Spreadsheets in Education (eJSiE)

Volume 1 | Issue 3 Article 1

10-5-2005

Using Spreadsheets for Teaching Principles of On-line Checking of Logic Circuits

Ilya Levin
School of Education, Tel Aviv University

Vadim Talis
School of Education, Tel Aviv University

Using Spreadsheets for Teaching Principles of On-line Checking of Logic Circuits

Abstract
This paper examines the use of spreadsheets as a tool for learning theoretical principles of concurrent error detection. Basic concepts of concurrent checking are presented by using specific spreadsheet templates. A matrix representation of a system of logical functions is used for this aim. A specific technique is described for constructing a logic simulator implementing this matrix representation. After the logic simulator construction, students are able to solve practical tasks due to understanding its theoretical basis.

The proposed spreadsheet simulation approach for teaching the subject achieves the theoretical goal of the lesson by making use of practical student activities.

Keywords
spreadsheets, learning environment, concurrent error detection, logic design learning

Using Spreadsheets for Teaching Principles of On-line Checking of Logic Circuits

Ilya Levin
School of Education, Tel Aviv University
i.levin@ieee.org

Vadim Talis
School of Education, Tel Aviv University
talisv@yahoo.com

October 5, 2005

Abstract

This paper examines the use of spreadsheets as a tool for learning theoretical principles of concurrent error detection. Basic concepts of concurrent checking are presented by using specific spreadsheet templates. A matrix representation of a system of logical functions is used for this aim. A specific technique is described for constructing a logic simulator implementing this matrix representation. After the logic simulator construction, students are able to solve practical tasks due to understanding its theoretical basis.
The proposed spreadsheet simulation approach for teaching the subject achieves the theoretical goal of the lesson by making use of practical student activities.

Submitted January 2004; revised and accepted March 2004.

Keywords: Spreadsheets, learning environment, concurrent error detection, logic design learning.

1 Introduction

The subject of concurrent error detection is normally considered an advanced topic and therefore, is usually not covered in a textbook of introductory logic design. However, the subject can be relatively easily included in a digital logic course.

The theory of concurrent error detection, being based on exact definitions and formal reasoning, represents a good example of a well-formed academic discipline. At the same time, the practical component of the discipline is industry oriented and, consequently, requires non-formal understanding of its basic principles. Indeed, the concept of concurrent error detection has come from the industrial practice. The central question in this field of computer engineering is how a fault affects behavior of a specific circuit.

Both the analytical and the synthetic kinds of thinking are useful here and are usually applied when solving problems from the above field. The analytical thinking deals with a specific form of question: "what if..." which in this case means "what kind of behavior will a certain circuit produce in presence of a specific fault?" In this context, a concept of the *fault model* (a formal representation of a real physical fault) is typically used. The alternative synthetic thinking is usually applied in the form of synthesis of a circuit followed by its computer-based simulation. While the analytical thinking concerning concurrent checking circuits is usually connected with a theoretical approach to solving the problem, the synthetic thinking supports a practical approach. Needless to say, both of the approaches are essential in studying concurrent checking, and have to be supported by appropriate educational means.

A gap between the theoretical and practical approaches to teaching the topic demonstrates a typical problem in engineering education, which is a contradiction between the practical orientation of the engineering education and its academic depth.

In this work, we illustrate a possibility to close the gap between the theoretical and the practical components of the discussed topic by using circuits' simulation within a Microsoft Excel-based learning environment.

We propose to use a matrix logic simulator [1] as a basic model for simulating a logic circuit within a spreadsheet. This simulator allows implementing a system of logical functions as a specific spreadsheet template, and very simple reprogramming of the system. As a result, the spreadsheet becomes an ideal environment for teaching a wide plurality of topics that are based on logical functions. The matrix simulation was used in a number of applications including: circuit simulation [1], educational robotics [2], state machine simulation [3], and logical control design [4], [5].

The paper is addressed to those teachers of a digital design course who are interested in introducing the subject of concurrent error detection into the curriculum. It has to be especially useful for instructors of practical exercises since the practice lessons have to provide a variety of student activities in the limited time available.

In the present paper we expand the above list by introducing a method for teaching basics of concurrent checking circuits. According to the proposed approach, students build matrix models of logic circuits within the spreadsheet, explore circuit behavior in the presence of faults and come to general conclusions concerning fundamental concepts of the topic under discussion. The approach is oriented to teaching the theoretical basics of the discussed topic. It is focused on developing and exploring a spreadsheet based learning environment that enables students to build logical circuits having a self-checking ability.

We assume that the students have basic skills in operating a spreadsheet and in defining functions for its cells. Thus we consider the students to be able to implement the following computer-based scenario under teacher guidance.

1. Development of a spreadsheet template modeling a circuit.

2. Observing operation of the circuit modeled by the template. Observations are made both without a fault, and in the presence of a fault.

3. Verification of the circuit from the point of its self-checking ability.

Our virtual lessons comprise four learning modules. Each of these modules implements the three steps and is dedicated to a specific topic of the subject. The modules are related in such a way that after performing a specific module a student will be motivated to perform the next one. In other words, each module solves the problem of a specific topic and formulates a new problem for the next module. Modules that are discussed in the paper relate to the following topics.

1. Combinational circuits
2. On-line checking circuits
3. Error-detection coding
4. Self-checking checker

The paper is organized as follows. Section 2 introduces the concept of a matrix logic simulator as a main tool for modeling circuits by a spreadsheet. Appropriate definitions from the field of on-line checking circuits are given in Section 3. A concurrent error detection architecture within the spreadsheet is described in Section 4. Section 5 gives an example of using the proposed approach at a lesson. Conclusions are presented in Section 6.

2 Logic circuits within a spreadsheet

Spreadsheets are useful and flexible modeling tools. They may be regarded as normal calculator-type tools, but are also universal homogeneous two-dimensional fields, which can be used for implementation of various computational and logical functions forming any functional circuit.

The paper [1] considers the use of spreadsheets for simulation of a system of logical functions in matrix form (usually called Programmable Logic Arrays (PLA) [6] or Decision Tables [7]). The main idea of such a simulation is based on a two-layer spreadsheet template. The first *programmable* layer relates to a system of logical functions in its matrix form, while the second *functional* layer implements the universal PLA matrix. Students are able to implement any system of logical functions by reprogramming the first layer of the spreadsheet template. Moreover, they are able to run the simulation of the circuit and analyze its functioning "on-line".

Any system of logical functions $y_1, \ldots, y_N$ of variables $x_1, \ldots, x_M$ can be presented in a Sum-of-Products (SOP) form and implemented in the form of a two level structure: products $X_1, \ldots, X_H$ are implemented at the first level of this structure and sum $y_1, \ldots, y_N$ of these products at the second level.

Table 1: Description of the matrix structure

x_1	x_2	x_3	x_4	y_1	y_2	y_3
1	0	–	–	1	1	•
0	1	–	1	1	•	1
0	–	0	0	1	•	•
1	–	1	0	•	1	•
1	1	–	–	•	1	1
–	0	–	0	•	1	•
0	–	0	–	•	•	1

Let us consider one example of the matrix implementation [1] of a system of logical functions, using a system presented in eqs 1–3 in a SOP form.

$$y_1 = x_1 x_2^{'} + x_1^{'} x_2 x_4 + x_1^{'} x_3^{'} x_4^{'} \tag{1}$$

$$y_2 = x_1 x_2^{'} + x_1 x_3 x_4^{'} + x_1 x_2 + x_2 x_4 \tag{2}$$

$$y_3 = x_1^{'} x_2 x_4 + x_1 x_2 + x_1^{'} x_3^{'} \tag{3}$$

Any SOP can be presented in tabular form. Columns appearing in the tables are marked by variables $x_1, \ldots, x_M$ and functions $y_1, \ldots, y_N$. Every product corresponds to a specific row of the table. Character **1** appears at the intersection of row h and column m, if the variable x_m is contained in the term X_h in the direct form. Character **0** appears at the intersection of row h and column m, if the variable x_m is contained in the term X_h in the inverse form. Character "-" appears at the intersection of row h and column m**,** if the variable x_m is absent in the product X_h.

Character **1** appears at the point of intersection of row h and column y_N**,** if term X_h is contained in the function y_N $(n = 1 \ldots N)$; Character • appears in the opposite case.

The logic arrays of our example, corresponding to system (1) are shown in Table 1.

A hardware circuit implemented in the form of a two level matrix structure which has a fixed numbers of inputs **M**, internal rows **H** and outputs **N** is known as a *Programmable Logic Array* (PLA). The PLA implements a SOP form of the system of logical functions of interest.

The proposed method of SOP simulation includes constructing two interacting worksheets named **SOP** and **PLA**. Worksheet **SOP** represents the SOP of a system, and worksheet **PLA** models the PLA operation.

Every cell of **PLA** is programmed by a universal function, which will be presented below. This function's value depends on the value of a corresponding cell of the worksheet **SOP**, which is a copy of Table 1. Let us now construct the **PLA** worksheet.

Variables x_1, x_2, x_3, x_4 and y_1, y_2, y_3 are directly copied on to the worksheet **PLA** from Table 1. According to the matrix structure, each cell of the AND matrix can implement one of three different functions: reference to the contents of the left hand cell, product of the variable and contents of the left hand cell, product of negation of the variable and contents of the left hand cell. Each cell of the OR matrix can implement

one of two functions: reference to the value of the higher preceding cell or the sum of the higher preceding cell and a corresponding output of the AND matrix.

Functions of the worksheet **PLA** are defined by the following rules.

For the AND array:

If 1 appears in a cell (m, h) of **SOP**, then the product of the value of the corresponding variable and that of a preceding left hand cells will be updated into the correspondent cell $F_{m,n}$. This can be expressed by the function $F_{m,n} = F_{m-1,h}$ & $F_{m,1}$.

If 0 appears in a cell (m, h) of **SOP**, then the conjunction of the inversion of value of the corresponding variable and of the value of the preceding left hand cells will be updated into the corresponding cell $F_{m,n}$. This can be expressed by the function $F_{m,n} = F_{m-1,h}$ & $F'_{m,1}$.

If "-" appears in a cell (m, h) of **SOP**, then the value of the corresponding variable will be updated into the corresponding cell $F_{m,n}$. This can be expressed by the function $F_{m,n} = F_{m-1,h}$.

Formally, all these conditions can be expressed as one function:

$$F_{m,n} = \begin{cases} F_{m-1,h} \ \& \ F_{m,1} & \text{if 1 appears in cell } (m,h) \text{ of SOP} \\ F_{m-1,h} \ \& \ F'_{m,1} & \text{if 0 appears in cell } (m,h) \text{ of SOP} \\ F_{m-1,h} & \text{if "-" appears in cell } (m,h) \text{ of SOP} \end{cases} \quad (4)$$

For the OR array:

If character **1** appears in the cell (m, h) of **SOP**, then sum of the value of the corresponding variable and the value of the higher preceding cell will be updated into the corresponding cell $F_{m,n}$. The indicative function in this case is $F_{m,h} = F_{m,h} + F_{m,h-1}$. If character $\bullet$ appears in a cell (m, h) of **SOP**, then the value of the higher preceding cell will be updated into the corresponding cell $F_{m,n}$. This can be expressed by function: $F_{m,h} = F_{m,h-1}$.

Formally, both of above conditions can be expressed in the form of the function $F_{m,h}$:

$$F_{m,h} = \begin{cases} F_{m,h} + F_{m,h-1} & \text{if 1 appears in cell } (m,h) \text{ of SOP,} \\ F_{m,h-1} & \text{if } \bullet \text{ appears in cell } (m,h) \text{ of SOP.} \end{cases} \quad (5)$$

Functions (4) and (5) are implemented within the cells of the AND and OR arrays of the SOP worksheet correspondingly.

In [8], the logical capacity of spreadsheets was used for teaching principles of the error detection in digital logic courses. The present study also uses the logical capacity of spreadsheets as a basis for developing systems of logical functions. The simplicity of both defining and redefining any logic system is exploited in our study for modeling functional circuits in the presence of a fault, in comparison with the fault-free case.

3 Basic definitions

The *self-checking* property can be defined as the ability of a circuit to verify concurrently and automatically, whether there is any fault in the circuit's logic (without the need for

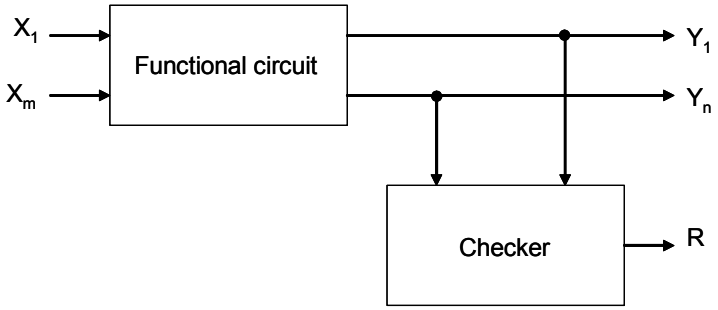

Figure 1: Concurrent checking architecture.

an externally applied test). Thus, self-checking circuits allow concurrent on-line error detection, i.e., allow detection of faults during the normal operation of the circuit.

A circuit is *self-testing* if, for every fault from an assumed fault set, the circuit produces a non-codeword at the output for at least one input code word.

A circuit is *fault-secure* for an assumed set of faults if, for any fault in the set, the circuit's output is either a correct codeword or a non-codeword. It means that the circuit never produces an incorrect codeword for an input codeword.

The circuit is *totally self-checking* if it is both self-checking and fault secure. Totally self-checking circuits are very desirable for highly reliable digital systems, since during normal operation all faults from a given set would cause a detectable, erroneous output.

A schematic diagram of a concurrent self-checking circuit is shown in Figure 1. It consists of a functional circuit (FC) and a checker, both of which are self-checking. The function of the checker is to check validity of output codewords of the circuit.

We use a Sum-of-Minterms (SOM) based checker [9] that implements a sum of all minterms corresponding to output vectors $y_1, \ldots, y_N$ of the functional circuit. When an output vector is a codeword, the corresponding minterm will be activated and consequently, the output error signal R will be equal to one. In the opposite case, when the output vector is a non-codeword, no SOM checker's minterm will be activated and the error checker's signal R will be equal to 0, which means that an error is detected.

4 Concurrent checking architecture within a spreadsheet

Now we construct an Excel template that simulates functioning of the self-checking functional circuit implementing a certain system of logic functions represented in a matrix form. We assume that students are able to create a spreadsheet model of the logic system.

We construct the checker as a SOM-checker [9]. The SOM checker implements a logical function and, consequently, can be implemented by the matrix model. Table 2 shows a programmable layer of the spreadsheet model for the self-checking architecture for an exemplary system of logical functions.

Table 2: Programmable matrix of spreadsheet template for self-checking scheme

x_1	x_2	y_1	y_2	y_3	
0	0	*0*	*0*	*1*	
0	1	*1*	*1*	*0*	
1	0	*0*	*0*	*0*	
1	1	*1*	*0*	*1*	
		y_1	y_2	y_3	R
		0	0	1	*1*
		1	1	0	*1*
		0	0	0	*1*
		1	0	1	*1*

The upper matrix in Table 2 corresponds to a functional circuit (FC) (x_1 and x_2 are inputs of the FC y_1, y_2 and y_3 are its outputs) while the lower matrix corresponds to the checker (R is an error signal). Notice that here and below, AND matrices are written in a regular font while OR matrices are written in italic.

5 Sample lesson for studying the self-testing property

Students study basics of the concurrent error detection (which are unidirectional errors and an unordered coding) by developing a matrix model of a functional circuit implementing a system of logic functions and by constructing the SOM checker for this system (Table 2).

Students "inject" a fault into the spreadsheet model of the functional circuit and observe/analyze consequences of the fault injection, using the R output of the circuit. Table 3 presents the matrix from Table 2, upon injection of the certain fault. Let this fault is a stuck-at-1 crosspoint fault in the intersection of the output line y_2 and the first product term. Such a fault can be presented by "one" instead of "zero" in the intersection of the first row and the y_2 column of the programmable matrix (see Table 3). Students can come to a conclusion that this fault will be detected when the input vector (00) occurs. In this case, the output vector of the FC is equal to (011) and, since this vector is absent in the AND matrix of the checker, the checker's output will be equal to 0 that is a signal of error. This error signal can be observed in the functional layer of the matrix model.

Students come to the conclusion that the proposed model enables detection of all errors that lead to occurrence of a non-code output word. However, some faults may lead to appearance of a code vector, which means that this fault is undetectable. Two following examples illustrate this situation.

Example 1 *If due to a fault, the third and the fourth product terms (see the third and the fourth lines of Table 2) are equal to one, the resulting output vector will be equal to codeword 101 resulting from the OR summing of 000 and 101.*

Table 3: Programmable matrix after fault injection

x_1	x_2	y_1	y_2	y_3	
0	0	*0*	**1**	*1*	
0	1	*1*	*1*	*0*	
1	0	*0*	*0*	*0*	
1	1	*1*	*0*	*1*	
		y_1	y_2	y_3	R
		0	0	1	*1*
		1	1	0	*1*
		0	0	0	*1*
		1	0	1	*1*

Example 2 *The fault is a stuck-at-1 fault in the y_3 output column, on its third row. In this case, when input vector 10 occurs, the resulting output vector will be equal to correct codeword 001 which means that the fault is not detected.*

In both of the above cases, the error signal is absent despite the presence of the fault. The students have to analyze the reason for this phenomenon, and will be asked to suggest their own solution to this problem. In other words, they will have to propose a method of achieving the self-testing property of the system. One well-known solution of the problem is to complement outputs of the Functional Circuit by check bits representing any unordered code [10]. The teacher may suggest this elegant solution after explaining the nature of the problem. The students may complete the template with the unordered code as shown in Table 4. In this table, the well known modified Berger code is applied (each output vector is additionally provided with check bits being a binary value of the number of zeros in the vector decreased by one). Notice that introduction of the check bits does not require introduction of any new formula. Indeed, the check bits of the FC are programmed by the OR array formula (3), while the check bits of the checker are programmed by the AND array formula (2).

Upon building such a template, the students can verify the self-testing property of the functional circuit and the checker. They are able to inject faults that were not detected in an original circuit and observe that the new solution enables detection of these faults. This remarkable fact can motivate students to answer for a question: why it became possible? What price do we pay here for achieving the self-checking property? The teacher should be able to introduce the concept of redundancy and to explain the idea of trade-offs between the redundancy and reliability in solving technological problems.

The next step of the work can be stimulated by the fact that the checker does not meet the self-checking property. Indeed, if any of its output crosspoints (R-column) become stuck-at-1 (or so-called *stuck-at-product* faults), the checker will never produce the error signal. The teacher may propose that the students solve this problem by themselves. The authors' experience indicates that some students are able to solve this problem and, sometimes, even to propose interesting solutions.

Table 4: Spreadsheet template for unordered coded circuit

x_1	x_2	y_1	y_2	y_3	*Check*	*bits*	
0	0	*0*	*0*	*1*	*0*	*1*	
0	1	*1*	*1*	*0*	*0*	*0*	
1	0	*0*	*0*	*0*	*1*	*0*	
1	1	*1*	*0*	*1*	*0*	*0*	
		y_1	y_2	y_3	Check	bits	R
		0	0	1	0	1	*1*
		1	1	0	0	0	*1*
		0	0	0	1	0	*1*
		1	0	1	0	0	*1*

Table 5: Programmable matrix for the self-checking scheme with dual-rail checker

x_1	x_2	y_1	y_2	y_3	*Unordered*	*code*		
0	0	*0*	*0*	*1*	*0*	*1*		
0	1	*1*	*1*	*0*	*0*	*0*		
1	0	*0*	*0*	*0*	*1*	*0*		
1	1	*1*	*0*	*1*	*0*	*0*		
		y_1	y_2	y_3	Unordered	code	R_1	R_2
		0	0	1	0	1	*1*	*0*
		1	1	0	0	0	*1*	*0*
		0	0	0	1	0	*0*	*1*
		1	0	1	0	0	*0*	*1*

The solution that may be proposed by the teacher is presented in Table 5 and termed a *dual-rail implementation*, and is very important for understanding of concurrent error detection principles. This new self-checking architecture includes two columns corresponding to the error signal (R_1 and R_2).

The students investigate the solution in the same style as the previous tasks. They inject various faults into the matrix and analyze obtained results of the matrix functioning. It would be easy to convince the students that in the case of proper functioning of the circuit the output signals of the checker will be $R_1 = 0$; $R_2 = 1$ or $R_1 = 1$; $R_2 = 0$. Every other output vector indicates the fact of the fault.

The authors of the paper checked the presented approach in teaching the subject *Introductory Logic Design* during one semester. The approach had proven its success by essentially improving the understanding of the subject by the students, and by effectiveness of practical exercises, namely the students were available to solve much more suggested practical tasks during one lesson.

6 Conclusions

We have presented a method for teaching the subject of concurrent error detection to electrical engineering students, in the framework of a digital design course.

We propose using a matrix simulator as a base for building spreadsheet models of logical circuits. This matrix simulator is suitable for constructing a flexible learning environment that allows performing a number of experiments under the teacher's guidance.

We propose a sequence of steps for creating various self-checking architectures. The aim of these experiments is to introduce to students the fundamental concepts of the theory of concurrent error detection in logical circuits. All experiments with spreadsheet templates are performed in the framework of the same lesson, step-by-step, and can therefore be considered as a spreadsheet based introduction to on-line checking principles.

Notice that contrary to the regular spreadsheet based teaching and learning methods, the proposed approach is addressed to teaching theoretical concepts, and not just teaching applicative issues of a certain subject matter.

Students are taught to build spreadsheet templates for modeling digital circuits, and these templates allow visualization of the error detection process. The proposed method can be extended for teaching other concurrent checking concepts and methods.

References

[1] Levin, I. (1993). Matrix Model of Logical Simulator within Spreadsheet, *International Journal of Electrical Engineering Education*, Vol.30, 3, pp.216-223.

[2] Mioduser, D., Levin, I., and Talis, V. (1995). Cognitive-Conceptual Models for Defining Robot Control, *Artificial Intelligence in Education*, Association for Advancement of Computers in Education, Washington, USA, 131–134.

[3] Levin, I. (1994). The State Machine Paradigm and the Spreadsheet Learning Environment, In: A.J. Smith (ed.) *Engineering Education, Increasing Students Participation*, Sheffield Hallam University, Sheffield, UK, 351–355.

[4] Levin, I., and Levit, V. (1998). Controlware for Learning with Mobile Robots, *Computer Science Education*, **8**(3): 181–196.

[5] Levin, I., and Mioduser, D. (1996). A Multiple-Constructs Framework for Teaching Control Concepts, *IEEE Transactions of Education*, **39**(4): 488–496.

[6] Baranov, S. (1994). *Logic Synthesis for Control Automata*, Kluwer Academic Publisher.

[7] Humby, E. (1973). *Programs from decision tables*, London, Macdonald and Co.; New York, American Elsevier.

[8] Anneberg, L. (1999). Error Detection and Correction Templates for Digital Courses, *IEEE Transaction on Education*, **42**(2).

[9] Levin, I., and Karpovsky, M. (1998). On-line Self-Checking of Microprogram Control Units, *4-th International On-line Testing Conference*, Capri, Compendium of papers, 153–159.

[10] Lala, P. (2000). *Self-checking and Fault-Tolerant Digital Design*, Morgan Kaufmann Publishers, San Francisco.

