Spreadsheet Learning Environment for Teaching Advanced Topics in Computer Engineering

Ilya Levin and Hillel Rosensweig
School of Education, Tel Aviv University,
Ramat Aviv, Tel Aviv, 69978, Israel
Phone: +972-3-6407109, fax: +972-3-6405068, ilia1@post.tau.ac.il, hillelro@yahoo.com

Abstract- Despite differences between Computer Science (CS) education and Computer Engineering (CE) education, certain basic principles exist in both. We present a spreadsheet model of an elementary hardware Sorter, and show its efficiency in allowing students to gain deep understanding into the complicated inner workings of Checkers and in expressing some basic principles shared by both CS and CE. We describe the usage of Sorters in designing Checkers and describe a potential constructive teaching method for advanced hardware topics.

I. Introduction

It is accepted that computers are studied in two different academic curriculums – Computer Science (CS) and Computer Engineering (CE). Though some integrated curriculums have been developed and are in use in a number of universities, there is a principle difference in the approaches to teaching CS and CE.

The difference can be seen at all levels – in the curriculum, in the theoretical base, in the methodology of research, etc. Due to that, graduating students of CS and CE have different approaches to problem solving, different structure of scientific knowledge, and different practical capabilities and skills. Understanding these differences between CS and CE is very important for preparing and teaching specific courses within the two curriculums

For example, the process of teaching hardware courses has some distinctive features in comparison with teaching software courses. It should be noted that these features couldn't be formulated just as different mental approaches or difference in basic knowledge. The very approach to solving problems and performing tasks can be principally different.

While the key term in teaching software is the "algorithm", the key term in teaching hardware is "structure" and interaction between elements in a structure. These different approaches and terms indicate computer specialists of two different types: a programmer and a hardware engineer.

In this paper, we focus our attention not on the differences between approaches; on the contrary, we formulate principles we believe to be common to both approaches. It is our goal to show that these principles may become the base for building individual learning environments by students. One of such environments is the spreadsheet tool. The idea using the spreadsheet as learning environment in Computer Engineering was studied in [1, 2]. Later the approach was extended for teaching principles of on-line hardware checking [3].

Our the present work, we demonstrate a number of innovative spreadsheet based solutions for teaching complex hardware constructs belonging to advanced topics of Computer Engineering.

We examine teaching principles of a checker for equal-weighted codes - codes with an equal amount of 0's and 1's [4]. The properties of this type of topics are unique:

1) complex structural connections are characteristic in two-dimensional schemes of hardware solutions, as well as parallel functioning of elements within the scheme;

2) the traditional algorithmic approach for Computer Science is principally unsuitable when searching for a hardware solution. A more structural approach is necessary.

The common principle that binds this topic to both CS and CE has to do with the nature of solutions to problems in this area. Teaching hardware solutions has an essential difficulty. As a rule, hardware solutions have an inductive character. Namely, some elementary scheme with a simple function is taken as a base, and further on, a more complex solution is developed on that base.

If the way of connecting simple component schemes is understandable, the resulting scheme usually occurs to be understandable, as well. However, very often the resulting scheme is complex and comprises an original idea of the developer. In such situations, a student encounters a non-trivial task when the study of a common scheme requires serious and usually informal analytical skills.

Spreadsheets as Constructive learning environments

New means for computer simulations that have recently appeared present powerful tools which support solving analytic problems. One important step in teaching hardware is using computer micro-worlds and utilizing the constructive approach for building learning environments. According to this novel method, study of a scheme proceeds dynamically.

In this work, we show that learning the functioning rules of an elementary scheme, fulfilled by constructive experiments with a model of a fully integrated, complex scheme, form an excellent ground for studying complex hardware solutions.

We consider the teaching of designing an *m-out-of-n* checker. The following two hardware schemes are proposed to students, one after another:

1. Checker based on a sorting matrix,
2. Smith's Checker

We demonstrate a method of building and studying

©EDA Publishing/EWME2012

ISBN: 978-2-35500-019-5

Checkers using spreadsheet modeling.

The paper is organized as follows. Section II is devoted to building a Checker on the basis of an elementary Sorter scheme. Section III describes the structure of a cellular Sorter and the method of modeling the Sorter. Section IV describes how a Smith's Checker can be built on the basis of the cellular one, and how it can be modeled. Conclusions are given in the final Section.

II. Hardware Implementation of Sorting

The initial task is formulated as follows: for a given binary vector *B*, a combinational scheme is synthesized, transforming the vector into a sorted vector *S* in which all 1's are positioned in its left portion. Such a scheme allows for determining a variety of vector characteristics. For instance, it can easily be seen whether an initial vector is equal-weighted simply by checking the value of the central binary positions in the resulting sorted vector.

A. Basic Sorter

There is an elegant hardware solution for the task formulated above. This solution is based on using an elementary 2x2 Sorter (2 inputs - 2 outputs) and comprises of logical OR and AND elements. As can be seen from Fig.1, the elementary Sorter transforms a binary pair in such a manner, that 1's are always shifted to the left.

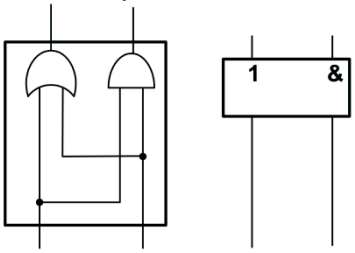

Fig. 1 elementary 2 bit Sorter

B. Sorting Matrix

To sort a binary vector of arbitrary length *n*, the natural choice is to use a matrix of interconnected elementary Sorters. The most naive implementation is shown in Fig. 2 below. Starting from the bottom layer, Sorters in each layer are shifted with respect to Sorters of the previous layer. In such a manner, connection between binary positions is achieved and the sorting is performed at the *n*-th layer.

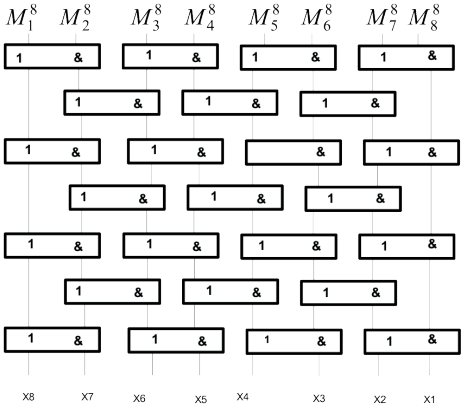

Fig. 2 8x8 Sorter made of interconnected 2x2 Sorters

C. Sorting Matrix in Spreadsheet

A simulation of an 8x8 Sorter is possible using spreadsheets. The sorting matrix spreadsheet is built as follows:

1. The first (bottom) layer comprises the initial vector *B*.
2. Each Functional layer consists of elementary Sorters. Outputs of lower layers are connected to inputs of upper layers.

TABLE I
Excel Spreadsheet Sorter Simulation

S	1	1	1	0	0	0	0	0
8	1	1	1	0	0	0	0	0
7	1	1	1	0	0	0	0	0
6	1	1	1	0	0	0	0	0
5	1	1	0	1	0	0	0	0
4	1	0	1	0	1	0	0	0
3	1	0	0	1	0	1	0	0
2	1	0	0	0	1	0	1	0
1	0	1	0	0	1	0	1	0
B	0	1	0	0	1	0	1	0

III. Cellular Checker

Based on the idea of an elementary Sorter, a scheme for a *m-out-of-n* cellular Checker (a Checker that only returns *no-error* when there are *m* 1's in a vector of *n* bits) can be built. The cellular Checker is an assembly of elementary Sorters. Specifically, the basis for building a *4-out-of-8* cellular Checker is using a 4x4 Sorter (4 inputs - 4 outputs). The 4x4 Sorter scheme comprises a number of elementary 2x2 Sorters, interconnected, so that at the output the scheme generates the desired sorted binary vector. The complete scheme for a 4x4 Sorter is shown in Fig. 3. (The scheme is taken form [4]).

Each binary position in the final sorted vector *S* corresponds to a logical function (For instance: *M(4,1), M(4,2), M(4,3), M(4,4)* in Fig. 3).

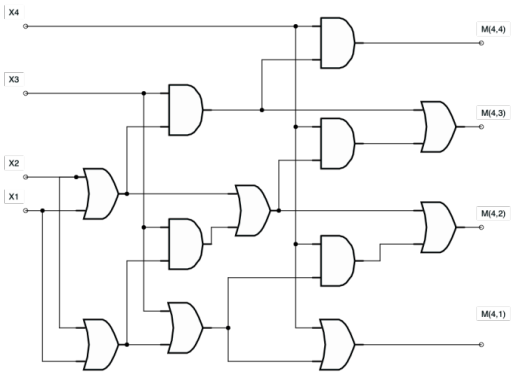

Fig. 3 4x4 Sorter

These logical functions are both symmetrical and monotonous. Let us denote these functions *M(i,n)*. Careful observation of each binary position in the sorted vector raises the following formulation:

M(i,n) is equal to one 1, when its n-positional code comprises no less than i ones.

The output vector consists of *M(1,n),..,M(n,n)*. Analysis of the scheme by analysis of each binary position in the output vector is useful for an informal yet in depth understanding of the scheme operation. For example, it is easy to see that the binary position *M(n,n)* will be equal to 1 only when the whole input vector consists of ones, and that *M(1,n)*will be equal to 0 only given a 0 input vector.

We will use the obtained functions at the next stage of building the *4-out-of-8* Checker. At the output of a *4-out-of-8* Checker, there are two outputs, which take opposite binary values when a code combination is fed to the input, and equal binary values when a non-code combination is fed to the input.

The main task of designing the Checker is implementing two functions Z_1 and Z_2, with behavior corresponding to the described outputs. These functions are implemented using 4x4 Sorters marked M_A, M_B. For implementation of the Checker these functions are expressed as follows:

$$Z_1 = M^4_{1A} \,\&\, M^4_{3B} + M^4_{3A} \,\&\, M^4_{1B} \quad (1)$$

$$Z_2 = M^4_{4B} \,\&\, M^4_{0A} + M^4_{2B} \,\&\, M^4_{2A} + M^4_{0B} \,\&\, M^4_{4A} \quad (2)$$

Z_1 and Z_2 are implemented as combinations of the output for Sorters M_A and M_B. Indeed, all possible combinations for M_A and M_B corresponding to a *4-out-of-8* code are listed in the logical equations shown above.

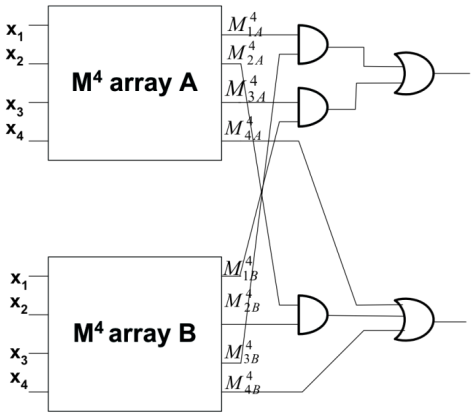

Fig. 4 m-out-of-n Checker

Each binary position of the last layer corresponds to a monotonic symmetric function. A monotonic symmetrical function *M(8,k)* is equal to 0, when the number of ones in the code is less than k. Each of the binary positions *(k=1,2,..)* of the last cascade corresponds to a monotonic function *M(8,k)*. Due to that, the rightmost binary position is equal to 1 if the code is equal to 2^{n-1}, and the last position on the left is equal to 0 when the code is equal to 0. In general, it is easy to see that the Checker somehow shifts the 1's to the bottom. This is the principle of binary sorting.

It should be noted that the described sorting matrix performs so-called structural sorting, while the classical programmed sorting operates according to the algorithm of bubble-sort.

IV. SMITH'S CHECKER

The sorting matrix is only an intermediate solution and almost cannot be used as is in practical design solutions. The scheme of Smith's Checker, which is much more practical, can also be built on the basis of a sorting matrix. The Smith's Checker uses the matrix structure in which the border right and left layers are interconnected according to special rules. These rules allow, having the *m-out-of-n* code at the input, obtaining the code 010101… at the output of the matrix. A simple assembly of AND-OR elements allows determining correctness of the code in the form of a two-bit word.

In order to understand the structure and the principles of the scheme's operation, we suggest to our students that they convert the spreadsheet matrix into the Smith's Checker [4]. The figure below (Fig. 5) is presented for that purpose. Upon building a spreadsheet model of the Smith's Checker, we may suggested to the students to simulate it's operation on different input vectors.

The spreadsheet model of the Checker not only allows students to see whether the scheme works properly, but also gives them the possibility to observe how the code passes through layers in the Checker. It allows to deeply understanding the principles working within a Checker in operation.

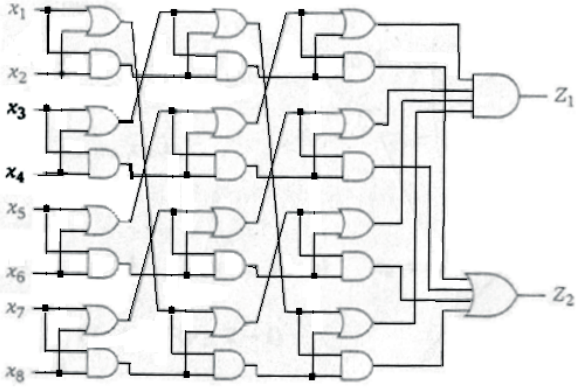

Fig. 5 Smith's Checker

CONCLUSIONS

We presented a spreadsheet model of a basic Sorter. This Sorter was shown to posses certain properties, making it ideal for designing a Checker. This model was shown to express the basic principle of inductive reasoning - building a complex construct out of very simple basic structures. The simple manner in which a Sorter can be used for the design and understanding of Checkers makes it a good candidate for advanced logic design courses, where an interactive and easily programmable interface can be a powerful tool. Using the spreadsheet model of a Sorter, students can design a Checker and truly touch on its inner workings.

REFERENCES

[1] Levin, I. (1993). Matrix model of logical simulator within spreadsheet, *Int. J. Elect. Engineering. Educ*., 30(3), pp. 216-223.

[2] Levin I., (1994). Behavioral simulation of an arithmetic unit using the spreadsheet, *Int. J. Electrical Eng. Educ*., 31, pp. 334-341.

[3] Levin I., and Talis V. (2004). Using Spreadsheets for Teaching Principles of On-line Checking Logic Circuits. *Spreadsheets in Education, Vol. 1, No. 3, 131-141*.

[4] Lala P., (2001). *Self-checking and fault-tolerant digital design*. Morgan Kaufmann Publisher.