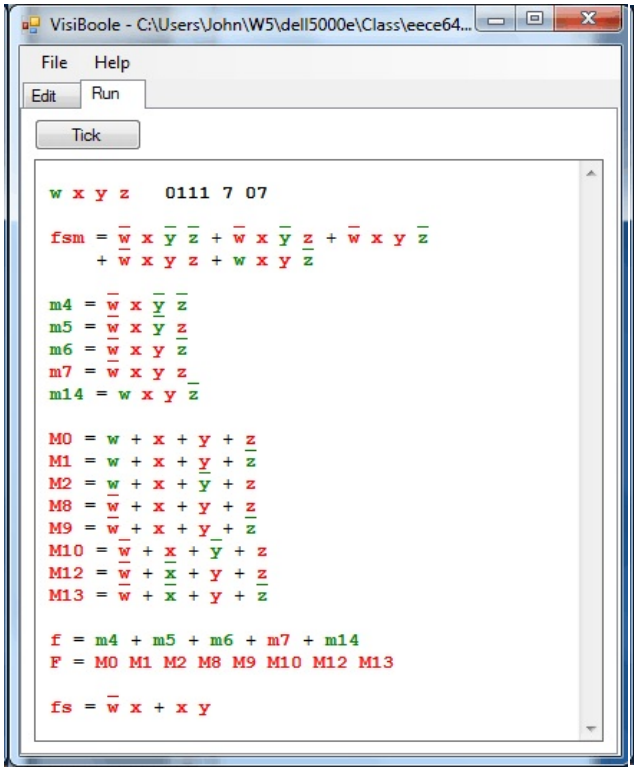

Figure 2a. F(0111)

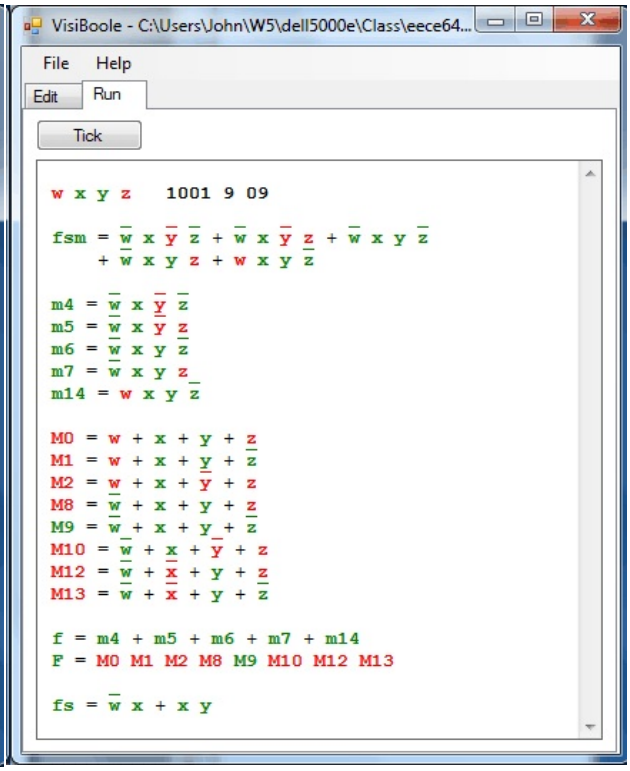

Figure 2b. F(1001)

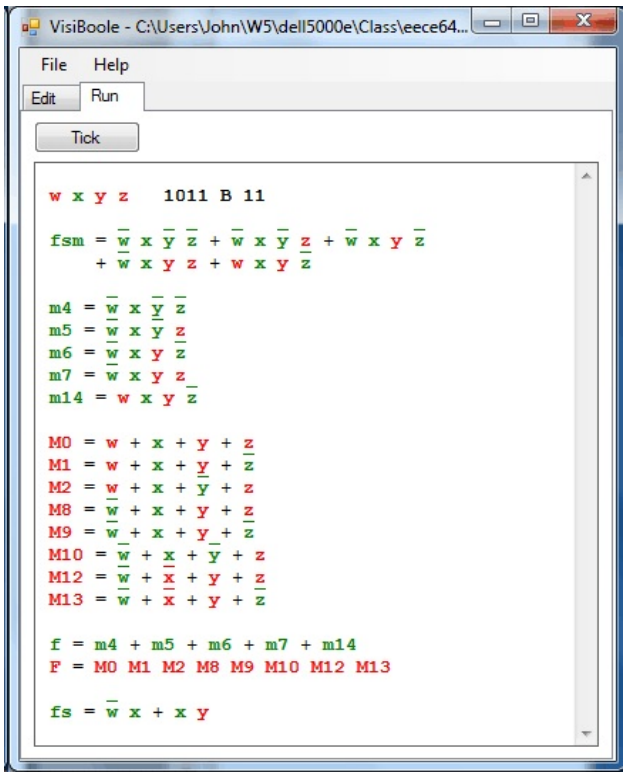

Figure 2c. F(1011)

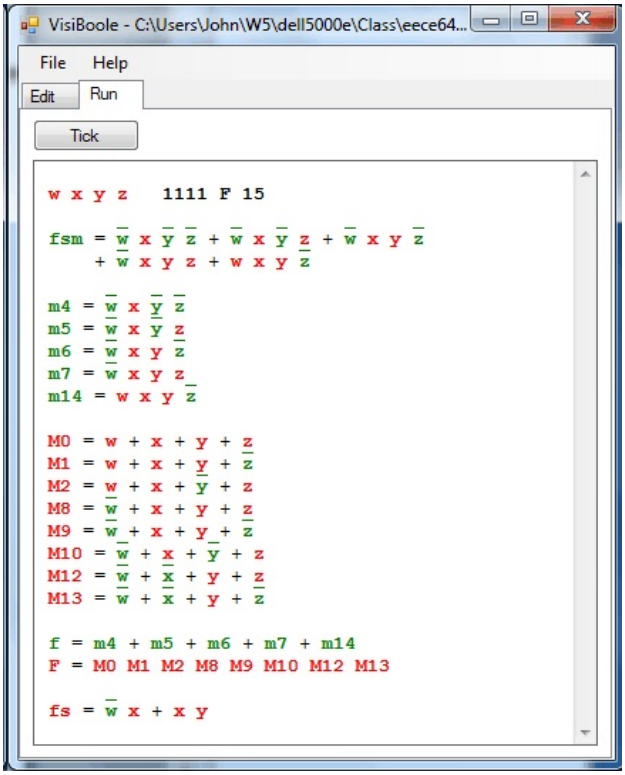

Figure 2d. F(1111)

A really important use of the VisiBoole program is to see exactly why the value of each form of the function is zero or one for each set of inputs.

The next example is appropriate for a first college course in computer engineering. It is a two-bit up-down counter. There are two inputs, up and down (the counter's state will remain unchanged if neither is true); and there are two state variables, c1 and c0. The states are decoded into s0 to s3 so that a rough state-diagram-like display can be shown by repeating those variable names in a pattern. A single panel of this display is shown. It is in state 2, and because down is active the next state will be state 1. This can be seen in the values of the .d's to the right of the state bit names. These are the values that get assigned to the state variables when the TICK button is clicked. One can analyze exactly why c1 is about to become 0 and c0 about to become 1. If from the screen-shot below (down active), the student were to repeatedly click the TICK button, the active (red) state would circle counter-clockwise in the state-diagram-like display of the states. The design specifications used in the design shown specified don't-cares for when both up and down were true. It is an interesting exercise to see how the counter behaves in that case. Unfortunately, there is not room to include displays of all the interesting aspects of using the VisiBoole program.

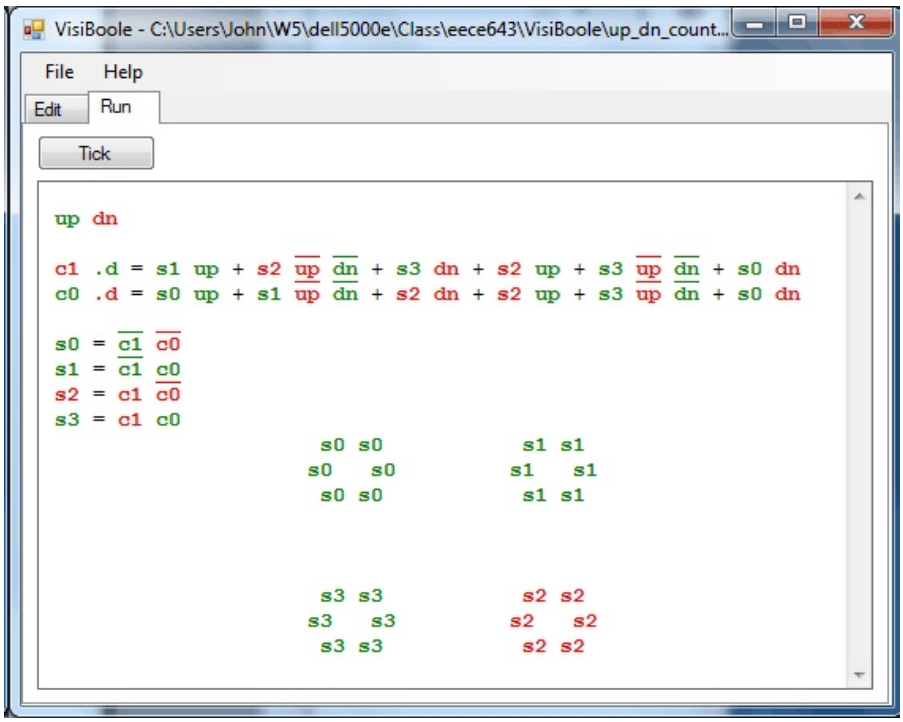

Figure 3. An Up-Down Counter with State-Diagram-Like Display

The final example is from a very basic 8-bit computer designed in ECE 643 a senior computer-engineering course. The computer utilizes an 8-bit multiplexed address/data bus and has only a program counter (PC), instruction register (IR), accumulator (Acc), and memory address register (MAR) for registers. Its memory consists of 256 bytes containing RAM, ROM, and memory-mapped I/O. It utilizes both 8-bit addresses and data. The processor can perform only ALU, Store, and Conditional Branch instructions and has inherent, immediate, direct, and indirect addressing modes. The example is the control unit (CU) for this computer. It requires a 6-state state machine to sequence the steps of the instructions. The sequence shown is executing a Store (direct addressing) instruction which requires five of those six states. By inspecting these panels, one can see how the CU is executing the following set of RTN statements in turn:

MAR <- PC++	;transfer PC via addr/data bus to MAR and post-increment the PC
IR <- M[MAR]	;read the opcode byte of the instruction into the IR
MAR <- PC++	;transfer PC via addr/data bus to MAR and post-increment the PC
MAR <- M[MAR]	;read the operand byte of the instruction into the MAR
M[MAR] <- Acc	;write the Acc to memory using the address specified in the operand

Only the section of the design that shows the next state variables and the equations for each control signal are shown to conserve space. Not shown are the input bits of the IR and how they are decoded to determine it is a store instruction using direct addressing. All variable names used should be obvious decodings of IR bits and state bits. The only one that is probably not obvious is "bct" which stands for branch condition true. It is used for the conditional branch instruction and is not involved in the example shown.

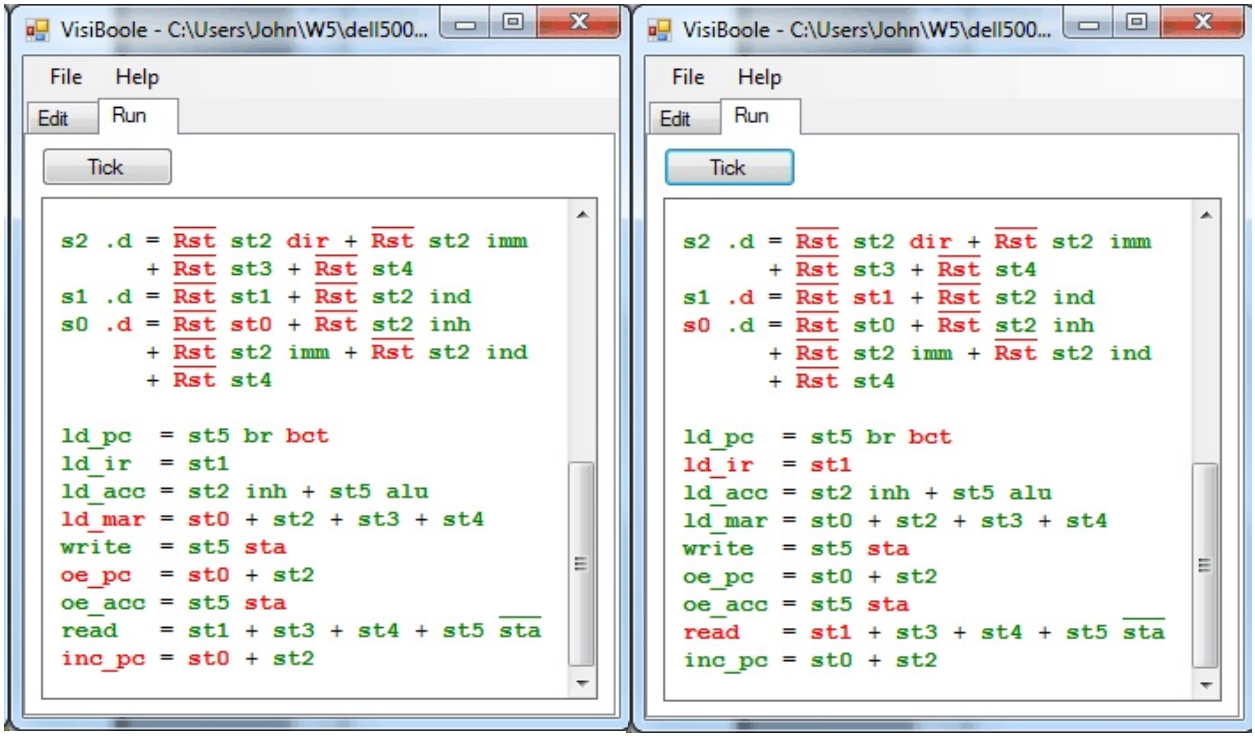

Figure 4a. st0: MAR <- PC++ :Next 1

Figure 4b. st1: IR <- M[MAR] : Next 2

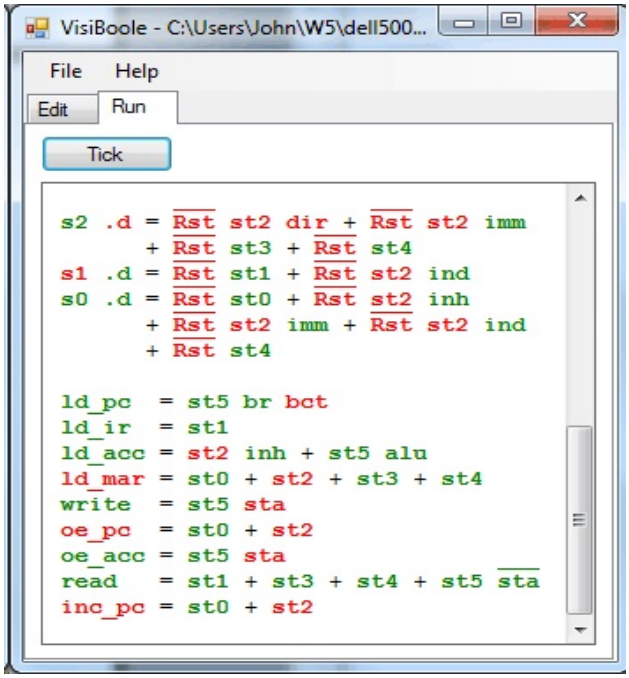

Figure 4c. st2: MAR <- PC++ :Next 4

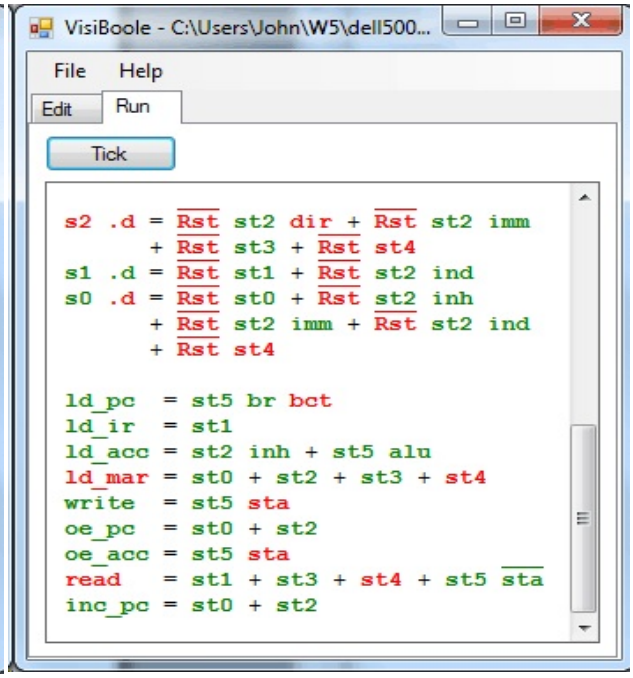

Figure 4d. st4: MAR <- M[MAR] : Next 5

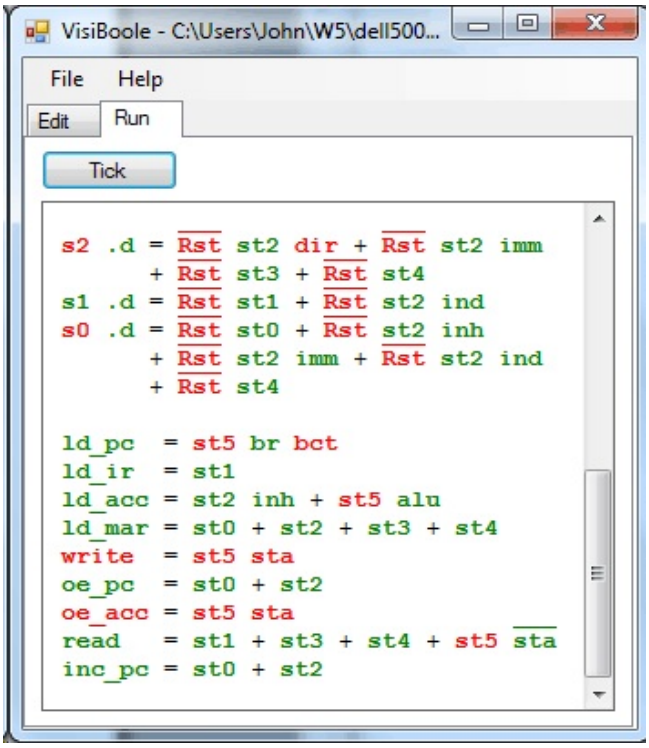

Figure 4e. st5: M[MAR] <- Acc : Next 5

Students in ECE 643 implement this computer in hardware first using a handful of 22V10 chips and later on an Altera FPGA. The control unit of the PLD version occupies only one of the chips. For the semesters since VisiBoole has been available, students have been able to create and test their CU design before converting it in to a HDL for programming. This has cut the design time by half. Further, we have been able to double the complexity of additional projects for the FPGA than in previous semesters. Every student stated that they enjoyed interacting with the VisiBoole display.

The wish list of features which will be addressed as time and funding allows include:

1. Provide subdesign capability so a hierarchy of designs could be created. A way of showing only the aspects of interest at a given time would need to be devised. One possibility for this would be in the form of function references in the Boolean expressions.
2. Support definable RAM and ROM elements to incorporate into designs.
3. Support test-vector scripts with a NEXT button that would modify input and state variables with the next vector from a test file. There should be a way to leave specified variables unchanged. Alternating between the correct set of NEXT inputs and the TICK

option could be a powerful way to investigate the operation of a given design. One might include a NEXT-ERROR button that would run the script until the simulation did not match the expected values in a test vector.

4. Create a RECORD option that will generate a test-vector script from the interactive exercising of a design.
5. Add support for asynchronous sequential circuits. At present no propagation delays are factored into the simulation of the equations.

Evaluation

The first use of this software has been in a senior design course – ECE 643, Computer Engineering Design Laboratory. The first project in ECE 643 is to implement an extremely small computer in hardware using several 22V10 chips. These simple chips have been retained to challenge the students to make sophisticated designs fit in a small amount of logic. The control unit occupies one of the chips; 8-bit registers and an ALU occupy others. Until a year ago, students created their design for these circuits in an HDL and used the simulation capability of the design software to verify its operation. This had occurred for several years without use of the VisiBoole program (which did not exist). For two years students have been introduced to VisiBoole first, and created and exercised (tested) their designs on it. Then they recreated their designs using the HDL supporting the 22V10 programmer. All ECE 643 students who have used VisiBoole have been asked to evaluate it. A surprising result was obtained. Whereas, the motivation for creating the program was to aid in student "understanding" of circuits, this received only minor mention in the evaluations. Two other aspects of the program were what received rave reviews. One was how easy the program was to learn and use. A representative pair of comments are, "The simplicity of the syntax makes learning VisiBoole quick and painless. This is nice, as being computer engineers; we are expected to learn all kinds of programming languages and syntaxes, as well as being able to use them all in a variety of situations." The other was how easy it made creating and testing designs. It was pointed out many times that the interactive nature of the program made it possible to "design and debug" incrementally. The fact that you can quickly switch back and forth between **edit** and **run** mode encouraged the testing of each equation or a small set of equations immediately after they are written. An observation by the instructor, independent of the student surveys, is that the students as a class were able to produce working designs in much less time when using VisiBoole than classes that did not use VisiBoole. Also, the class GPA for the first semester it was used (the second semester is in progress at the time of this writing) was one/third of a grade point above the long-term average. The authors suspect that a key factor in the helpful user-friendly aspects of the program is the fact that there are no declaration of variables in the language. The user can design on-the-fly in **edit** mode by making up variable names as she types in design equations. VisiBoole determines the role of each variable (input v.s. output and registered v.s. unregistered) by context.

The author suspects that the lack of specific praise for VisiBoole aiding in understanding Boolean circuits is likely the fact that these first users are mostly seniors in our computer engineering program. Students that have made it that far have probably already experienced their "eureka" moment, so VisiBoole did not have the opportunity to provide it.

We will be introducing VisiBoole into a freshman/sophomore-level course, ECE 241, Introduction to Computer Engineering, and a junior-level course, ECE 441, Design of Digital Systems. Enrollment for ECE 241 averages about 100 students a semester and is required for students in computer engineering, computer science, electrical engineering, and information systems. ECE 441 averages 25 students a semester and is required for computer engineering students and is taken as an elective by electrical engineering students. After that semester is over we will have significantly more evaluation data and wider demographics of the users. In addition, we are offering the program free of charge to the educational community. We are hoping that a use for it can be found in many courses at other schools. We would encourage the instructors of those courses to participate in the evaluation of its usefulness, and in helping us prioritize (and/or suggesting more) new features we are planning to add.

In addition to surveys, the evaluation of the impact of the introduction of VisiBoole in ECE 241 and ECE 441 will use existing data taken from final examinations as part of the ABET evaluation process to establish a base-line of performance on specific digital logic problems. This data will be compared to performance of students after the introduction of VisiBoole in the courses.

Conclusion

Most digital logic simulations run from a script providing a set of inputs and their corresponding expected outputs (test vectors). These handle small to very large designs, but offer virtually no insight into the logic being tested. Further, constructing a set of test vectors is tedious and time-consuming. Existing "teaching" simulators[6-8] animate a circuit diagram (using logic gate symbols and connecting wires) of the design. These diagrams are time-consuming to create, and often become hard to follow because it is difficult to avoid some haphazard placement of components or complex routing of connections except on the simplest of designs. VisiBoole can easily display designs that are much more than an order of magnitude more complex than can circuit-diagram-based simulators without the set becoming difficult to comprehend. This seems to be the key element of the program touted by senior computer engineering students. Furthermore, a design expressed in VisiBoole requires very little modification to convert to an HDL design to input into an HDL compiler. One of the students in our senior design course this past year wrote a conversion program that did 95% of the conversion automatically. Several students have suggested that a multi-language converter be added to the program itself.

References.

[1] M. M. Mano and M. D. Ciletti, *Digital Design*, 4th ed., Englewood Clifts: Prentice-Hall, 2007.

[2] J. J. Devore and D. S. Hardin, "A Computer Design for Introducing Hardware and Software Concepts," *IEEE Trans. Educ.*, vol. E-30, pp. 219-226, Nov. 1987.

[3] Institute of Electrical and Electronics Engineers, *IEEE Standard Verilog Hardware Description Language Reference Manual,* IEEE: Piscataway, NJ, 2001.

[4] Institute of Electrical and Electronics Engineers, *IEEE Standard VHDL Language Reference Manual*, 2000 ed. IEEE: New York, NY, 2002.

[5] http://quartushelp.altera.com/9.1/master.htm#mergedProjects/hdl/ahdl/ahdl_list_how_to.htm?GSA_pos=9&WT.oss_r=1&WT.oss=ahdl, as of 5/5/2011.

[6] http://wps.aw.com/aw_brookshear_compsci_8/18/4742/1214158.cw/content/index.html, as of 5/5/2011.

[7] http://www.teahlab.com, as of 5/5/2011.

[8] http://elm.eeng.dcu.ie/~digital1/afdez/JavaScript/Page2.htm, as of 5/5/2011.