ENHANCING AUTOMATED ASSESSMENT FOR ENGINEERING MOOCS

Azam Beg¹, Mouza M. Alhemeiri² and Ajmal Beg³

¹,²United Arab Emirates University, Al-Ain, United Arab Emirates; ³Cortex Business Solutions, Calgary, Canada
¹abeg@uaeu.ac.ae; ²201300917@uaeu.ac.ae; ³ajmal_beg@yahoo.com

ABSTRACT
Massive Open Online Courses (MOOCs) have the potential to free up learners from the bounds of time and space. Considering the likelihood of large numbers of students enrolling in a MOOC, the ability to create large numbers of test question and answers is highly desirable. In this paper, we present a set of algorithms for automatically creating questions for different courses in electrical/computer engineering (ECE). We have implemented the algorithms in a common scripting language. We have included examples of test questions for two different ECE courses. However, the presented methodology can be applied to many other courses. An advantage of our methodology is that it eliminates the need for the purchase and maintenance of a commercial design-software package.

KEY WORDS
Verilog Hardware Description Language (HDL) Model, Logic Circuit, Boolean Equation, Electrical and Computer Engineering, Massive Open Online Course, Automated Assessment

1 Introduction

The wide availability of Massive Open Online Courses (MOOCs) has facilitated lifelong continued learning. The MOOCs are somewhat different from distance education courses in that the latter entail significant interaction among the students and the instructor. In contrast, the MOOCs tend to have unlimited enrollment but with limited active involvement of an instructor.

The MOOCs generally rely on automated assessment. For the sake of improving credibility of the assessment process, it would be highly preferable to present every student with a unique set of questions. To do so, question-and-answer sets need to be generated on the fly.

Electrical/Computer Engineering (ECE) courses such as Logic Design, Digital Electronics, Computer Architecture, etc., are quite amenable to automated assessment process. One way of expediting the creation of test-questions is relying on randomly generated logic functions and their synthesized forms. The synthesis entails converting an abstract form of a logic function into a set of logic gates.

In this paper, we present a set of algorithms for automatically and randomly creating both the Boolean equations and their synthesized descriptions; these can be used for creating question-statements, truth tables, schematics, etc. The equations and the descriptions help generate not only the questions but also their answers.

Section 2 concisely reviews the related work. Section 3 describes our set of algorithms for creating the Boolean equations and their descriptions. Section 4 details how the algorithms are implemented. This section also provides a set of sample outputs. Section 5 presents use-cases for two different ECE courses and Section 6 concludes the paper.

2 Related Work

These days, a wide range of ECE courses are available in the MOOC format [1–3]. The assessment instruments for digital/logic design courses include short quizzes, homework assignments and examinations, which in turn require creation of Boolean equations, truth tables, schematics, etc.

Several commercial circuit synthesis and schematic-creation products are available in the market, however, they usually incur hefty licensing fees; such commercial tools are available from Silvaco [4], Concept Engineering [5], Zuken [6], and Cadence [7]. The aforementioned products are capable of synthesizing circuits and creating schematics automatically, however, these products are best-suited for circuit designers rather than the academics who would want to perform online and/or automated assessment.

The research related to schematic creation includes [8–15]. A few open-source synthesis/schematic tools are also available. Verilator [16] is a synthesis tool but it neither allows behavioral code nor does it generate schematics. ChipVault [17] is also limited to the synthesis of Verilog and VHDL (two popular hardware description languages/HDLs). Icarus [18] simulation and synthesis toolset also suffers from the same limitation. QFlow [19] is yet another tool for synthesis sans schematics. Yosys [20] performs synthesis and draws circuit block diagrams but no schematics per se.

For the purpose of online automated assessment, the commercial and non-commercial products listed earlier are generally unsuitable due to installation/maintenance efforts, usage-complexity and exorbitant licensing fees. On the contrary, the tools needed by academicians for creating circuit descriptions, need to be compact and easy-to-install/host on web-servers.
3 Algorithms for Creating Random Logic Circuit Descriptions

In this section, we present a set of three algorithms for generating random Boolean functions and related circuit descriptions or Verilog models, in a speedy fashion. The algorithms can be coded in any web-friendly language, such as, Java, Perl, PHP, Python, etc. The algorithms serve these functions:

1. Creating a Boolean equation
2. Representing the Boolean equation in reverse polish notation (RPN)
3. Creating Verilog description from the RPN

The creation of random Boolean equations is constrained by a set of user-specified parameters. The user can specify the complexity of a logic function/circuit in terms of number of input variables, number of logic gates, and layers/levels. The allowed logic functions can also be specified, i.e., INV, AND, OR, NAND, NOR, XOR, and XNOR. The algorithm (Algorithm-1) is given in Listing 1. Being able to specify the complexity of logic functions helps tailor the questions to the level of difficulty of a course or a topic.

A Boolean equation created by Algorithm-1 needs to be pre-processed before the former is transformed into Verilog format. To do so, we opted for widely-known RPN representation. The common RPN algorithms deal only with arithmetic functions and not Boolean functions. An issue to contend with is the identification and processing of multi-input ‘dual’ functions, i.e., NAND (applying NOT to the ANDed variables), NOR or XNOR. Therefore, we had to come up with a modified algorithm to process logic operations, both singular and dual. The algorithm (Algorithm-2) is shown in Listing 2.

An RPN-formatted Boolean question facilitates the process of Verilog code creation. Firstly, the string array representing the RPN-equation is parsed to create an array that contains: type of logic gate, set of gate input signals, and gate output. The internal wires are also identified. Then, the Verilog-file creation starts by generating the required ‘module’ command and the set of ‘input,’ ‘output,’ and ‘wire’ declarations. After this step, all gate declarations are made in the proper sequence: type (AND, OR, etc.), label (G1, G2, etc.), output and input nets. Lastly, the ‘endmodule’ keyword is added before saving the ready-to-use Verilog file. This algorithm (Algorithm-3) is presented in Listing 3.

```
Procedure: CreateBooleanEquation
Input:  integer nGateCount
Output: string BoolEq

initialize g, boolEq;
while (g < nGateCount) {
    Pick a random logic operation
    Pick a random number of input variables
    Apply logic operation to randomly picked variables to create a logic gate
    Save the logic gate as array-element gate[g]
    Increment g
}
Append a random number of left brackets to boolEq
Append gate[0] to boolEq
Append a random logic symbol to boolEq
for (g=1; g<=nGateCount; g++) {
    Append gate[g] to boolEq
    Append a random number of right brackets to boolEq
    Append a random logic symbol to boolEq
    Append a random number of left brackets to boolEq
}
Append any remaining right brackets to boolEq
return boolEq
```

Listing 1. Create Boolean Equation

```
Procedure: ConvertBoolean2RPN
Input:  string BoolEq
Output: string rpn

Store first token in BoolEq as output, outputName
for (i=2; i<length(BoolEq); i++) {
    Read token at i
    if token is a left parenthesis {
        Push it onto the stack
    }
    if token is a right parenthesis {
        Until the token at the top of the stack is a left parenthesis, pop stack contents to output string rpn
    }
    if token is an operator {
        while (operator precedence <= stack peek precedence) {
            Pop stack contents to rpn string
        }
        if token is an operator {
            while (operator precedence <= stack peek precedence) {
                Pop stack contents to rpn string
            }
            if token is an operand {
                Append token to rpn string
            }
        }
    }
}
Pop remaining stack contents to rpn string
```

Listing 2. Transform a Boolean Equation into RPN notation
Listing 3. Create Verilog HDL model from RPN notation

4 Implementation

We have implemented the three algorithms presented in the previous section as a Matlab script. We verified the script by comparing the Boolean outputs of the auto-generated Boolean equations and the corresponding Verilog modules. The script is fast enough to create thousands of unique questions in a matter of minutes. Given below are two examples of script runs:

Example 1: A 3-variable, 4-gate Boolean equation and its Verilog equivalent, both generated by the Matlab script, are shown below. For reference, the circuit schematic (based on the description, but not generated by the script) is also included.

Y = (((¬(c)) . (b)) + (¬(a) )

Example 2: This is an example of an auto-generated 3-variable Boolean equation, represented by 15 gates. The corresponding Verilog code, also created by the script, is shown below.

Y = (¬(b) + (¬(a·b)) . (¬(b)) . (a·c)) ^ (¬(b+c)) . (b) . (¬(a·b ))
5 Use Cases

The implementation of the algorithms presented in this paper enables the creation of many types of questions for different ECE courses. (Note that a development language, such as Matlab or Python, can easily add aesthetics to the Boolean equations – as in the following examples).

Logic Design – A few examples of assessment questions are given below:

1. Create the truth table for the following Boolean function:
   \[ Y = a + b \oplus c \]

2. Simplify the following logic function using Boolean theorems and draw the schematic for the simplified function:
   \[ Y = a.(b.c) + (c + d) \]

3. Represent the following Boolean function in the sum-of-product format:
   \[ Y = (a + b).x + (b + c.d) \]

HDL Design – The following are a few sample questions:

1. Transform the following dataflow style code into gate-level style:
   \[ Y = (a.b + c) \oplus (c + d) \]

2. Create an exhaustive testbench for the following circuit:

   ```verilog
   module function_Y (Y, a, b, c);
      input a, b, c;
      output Y;
      wire w1, w2, w3, w4, w5;
      and G1 (w1, c, b);
      not G2 (w2, w1);
      not G3 (w3, a);
      and G4 (w4, w3, w2);
      or G5 (w5, c, b);
      xor G6 (Y, w5, w4);
   endmodule
   ```

3. Use force command to inject stuck-at-1 faults into input-a of the following circuit and find the test-vector(s) that detect(s) the fault:

   ```verilog
   module function_Y (Y, a, b, c, d);
      input a, b, c, d;
      output Y;
      wire w1, w2;
      nand G1 (w1, a, b);
      xor G2 (w2, c, d);
      or G3 (Y, w1, w2);
   endmodule
   ```

6 Conclusion and Future Work

We have proposed a set of algorithms for facilitating automated question-creation for different ECE courses offered in traditional or online formats. We have shown that large question-sets can be built without deploying fully-fledged commercial electronic design automation tools.

We are currently developing a method for creating schematics from the auto-generated Verilog code, which would significantly enhance the range of self-generated questions.

References


