2013 Hawaii University International Conferences
Education & Technology
Math & Engineering Technology
June 10th to June 12th
Ala Moana Hotel, Honolulu, Hawaii

VisiBoole: an Equation-Based Interactive Digital Design Tool

John J.Devore

Kansas State University

VisiBoole: an Equation-Based Interactive Digital Design Tool

Synopsis

A visual-feedback, interactive, rapid digital design and verification tool (Windows-based program) is presented. The visual feedback consists of a color-coded display of a simulation of a design's HDL equations. Design verification consists of clicking independent variables on the active display (toggles their value) or TICK (simulates a clock cycle).

Abstract

A novel software tool (Windows-based program) is helping students create complex digital logic designs much more quickly than without its use. The tool was envisioned as a way to help teach binary number systems and basic Boolean logic. It has been useful for that, however, a perhaps more important use is in digital design. Its ease of use and immediate feedback has tremendously shortened the design time for projects in a senior-level digital design course allowing more complex projects to be assigned. This presentation will provide the attendee the knowledge to use the tool, and also a free copy of the VisiBoole software. One of the great features of this tool is how easy it is to learn to use.

VisiBoole, provides an interactive display of a set of standard-looking equation-based hardware design language (HDL) statements. The name was intentionally patterned after VisiCalc (the original spreadsheet program) to suggest its unique spreadsheet-like interactive Boolean equation display. VisiBoole displays the current value of each variable in every equation of a design as a color. Currently red indicates a value of one, while green indicates a value of zero. The program continuously calculates (simulates) the value of each dependent variable which determines its color as well. These dependent variables may in turn affect other variables. The set of equations can contain any level of dependencies. The program provides a spreadsheet-like interaction between the user and the displayed equations. Updating values is simpler than in a numeric cell of a spreadsheet. As each variable has only two possible value, any variable can be "clicked" to toggle its value and that change will cause a reevaluation of all the values dependent on it. Furthermore, this "click" can occur anywhere the variable appears. There is also a "tick" command that simulates the action of a clock in a sequential circuit. It causes all registered outputs to take on their next state. Again this change of values propagates throughout the set of equations.

Background

Instructors of digital-logic continually strived 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. 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 these student to become fluent in hardware description languages. They still struggle with creating sets of Boolean equations to describe complex digital systems. Students that find it difficult to perform the basic operations usually never develop any HDL skill, and therefore choose to pursue a different area of specialization. This tool provides a view of Boolean expressions that helps develop an in-depth understanding.

VisiBoole

A prototype program (named VisiBoole) was developed and 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 (their reactions included several suggestions for enhancements) indicates that this tool deserves development and distribution throughout the educational community. VisiBoole, presented herein, provides a mechanism designed to greatly enhance understanding of Boolean algebra equations, especially a related set of equations. It also provides a rapid testbed for a digital design. Digital designs created in VisiBoole can be for either combinational or sequential circuits (nonregistered or registered variables). Use of this tool makes creating and verifying digital designs quick and intuitive. 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.

VisiBoole HDL

As VisiBoole is a simulation program for a Hardware Description Language (HDL), one must learn its HDL dialect in order to use it. The dialect is extremely simple so is easy to learn and use, yet powerful enough to create and test complex digital designs involving both combinational and sequential logic. VisiBoole designs consist of single-line statements ending in semicolons. A statement can be either a list of variables, or a single Boolean equation. The variables in a list may be either input variables, or copies of intermediate or output variables. All input variables must appear in such a list. The use of other variables in a list is merely to place a copy of their value in a more convenient location for observing during testing of the design. All types of variables can occur multiple times in the same statement and/or on multiple statements. There are two main uses for having variables listed multiple times. One is to keep a copy of inputs handy on designs that do not fit on a single screen. The other is to better display outputs or intermediate variables. Equations statements consist of a variable name followed by an equal sign followed by a sum-of-products expression. D flip-flops are used for sequential logic. Registered variables have their name followed by a .d before the equal sign indicating the equations is for the next state of the variable – the value it will take on at the next tick. Note that nothing is implied by the order of statements, just like nothing is implied by the position of a hardware component on a schematic. The author believes this is a way to emphasize the parallel nature of hardware. There is no flow of execution from top to bottom as in software.

The VisiBoole program has two modes; an **edit** mode and a **run** mode. Designs are created or modified in **edit** mode and simulated in **run** mode. In **edit** mode VisiBoole is just a simple text editor. Because of this, it is possible to create designs in some other text editor and copy them into the VisiBoole **edit** screen. There are no type declarations so the role of each variable is determined by context from the collective set of statements. If a variable never appears on the left-hand side of an assignment statement, it is an input. If it never appears on the right-hand side of an assignment, it is

an output variable. If it appears on both, it is either an intermediate variable or an output variable that is fed back into the logic. When (if) a given design is actually programmed into a device, one can tell the difference between these last two as any output would be assigned to a pin whereas an intermediate variable would not. For design and simulation only, there is no distinction between the two.

In VisiBoole the actual design (the part expressing the behavior of the digital system) consists only of assignment statements. There are no state-machine or other sequential execution constructs. This is considered a feature, not a defect. It forces the student (or other designer) to confront the parallel nature of hardware. Any sequence of activities must be handled by designing the state machine that drives the sequence. This design is embodied by writing the equations for the flip-flops that encode the state information. In teaching design this way, one can underscore the parallel nature of hardware by pointing out that all the statements of any VisiBoole design are order independent. Any working design can have its statements shuffled and it will still work the same. This can be related to the location-independent nature of components in a schematic. Any component can be placed anywhere without it affecting the design itself. The same is true of the hardware itself except that for speed considerations we keep the interconnection lengths as uniform and/or as short as possible. However, any arrangement of the components would work with a slow enough clock.

Assignment statements consist of a variable (optionally with a .d or .t suffix for sequential logic) followed by an equal sign followed by a sum-of-products expression. If something more complicated than a SOP expression of input and state variables is needed, then intermediate variables must be used. This SOP restriction allows elimination of parenthesis making the viewing of the logic easier. This is very important as VisiBoole is a visual-based simulator. Also, to make the expression less cluttered, the AND operator is implied, much as we learned to imply multiplication in algebraic expressions. The tilde (~) is used as the complement operator in **edit** mode. Thus, the expression a = b&c + (not)b&d is written as the following statement:
a = b c + ~b d;
The dependent variable above can be changed into a registered value for sequential logic. It would then be expressed as:
a.d = b c + ~b d;
Currently only D and T flip-flops are supported. Normally outside of teaching Boolean logic, only D flip-flops are used.

One additional feature is supported in the design file. The first occurrence of an input variable can have an asterisk (*) prefix. This indicates that its initial value is to be ON (1/true) instead of OFF (0/false) which is the default.

In the **edit** mode there is a RUN button. Clicking on it switches to **run** mode. In this mode the form of the statements change. Tilde operators are changed to over-bars, .d and .t suffixes are separated (by a space) from the variable names and take on next-state values separate from the variables current values. The value of each variable is color coded. Currently, red signifies a value of 1 and green a value of 0. All equations are evaluated dynamically to produce the correct value of all intermediate, output, and state variables. In RUN mode the display become interactive on a variable-value basis and no editing can be performed. The value of any input variable and any state variable can be changed by clicking on any occurrence of it on the display. The curser changes into a hand when

placed over any variable that can be changed. A state change can be caused by clicking on the TICK button, causing all registered (state) variables to take on their next-state value (always displayed by the color of the .d color to the right of each in its assignment equation or inferred by their current value and the value of the .t). The performance of a design can be tested and verified by observing the results of such interaction. Any errors that are discovered can be quickly corrected by returning to **edit** mode.

Aside from the digital simulation of equations and the display of each variable value by color, sets of variables can be formatted on variable-list lines in numeric form. The allowable formats are %b, %u, %d, and %h. They format variable sets into binary, unsigned decimal, 2's complement decimal, and hexadecimal, respectively. A given set of variables can be formatted multiple times in multiple ways. A numeric display is specified by placing the format specification followed by a list of variables inclosed in braces on a list line (with or without other variables). Example: %b{a b c d}. The **run** screen shows a numeric value in place of such fields. If the field contains only input variables and/or state variables, a hand will appear when you hover over it. Clicking the field will increment the number being displayed and update the values of the variables in the field correspondingly. This makes going through all possible combinations of a set of variables very easy. This is a common way to test combinational logic created from truth tables. Clicking the RUN button is the normal way to test sequential logic.

A rudimentary vector notation exists that greatly reduces the size of the design file (but not the simulation screens). This makes entering designs involving multiple bits that are closely related (such as bits of a register or adder) much easier and much less error prone. To use it, variable names must end in a number. A set of such variables can be expressed by following the name without the number followed by the beginning number, two periods, and the ending number of a set enclosed in brackets. When such a set appears in a list statement it is expanded into the set on that single line. When a set appears in an assignment statement, it is converted into a set of assignment statements where each term of each equation that has a vector expressed must contain the same number of elements. A simple example of a register that has only a bit-reverse control is:
R[7..0].d = ~brev R[7..0] + brev R[0..7];
Better example appear below.

Example VisiBoole Uses

The first example is an 8-bit ripple-carry adder. The .vbi file consists of 6 statements – a chain of full-adders. The first four statements list the inputs and place a copy of the outputs in a handy to verify location. The last two are the actual design expressed in vector notation. They specify the standard equations for the sum and carry bits of a full-adder circuit. The design is given below:

```
c[8..1]     %b{c[8..1]}    c0;
   a[7..0]   %b{a[7..0]}   %u{a[7..0]};
   b[7..0]   %b{b[7..0]}   %u{b[7..0]};
   s[7..0]   %b{s[7..0]}   %u{s[7..0]};
s[7..0] = ~a[7..0] ~b[7..0] c[7..0] + ~a[7..0] b[7..0] ~c[7..0]
        + a[7..0] ~b[7..0] ~c[7..0] + a[7..0] b[7..0] c[7..0];
c[8..1] = a[7..0] b[7..0] + a[7..0] c[7..0] + b[7..0] c[7..0];
```

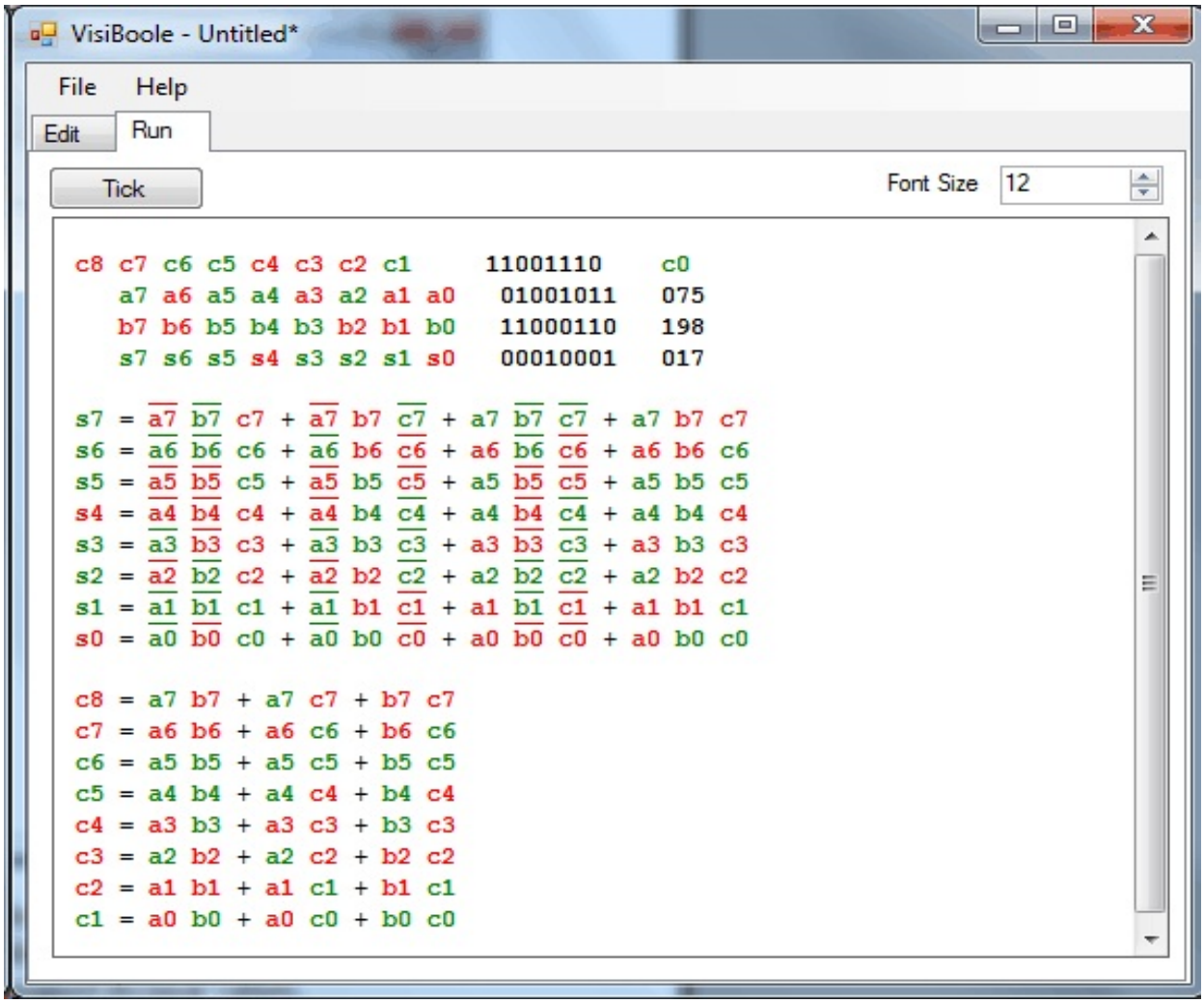

Figure 1. Sample VisiBoole Display Showing an 8-bit Ripple-Carry Adder
Red indicates a value of **1**(or true or active), **green** a **0** (or false or inactive),
the green values show the fields to their left as binary and unsigned decimal values.

Three more examples are presented and are ordered from basic to complex. Only the run screens are shown. The input equations can be reconstructed from them. 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.

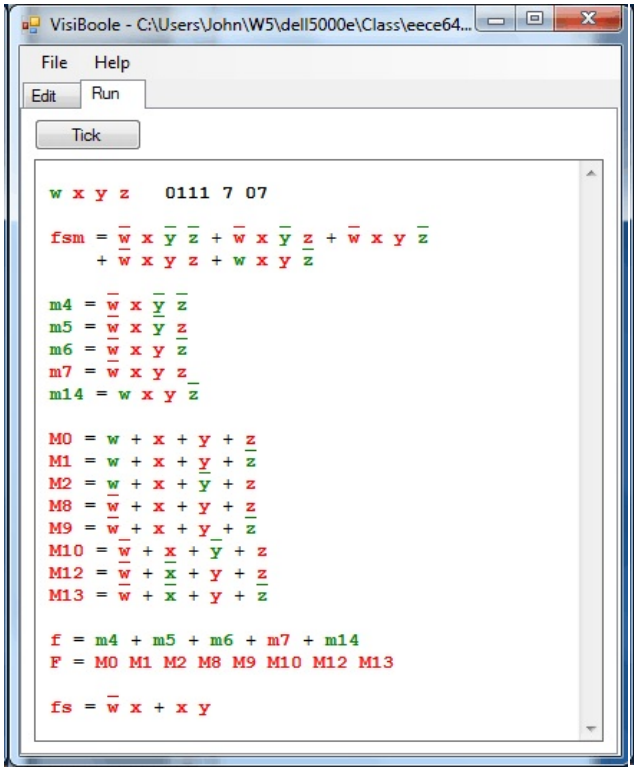

Figure 2a. F(0111)

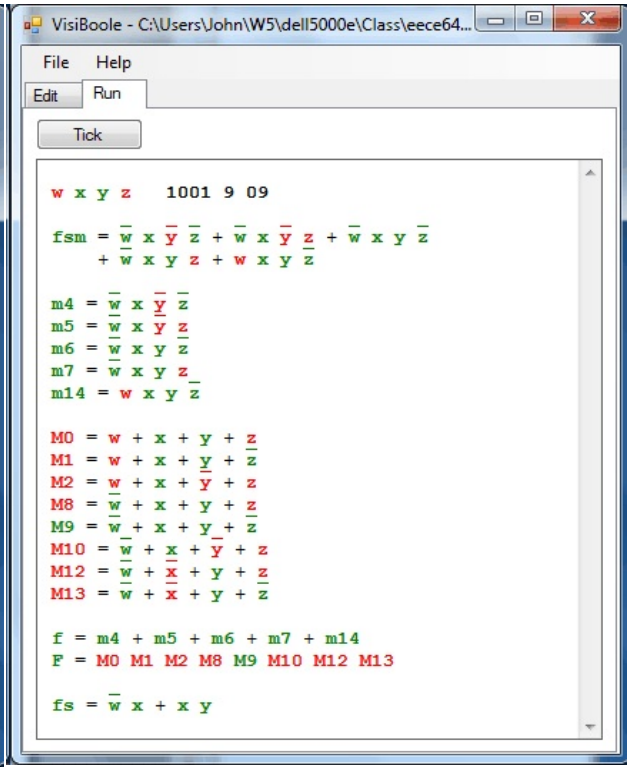

Figure 2b. F(1001)

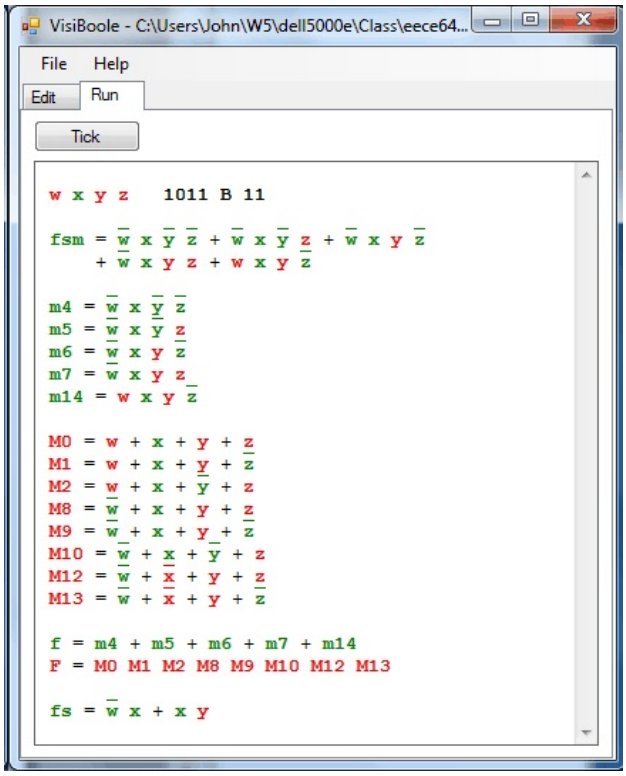

Figure 2c. F(1011)

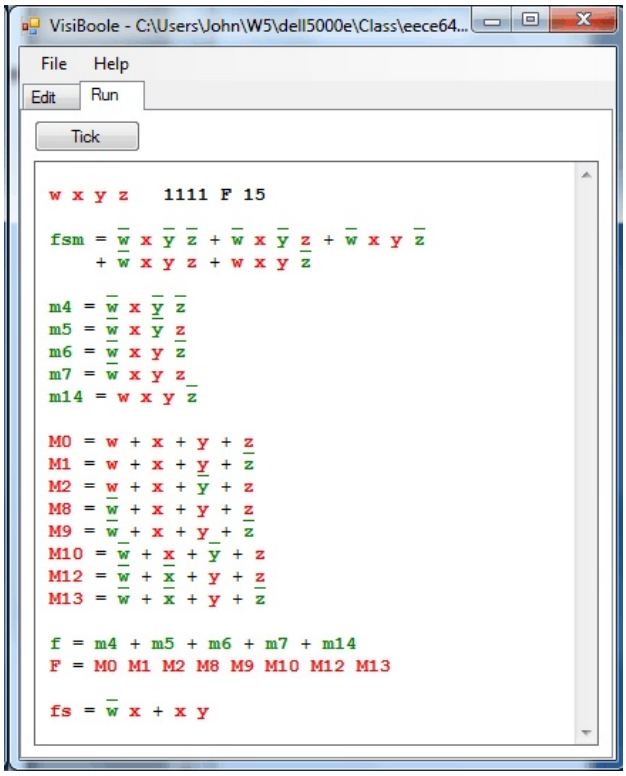

Figure 2d. F(1111)

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.

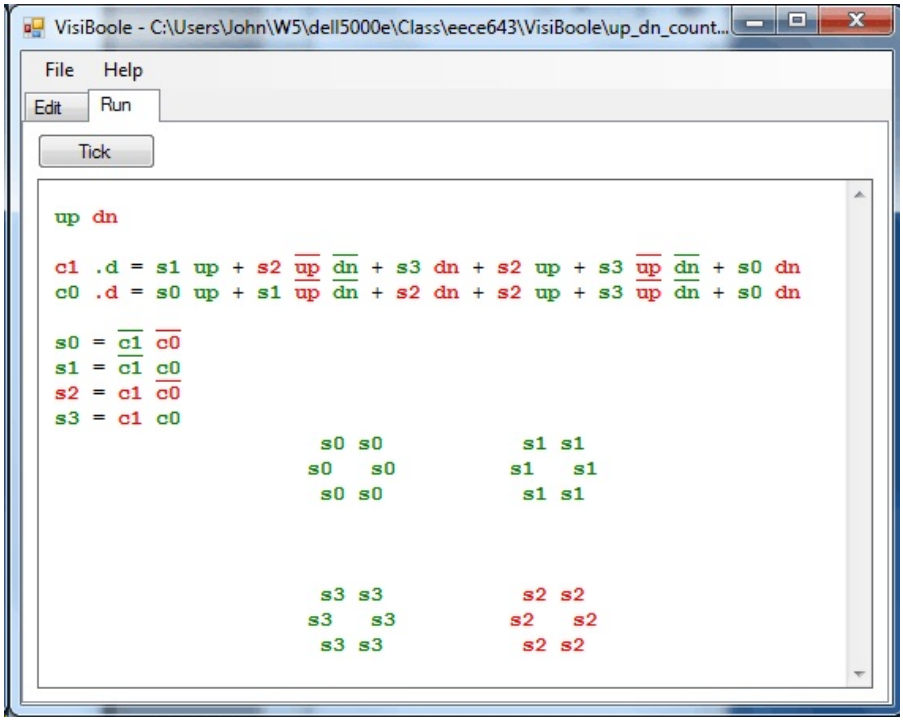

Figure 3. An Up-Down Counter with State-Diagram-Like Display

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.

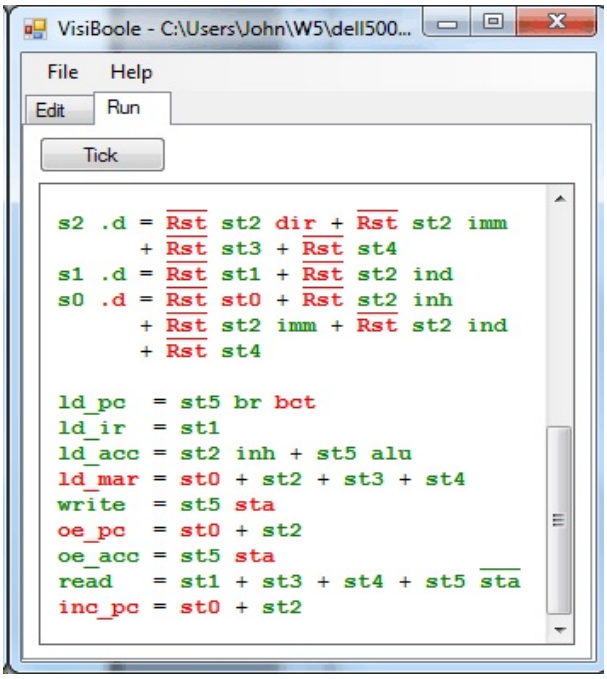

Figure 4a. st0: MAR <- PC++ :Next 1

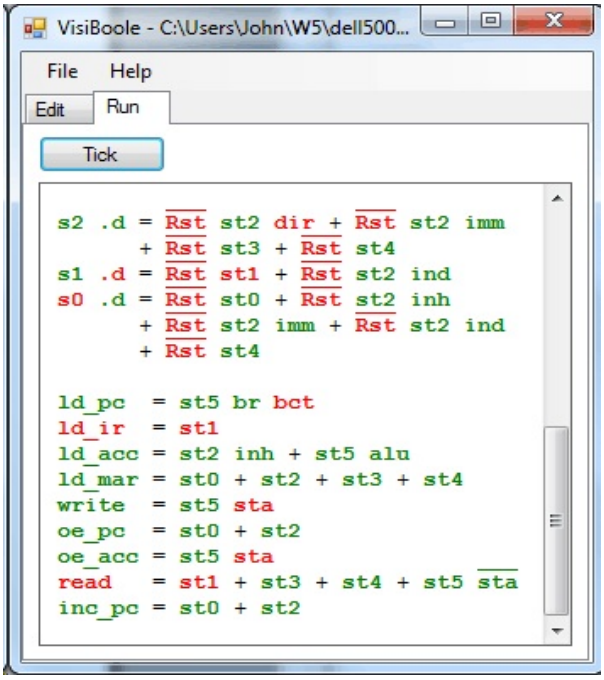

Figure 4b. st1: IR <- M[MAR] : Next 2

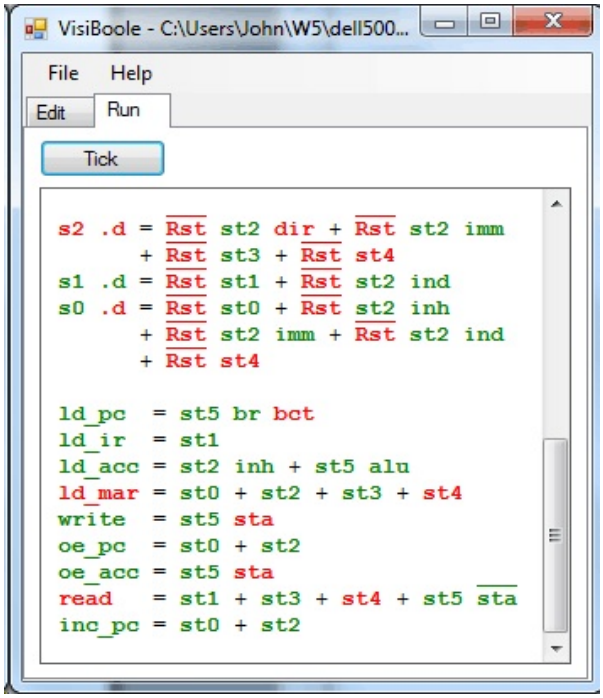

Figure 4c. st2 dir: MAR <- PC++ :Next 4

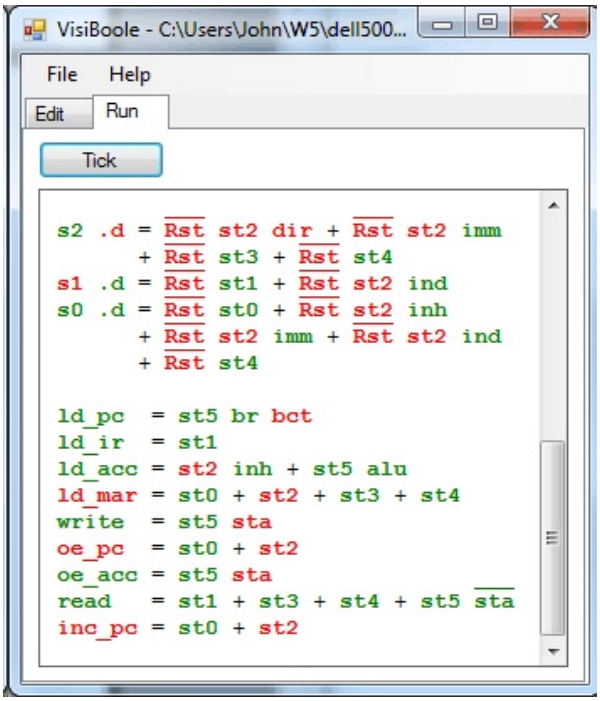

Figure 4d. st4: MAR <- M[MAR] :Next 5

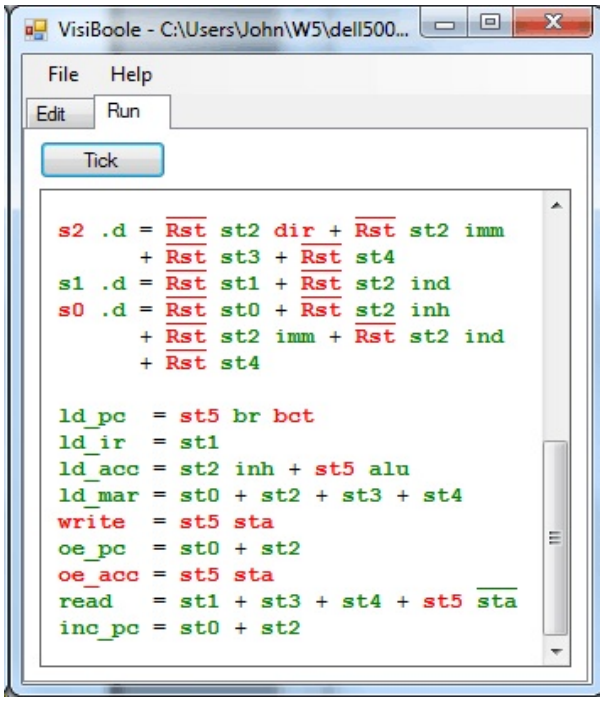

Figure 4e. st5 sta: M[MAR] <- acc :Next 0

Students in ECE 643 implement this computer in hardware using a handful of 22V10 chips. The control unit occupies one of the chips. For several semesters VisiBoole has been available for the students to create and test their CU design before converting it into a HDL for programming. Since VisiBoole has been in use, the class has been able to create and debug their CU much faster than in any previous semester. Every student stated that they enjoyed interacting with the VisiBoole display. In addition, the author's student evaluation ratings have increased.

Future Enhancements

The wish list of features is far to long to include in this document. The features will be prioritized and addressed as time and funding allow. Below is a sampling of our thoughts:

1. Changeable font so that other colors or attributes could be used for true and false so that the program would be useful to red/green colorblind users.
2. 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.
3. Support definable RAM and ROM elements to incorporate into designs.
4. 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.
5. Create a RECORD option that will generate a test-vector script from the interactive exercising of a design.
6. Add support for asynchronous sequential circuits. At present no propagation delays are factored into the simulation of the equations.
7. Provide code conversion so a working VisiBoole design can be exported in VHDL or other HDL that can be compiled and downloaded into actual hardware.

Evaluation

This software has been used mainly 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 couple of years 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 several semesters 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.

Bibliography.

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.