FACTA UNIVERSITATIS (NIŠ)
SER.: ELEC. ENERG. vol. 20, no. 3, December 2007, 461-477

Synthesis of Sequential Circuits by using Linearization

Iliya Levin, Osnat Keren, and Vladimir Ostrovsky

Abstract: The paper deals with synthesis of sequential circuits defined by their algorithmic state machine notation. Such circuits have a number of specific properties which enable efficient design of the circuits by utilizing so-called linearization techniques. A typical linearization technique includes calculation of autocorrelation values for a system of logic functions corresponding to the circuit. For the mentioned sequential circuits, the calculations which usually require massive computational recourses may be significantly reduced and thus low-overhead implementations of the circuits can be obtained relatively easy. The paper introduces a novel architecture of so-called linearized sequential circuits, and a piece-wise linearization approach for synthesis of sequential circuits. Results are evaluated both analytically and by using a number of standard benchmarks.

Keywords: Sequential circuits, logic functions, linearization, Binari decision diagrams, multi-terminal decision diagrams.

1 Introduction

Linearization is known as one of efficient means for optimizing the logic design. Extraction of a maximally possible linear portion from an initial logic function or from a system of logic functions allows reducing the implementation complexity in many cases in practice. Calculations of autocorrelation values that are required for the linearization are of high complexity, which makes the linearization of functions of a large number of input variables almost inapplicable. Numerous researches were made, providing reductions of the complexity of linearization [1–5]. One of the ways to reduce the complexity of the calculations is using the disjoint cubes representation of logic functions. Calculations over disjoint cubes are much simpler than in the general case. There are two objections which prevent considering the disjoint cubes representation as a ”panacea” in performing the linearization:

Manuscript received August 14, 2007.
The autors are with Tel Aviv University, Bar Ilan University, Israel

1. Complexity of the calculations over the disjoint cubes strongly depends on the order of processing the cubes.
2. An arbitrary disjoint cube representation usually comprises a great number of cubes and thus may become impractical.

In [2], the Authors proposed an analytical method for calculation of the autocorrelation function for functions represented by disjoint cubes; the first of the above objections may therefore be withdrawn since it was caused by the algorithmic character of other methods. The second objection seems more problematic since it is connected with the nature of logic functions. Fortunately, there are at least two ways to overcome the second objection. These ways are:

1. Using decomposition techniques. Indeed, it is possible to reduce the complexity of the function's representation by partitioning the function into a number of sub-functions each having a smaller number of disjoint cubes. Such an approach was examined in [6] where the initial function was presented as a superposition of a number of so-called "trapeze" functions comprising disjoint cubes of a smaller rank.
2. Applying the methods based on disjoint cubes to a specific class of logic functions for which the disjoint cubes representation is "natural". In most cases we deal with logic functions which are not "randomly generated" but developed by humans [7]. Since that, an enormously wide class of logic functions comprises the functions which somehow reflect cognitive abilities of humans in creating a description of the technological environment. Such a description usually has the algorithmic style and, consequently, the corresponding logic functions reflect this algorithmic nature [8]. In turn, the algorithmic nature is basically the desired property of disjointness. One of the well known examples of such a kind of descriptions is an Algorithmic State Machine (ASM) chart description. Development of the ASM chart requires thinking in a form of the ASM paths, each corresponding to a specific disjoint cube.

Other examples of the disjoint cubes representations are Binary Decision Diagrams (BDDs) [9] and multi-terminal BDDs (MTBDDs) [10]. BDD and MTBDD are data structures that are widely used in logic synthesis and verification. Description of such diagrams can actually be used as a universal concept on different conceptual levels of systems representation. Indeed, the MTBDD may be considered as a form of systems' specification, as a form of representation of a logic function and even as a certain model of the system's implementation. The present work has been triggered by the visual and cognitive closeness of the above two concepts

(ASM and MTBDD), as well as by their direct relations with the sequential circuits. It looks indeed reasonable and interesting to study techniques of linearization on a class of functions, which seem suitable for linearization in advance, by their nature.

One of the well known tasks in synthesis of sequential circuit is a task of states assignments. The states assignment versus the efficient linearization was deeply studied in [11]. However, the most popular state assignment being a so-called *1-hot* encoding was never examined from the point of its implementability would a linearization technique be applied. We pay a specific attention to this issue in the paper. Specifically, we propose and examine the *1-hot* assignment for linearized, ASM based sequential circuits SC. We describe a decomposition approach for implementing such SC. The approach is evaluated on a number of benchmarks.

The paper is organized as follows.

Section 2 recalls the basics of the linearization techniques, as well as the Algorithmic State Machines base notation and their connection to sequential circuits. Section 3 introduces a newly proposed concept of a Linearized ASM and describes a corresponding architecture of a Linearized SC. Section 4 introduces a piece-wise decomposition of ASM based sequential circuits. Experimental results of the piece-wise linearization of benchmark sequential circuits are presented in Section 5. The summary of the research is given in Section 6.

2 Definitions and Related Works

Two main fields form the background of the present study: Linearization Techniques and Synthesis of SC based on the Algorithmic State Machine notation. In this section we will recall some concepts from both of these fields.

2.1 Linearization of logic functions

Analysis and synthesis methods based on the properties of the Walsh spectrum of a Boolean function and/or on its autocorrelation function are referred to as spectral techniques. In this paper, we consider spectral techniques for performing linear decomposition of a system of Boolean functions (multiple-output function). The initial multiple-output function is presented as a superposition of a linear function and a nonlinear multiple-output function f_σ where $f(x) = f_\sigma(\sigma x)$. The linear function σ is determined by using the autocorrelation values of the initial function so that the linearly transformed function f_σ minimizes a certain cost (complexity) function. As in many works related to minimization of Binary Decision Diagrams (BDDs), the cost functions considered hereby are expressed by the number of nodes in the BDD, the number of paths and the Average Path Length (APL) of the BDD.

The cost functions reflect the required memory size, the testing complexity and average time of the logic simulations [12].

The linearization requires computation of the autocorrelation values of a Boolean function. The computation of the autocorrelation values can be performed either by a straightforward technique based on the definition of the autocorrelation function, or by using the Wiener-Khinchin theorem with complexity $O(N2^N)$ where N is the number of input variables. For functions of a large number of inputs (> 20) these approaches may be impractical. Nevertheless, for most of the applications, e.g. for FSMs derived from ASMs, the combinational part may be represented as a set of M disjoint cubes of small sizes $M \ll 2^M$. In these cases the autocorrelation function for the linearization can be calculated by using the algorithms presented in [2, 3], with the computational complexity proportional to M^2. A different approach to the problem of linearization of large systems represented by a set of non-disjoint cubes is a parallel decomposition of the function into sub-functions, combined with the linearization of each sub-function [6]. Preliminary results demonstrating the potential of this approach were presented in [13].

2.2 Specification of sequential circuits by algorithmic state machine chart

Consider a system that comprises a control unit and an operational unit (a data-path). The operational unit contains a number of computing elements, while the control unit produces vectors of control binary signals forming the set $Y = \{y_1, \ldots, y_N\}$ and forcing execution of operations in the operational unit. These control signals constitute Boolean functions of input variables $X = \{x_1, \ldots, x_L\}$ which arrive from the operational unit, and of so-called state variables $Z = \{z_1, \ldots, z_U\}$ of internal states of the control unit. Interaction between the control unit and the operational unit may be described algorithmically and expressed graphically in a form of an Algorithmic State Machine chart.

An ASM is a directed connected graph containing an initial vertex, a final vertex, a finite set of operator vertices and conditional vertices. Each conditional vertex contains a single logical condition from the set $X = \{x_1, \ldots, x_L\}$. Each operator vertex contains a specific output vector. Let $\{Y_0 \ldots, Y_{Q-1}\}$ be the set of such vectors, $Y_0 = (0, \ldots, 0)$ is the "empty" of "null" vector.

One of the important features of an ASM chart is its equivalence to a finite state machine (FSM). Both ASM and FSM representations are convenient forms for description of sequential circuits (SC). We say that an FSM *implements* a corresponding ASM. We mean that the ASM chart is an initial specification of a system, while the FSM is a mathematical model of the system and can be transformed to a certain hardware description (netlist, VHDL etc.). A simple procedure allows

transforming any ASM into the FSM form by associating so-called paths of the ASM with transitions of the corresponding FSM [8, 14]. The paths, in turn, are defined by using so-called *marks* on the ASM. Each of the marks corresponds to a certain state of the FSM. An example of the marked ASM with logical conditions $X = \{x_1, \ldots, x_4\}$, control signals $Y = \{y_1, \ldots, y_8\}$, output vectors $\{Y_0, \ldots, Y_7\}$ and marks (FSM states) $\{a_0, \ldots, a_4\}$ is shown in Figure 1.

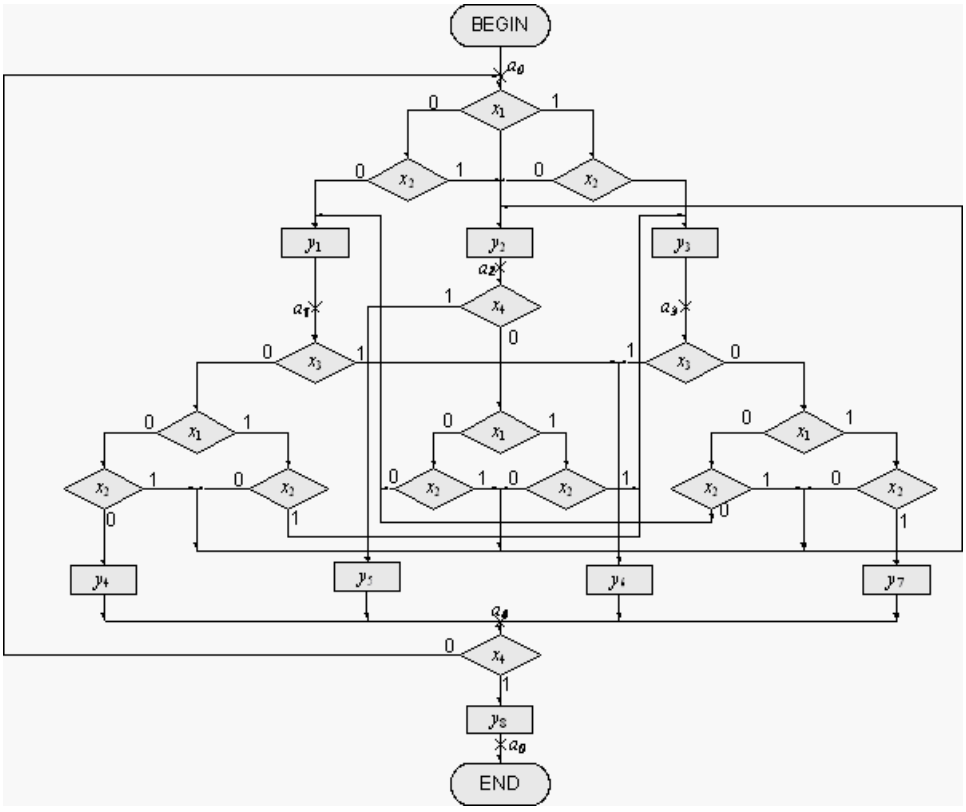

Fig. 1. Marked ASM.

After performing a state assignment, a structural table of the FSM may be constructed. The structural table is presented in Table 1, in the form of a list of the disjoint cubes.

Columns of the table relate to: an input portion of the FSM ("Input variables" and "State variables" columns) and to an output portion of the FSM ("Next state" and "Output functions" columns). Each row of the table corresponds to a specific transition of the FSM or to a certain path within the ASM from Figure 1. The structural table of the FSM represents a system of logic functions or a multiple-

Table 1. Structural table of the FSM corresponding to ASM for Figure 1.

Input variables				State variable				Next state var.				Output variables							
x_1	x_2	x_3	x_4	a	z_1	x_2	x_3	a	d_1	d_2	d_3	y_1	y_2	y_3	y_4	x_5	x_6	x_7	x_8
0	0	-	-	a_0	0	0	0	a_1	0	0	1	1	0	0	0	0	0	0	0
0	1	-	-	a_0	0	0	0	a_2	0	1	0	0	1	0	0	0	0	0	0
1	0	-	-	a_0	0	0	0	a_2	0	1	0	0	1	0	0	0	0	0	0
1	1	-	-	a_0	0	0	0	a_3	0	1	1	0	0	1	0	0	0	0	0
0	0	0	-	a_1	0	0	1	a_4	1	0	0	0	0	0	1	0	0	0	0
0	1	0	-	a_1	0	0	1	a_2	0	1	0	0	1	0	0	0	0	0	0
1	0	0	-	a_1	0	0	1	a_2	0	1	0	0	1	0	0	0	0	0	0
1	1	0	-	a_1	0	0	1	a_3	0	1	1	0	0	1	0	0	0	0	0
-	-	1	-	a_1	0	0	1	a_4	1	0	0	0	0	0	0	0	1	0	0
0	0	-	0	a_2	0	1	0	a_1	0	0	1	1	0	0	0	0	0	0	0
0	1	-	0	a_2	0	1	0	a_2	0	1	0	0	1	0	0	0	0	0	0
1	0	-	0	a_2	0	1	0	a_2	0	1	0	0	1	0	0	0	0	0	0
1	1	-	0	a_2	0	1	0	a_3	0	1	1	0	0	1	0	0	0	0	0
-	-	-	1	a_2	0	1	0	a_4	1	0	0	0	0	0	0	1	0	0	0
0	0	0	-	a_3	0	1	1	a_1	0	0	1	1	0	0	0	0	0	0	0
0	1	0	-	a_3	0	1	1	a_2	0	1	0	0	1	0	0	0	0	0	0
1	0	0	-	a_3	0	1	1	a_2	0	1	0	0	1	0	0	0	0	0	0
1	1	0	-	a_3	0	1	1	a_4	1	0	0	0	0	0	0	0	0	1	0
-	-	1	-	a_3	0	1	1	a_4	1	0	0	0	0	0	0	0	1	0	0
0	0	-	0	a_4	1	0	0	a_1	0	0	1	1	0	0	0	0	0	0	0
0	1	-	0	a_4	1	0	0	a_2	0	1	0	0	1	0	0	0	0	0	0
1	0	-	0	a_4	1	0	0	a_3	0	1	0	0	1	0	0	0	0	0	0
1	1	-	0	a_4	1	0	0	a_3	0	1	1	0	0	1	0	0	0	0	0
-	-	-	1	a_4	1	0	0	a_0	0	0	0	0	0	0	0	0	0	0	1

output function of the sequential circuit.

Multiple-output functions, derived from the ASM-based SC, form a specific class of functions. This class can be characterized by the following properties:

1. The multiple-output functions are defined by a set of disjoint cubes since all transitions of the ASM based SC are pair-wise disjointed.
2. The function is defined on cubes of a small rank (large cubes) since the majority of the SC transitions depend on a limited subset of input variables $X = \{x_1, \ldots, x_L\}$.
3. The multiple-output function has a limited number of different output vectors. The set of these vectors is known in advance.

This specific class of functions is in the focus of our study. Such functions correspond to a wide class of controllers that are used both in microprocessors and in many other applications. We utilize the above properties for reducing the compu-

tational complexity of the linearization, as well as for decomposing the system for its optimization.

3 Architecture of Linearized Sequential Circuit

The conventional SC having W states consists of two parts: a set of U, $(U \geq \lceil \log_2 W \rceil)$ memory elements and a combinatorial part implementing a multiple-output Boolean functions of $L+U$ input variables and of $N+U$ output variables, where L is the number of inputs variables $\{x_1,\ldots,x_L\}$, N is the number of output variables $\{y_1,\ldots,y_N\}$ and U is the number of next state variables $\{d_1,\ldots,d_U\}$ which equals to the number of bits of the encoded state $\{z_1,\ldots,z_U\}$ variables to be stored in the memory.

We propose a new architecture, called a *linearized* SC (LSC), for sequential circuits derived from ASMs. The block diagram of the linearized SC is shown in Figure 2.

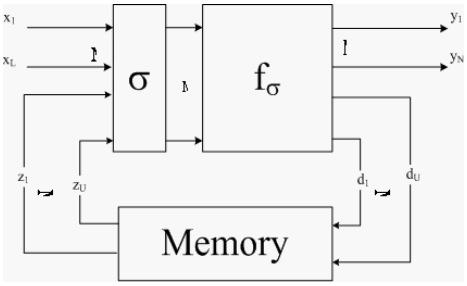

Fig. 2. Architecture of the linearized sequential circuit.

The task of synthesis of the LSC is to define a linearized BDD representing the combinatorial part of the SC. This task comprises three sub-tasks: a) a state assignment; b) an optimized linear transformation; c) an optimized synthesis of a BDD of the combinational part for minimization of the implementation cost of the overall system.

We do not deal with the problem of the state assignment for optimal linearization. This problem was deeply investigated in [15]. Nevertheless, we do apply a number of different states assignments to examine efficiency of spectral techniques for optimizing multiple-output functions derived from the ASM-based SC description. Particularly, we are interested in the minimal length assignment, as well as in the *1-hot* states assignment. The next section deals with the *1-hot* states encoded FSMs.

The problem of selecting an optimized σ is strongly connected with a form of

representation of the initial system of logic functions. Since the system is defined by disjoint cubes the procedure of selecting an appropriate σ has a complexity M^2 (M - the number of cubes).

In this paper, we propose a method for synthesis of an optimized BDD by using a decomposition technique. We call this technique a *piece-wise linearization of sequential circuits*. This technique is especially suitable for implementation of the *1-hot* encoded FSM. The next section is devoted to the piece-wise linearization technique.

4 Parallel Decomposition and Piece-wise Linearization of Sequential Circuits

Presently, the *1-hot* assignment is the most popular state assignment in the SC design. A multiple-output function corresponding to the *1-hot* encoded SC is defined on a set of non-disjoint cubes. Since that, conventional techniques for transforming such functions into their BDD form are ineffective. It is correct both for the linearization based techniques, and for other techniques. In this paper, we propose a specific decomposition technique that allows optimizing the transformation of multiple-output functions corresponding to the *1-hot* encoded SCs into their MTBDD form.

4.1 Piece-wise linearization of multiple-output functions

In [6], a piece-wise linearization technique was proposed. This technique comprises a step of parallel decomposition of the initial set of cubes into a number of components (subsets of cubes), followed by a step of independent linearization of the components. The resulting piece-wise linearized network is directly mapable onto a special type of a binary graph called Parallel Multi-Terminal BDD (PMTBDD). The PMTBDD is constructed by combining component MTBDDs using: a serial operation (i.e. replacing one terminal node of an MTBDD with another MTBDD), and a parallel operation (i.e. connecting roots of two or more MTBDDs). An efficient algorithm for construction of an optimized PMTBDD was proposed in [13]. The decomposition algorithm includes partitioning of the set of cubes of the function into a number of components. The partitioning is followed by a recursive decomposition of the components into a common header and a set of fragments. The efficiency of the decomposition algorithm, in terms of the number of MTBDD nodes, was demonstrated on a number of benchmarks.

The aim of the parallel decomposition step is creating a PMTBDD for an arbitrary multiple-output function. The decomposition algorithm is started with the

partitioning of the set of cubes representing the ON-set of the function, into a set of logic blocks. It is followed by the hierarchical separation of the blocks for a common header and a set of block fragments. The separation is accomplished by extracting a set of common factors (so-called prefixes) from a subset of the original set of cubes.

Let us formulate a number of definitions necessary for explaining the decomposition procedure.

We will call *a prefix* any part of a cube. A subset of cubes comprising prefixes with at least one common variable form a *block*. The cubes of the original set, not included in the block, are called a remainder.

A set of the prefixes of the block defines a *block header*. A set of cubes having exactly the same (common) prefix is called ıfamily. A set of cubes obtained by extracting the common prefix from a *family* forms a *tail*.

The block header is selected in such a way as to provide minimization of the resulting PMTBDD. We select the block header using the following criteria: 1) maximal increase of *density* (minimizing the percentage of "dont cares" in the block); 2) maximal suitability for further linearization of the block.

By is nature, the block header is a logic function, ON-set of which is a superset of ON-sets of the block. The block will be implemented as an MTBDD whose internal nodes are associated with the prefix variables. The terminal nodes of the MTBDD represent the tails, each to be implemented as a separate MTBDD. Each of the tails and the remainder will be each recursively decomposed in the same above described way, until no further decomposition is possible. A detailed formal algorithm of a single iteration of the decomposition is presented in the Appendix.

An important feature of the decomposition method is its ability to benefit from any other optimization method the user may wish to apply - Sifting, *K*-Procedure, etc [1, 4]. These can be applied to the component MTBDDs, and may further reduce the total diagram's size. A PMTBDD with the linearized blocks is denoted LPMTBDD.

The following example illustrates how the decomposition works.

Example 1: Let a Multiple-output function is defined by the set cubes presented in Table 2.

BDD, LTBDD, PMTBDD and LPMTBDD for the example are presented in Figures 3-6 correspondingly.

Terminal nodes of MTBDDs in Figures 3-6 are marked by decimal numbers of the corresponding outputs. A standard implementation of the MTBDD of our example, in the form of an ordered MTBDD, is presented in Figure 3. Figure 4 shows a linearized MTBDD corresponding to the Example. A PMTBDD obtained by the

Table 2. The list of cubes of the example.

#	x_0	x_1	x_2	x_3	x_4	F_0	F_1	F_2	F_3
0	0	1	-	0	-	1	0	0	0
1	0	1	-	1	-	0	1	1	0
2	-	-	1	-	0	0	0	1	1
3	-	-	-	0-	1	0	0	0	1
4	1	0	-	1	-	1	1	0	0

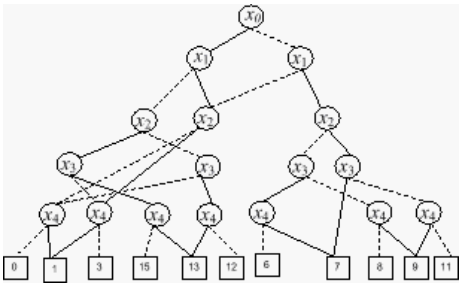

Fig. 3. Straightforward implementation of MTBDD of the Example.

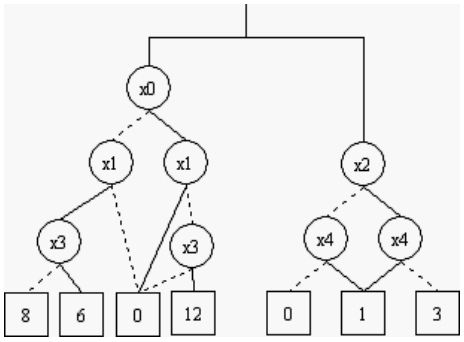

Fig. 4. Linearized MTBDD for the Example.

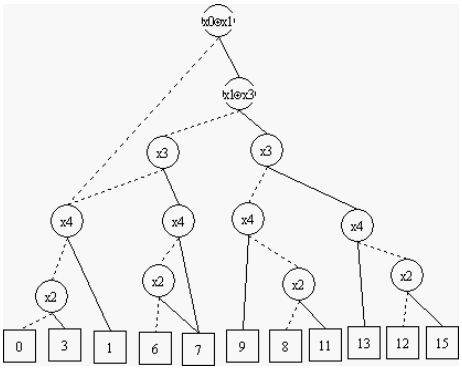

Fig. 5. PMTBDD corresponding to the Example.

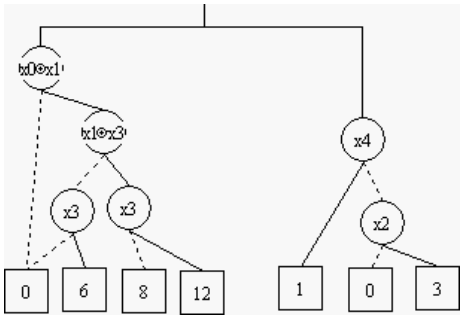

Fig. 6. Linearized PMTBDD of the Example.

proposed decomposition method, is shown in Figure 5. The PMTBDD comprises two portions the block (left) and the remainder (right). The portions are assembled by the newly introduced parallel connection of MTBDDs. Figure 6 shows the piece-wise linarized MTBDD. Notice that sets of terminal nodes in the PMTBDD and in the standard MTBDD are not the same. It is a result of the concatenation operation between the original terminal nodes. The concatenation is calculated as the OR function between corresponding output vectors [6]. For example, terminal node 7 in the MTBDD (Figure 3) corresponds to two terminal nodes 3 and 6 in the

PMTBDD (Figure 5). Such cases reflect the non-disjoint property, as in cubes 1 and 2 from Table 2.

The Example demonstrates the significant reduction of the number of MTBDD nodes. Indeed, the standard MTBDD has 17 non-terminal nodes (NTNs), after the linearization this number is reduced to 12 nodes (and 2 XOR gates). The PMTBDD has 8 NTNs, the linearization further reduces that number to 6 NTNs (and 2 XOR gates).

5 Piece-Wise Linearization of 1-hot Hncoded FSMs

The computational complexity of the above method [13] makes it impractical for arbitrary functions of a large number of variables. Nevertheless, a class of *1-hot* encoded FSMs should be considered a special class of FSMs from the point of our task. Such FSMs seem extremely suitable for the piece-wise linearization, since the cubes of the corresponding multiple-output functions are non-disjoint, while the cubes corresponding to a single FSM state are disjoint. The FSMs may be presented in a form of Shared MTBDDs (SMTBDDs) comprising a number of parallel MTBDDs each corresponding to a certain FSMs state. Although such a representation is much better than the straightforward classical MTBDD representation, it may be even improved by using the piece-wise decomposition. Specifically, we propose a method for decomposition of a multiple-output function corresponding to a *1-hot* encoded FSM.

Let Table 2 defines a benchmark of a *1-hot* encoded FSM. It is seen in the table that there are a lot of non-disjoint pairs of cubes in this function (for example the first and the fifth cubes are non-disjoint, as well as many other pairs). This fact prevents efficient transforming the benchmark to its MTBDD form by using conventional methods. However, the above-described piece-wise linearization gives an elegant solution presented in Figure 7.

The optimized parallel MTBDD corresponding to the function from Table 2 is shown in Figure 7. For the sake of simplicity, only the ON-sets of the output function and of the next state function are marked within the terminal nodes. Symbol stands for the empty output vector comprising zeros only.

The standard MTBDD implementation for the example requires 212 non-terminal nodes; the proposed piece-wise linearized solution (comprising a number of parallel MTBDDs corresponding to the FSMs states) requires 23 non-terminal nodes. However, the method enables obtaining the structure of the parallel MTBDD, shown in Figure 7, which has only 13 non-terminal nodes. This impressive result shows how MTBDDs can be improved by using specific properties of

Table 3. Table 3. Structural table (multiple-output function) with *1-hot* state encoding.

Input variables				State variables					Next state var.					Output functions							
x_1	x_2	x_3	x_4	z_1	z_2	z_3	z_4	z_5	d_1	d_2	d_3	d_4	d_5	y_1	y_2	y_3	y_4	y_5	y_6	y_7	y_8
0	0	-	-	1	-	-	-	-	0	1	0	0	0	1	0	0	0	0	0	0	0
0	1	-	-	1	-	-	-	-	0	0	1	0	0	0	1	0	0	0	0	0	0
1	0	-	-	1	-	-	-	-	0	0	0	1	0	0	1	0	0	0	0	0	0
1	1	-	-	1	-	-	-	-	0	0	1	0	1	0	0	1	0	0	0	0	0
0	0	0	-	-	1	-	-	-	0	0	0	0	1	0	0	1	1	0	0	0	0
0	1	0	-	-	1	-	-	-	0	0	1	0	0	0	1	0	0	0	0	0	0
1	0	0	-	-	1	-	-	-	0	0	1	0	0	0	1	0	0	0	0	0	0
1	1	0	-	-	1	-	-	-	0	0	0	1	0	0	0	1	0	0	0	0	0
-	-	1	-	-	1	-	-	-	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	-	0	-	-	1	-	-	0	1	0	0	0	1	0	0	0	0	0	0	0
0	1	-	0	-	-	1	-	-	0	0	1	0	0	0	1	0	0	0	0	0	0
1	0	-	0	-	-	1	-	-	0	0	1	0	0	0	1	0	0	0	0	0	0
1	-	-	0	-	-	1	-	-	0	0	0	1	0	0	0	1	0	0	0	0	0
-	-	-	1	-	-	1	-	-	0	0	0	0	1	1	0	0	0	1	0	0	0
0	0	0	-	-	-	-	1	-	0	1	0	0	0	0	0	0	0	0	0	0	0
0	1	0	-	-	-	-	1	-	0	0	1	0	0	0	1	0	0	0	0	0	0
1	0	0	-	-	-	-	1	-	0	0	1	0	0	0	1	0	0	0	0	0	0
1	1	0	-	-	-	-	1	-	0	0	0	0	1	0	0	0	0	0	0	1	0
-	-	1	-	-	-	-	1	-	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	-	0	-	-	-	-	1	0	1	0	0	0	1	0	0	0	0	0	0	0
0	1	-	0	-	-	-	-	1	0	0	1	0	0	0	1	0	0	0	0	0	0
1	0	-	0	-	-	-	-	1	0	0	1	0	0	0	1	0	0	0	0	0	0
1	1	-	0	-	-	-	-	1	0	0	0	1	0	0	0	1	0	0	0	0	0
-	-	-	1	-	-	-	-	1	1	0	0	0	0	0	0	0	0	0	0	0	1

the ASM based SCs.

6 Experimental Results

In this section we present experimental results obtained on a number of industrial benchmarks provided by different high-tech companies. In most cases these benchmarks describe micro-controllers as an embedded part of various computer systems. Experiments were carried out by using our especially developed software. The results of the experiments are presented in Tables 4-6. For some benchmarks, the calculations required were too huge and results were not obtained. Such cases are highlighted in the tables.

First six columns of Table 4 comprise titles of benchmarks and their parameters. The 7-th, 8-th, and 9-th columns indicate the number of MTBDD nodes in the conventional, decomposed and piece-linearized realizations respectively. The last col-

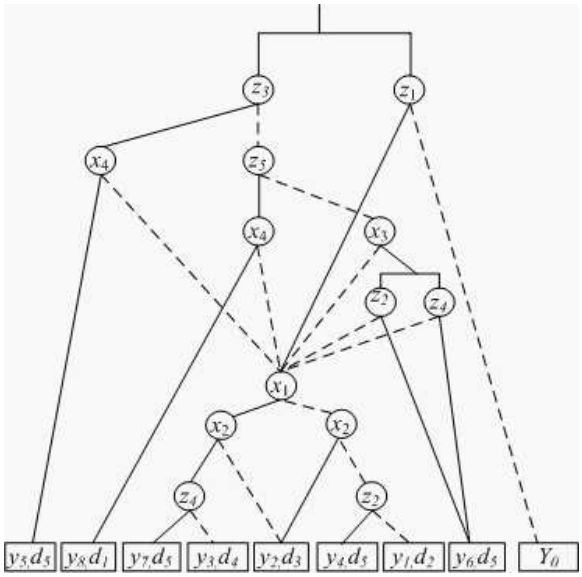

Fig. 7. Parallel MTBDD implementing a *1-hot* encoded function from Table 2.

umn shows reduction of the number of nodes (in percents) for the linearized implementations in comparison with the number of nodes in the conventional MTBDD: $\Omega_C = (1 - C/B)100$ %. Notice, that the experiment utilized the original (provided by the companies) state assignment. In most cases the number of state bits U is greater than $U = \lceil \log_2 W \rceil$. Table 4 indicates that linearization of the benchmark function gives the reduction of about 30MTBDD nodes.

Table 4. Benchmarks results for the original state encoding.

Benchmark	Inputs L	Outputs N	State W	State bits U	Cubes	MTBDD	Linearized MTBDD	Ω %
s208	11	2	18	8	153	157	116	26.1
s298	3	6	218	14	1096	-	-	-
s386	7	7	13	6	64	126	91	27.8
s420	19	2	18	16	137	210	167	20.5
s510	19	7	47	6	77	111	75	32.4
s820	18	19	25	5	232	251	178	29.1
s832	18	19	25	5	245	311	216	30.5
s1494	8	10	48	6	250	565	240	57.5

Table 5 reflects experiments related to the Linearization of MTBDDs of the same benchmarks, where an optimized minimal length state assignment is applied. The last column indicates the number of nodes in MTBDDs corresponding to the

linearized benchmark functions. All MTBDDs of Table 5 were constructed by using a conventional technique (without decomposition). The table illustrates the high sensitivity of the number of MTBDD nodes to different state encodings. Indeed, even applying the minimal length state encoding results in the significant reduction of the number of nodes of the obtained MTBDD.

Table 5. Benchmark results for the optimal minimallength state assignment.

Benchmark	State bit	Nodes
s208	5	90
s298	8	430
s386	4	53
s420	5	90
s510	6	76
s820	5	154
s832	5	154
s1494	6	261

Table 6. Benchmark results for the 1-hot state assignment.

Benchmark	MTBDD	Parallel Decomposition	Piecewicw Linearization	Ω %
s208	1864	146	137	6.2
s386	447	92	82	11
s420	938	142	138	2.8
s510	>1 600	106	89	16
s820	>13 000	352	328	6.8
s832	>10 000	358	335	6.4
s1488	>14 000	412	401	2.7
s1494	>32 000	399	377	5.5

Table 5. Benchmark results for the optimal minimal length state assignment

Table 6. Benchmark results for the *1-hot* state assignment

Results of the *1-hot* state assignment experiments are presented in Table 6. Columns of this table are similar to those of Table 4. The last column indicates the reduction, achieved by the piece-wise linearization in comparison with the parallel decomposition: $\Omega = (1 - P/C)100\%$. The conventional MTBDDs in most of the cases appeared too big, and even could not be constructed in a reasonable time period. However, for all cases of Table 6, the both proposed decomposition methods provide very good solutions. The last column of the table shows the reduction in the number of nodes, provided by the piece-wise linearization in comparison with the parallel decomposition, which is of about 10%.

7 Conclusions

The paper describes an approach for synthesis of sequential circuits (SCs) described by their Algorithmic State Machine (ASM) chart. Such SCs may be synthesized efficiently since their corresponding logic functions are defined by disjoint cubes. Moreover, the ASM form of the SC representation is close to the BDD form. We consider that the optimal MTBDD representation as the criterion of the SC optimization. We propose a method for transformation of an initial multiple-output function, derived from ASM, into an optimized Multi-Terminal BDD. The method

is based on a newly introduced concept of the Linearized SC and comprises the following steps: a) transforming an initial ASM chart into a corresponding multiple-output function, followed by a specific states assignment; b) either linearization of the function or decomposition of the function followed by linearization of its components.

Experimental results, carried out on a set of benchmarks, allow concluding the following:

- We have discovered a wide class of multiple-output functions on which linearization techniques work extremely efficiently. This class comprises, *inter alia*, so-called "human-produced" functions having the algorithmic structure.
- We have opened a new way to linearize functions of a large number of input variables for the above-specified class of the multiple-output functions. It has become possible since the functions of that class are derived from an ASM and, consequently, are defined by disjoint cubes of a small rank.
- Representations of the initial ASM-based functions in the MTBDD form are very sensitive to the states encoding, which requires a special attention of researchers.
- The proposed decomposition technique, as well as the piece-wise linearization has proven to be very efficient in the case of *1-hot* states assignment, while conventional techniques have failed on the same benchmarks due to their high complexity.

The Authors believe that it is highly desirable to recognize the class of the functions to be synthesized and to utilize specific properties of the recognized class for optimization of the synthesis. Our study of the ASM-derived functions has allowed extending the plurality of functions that can be efficiently linearized.

Appendix

The decomposition algorithm

The presented below algorithm, details the flow of a particular iteration of the decomposition procedure. The main part of a particular iteration consists of choosing the prefixes for the block. The prefixes are chosen one by one. Each time a prefix is chosen, all the prefixes non-disjoint with this prefix and belonging to different characteristic functions are moved to the remainder. Non-disjoint prefixes belonging to the same characteristic function as the prefix are included in the block. This continues until no more suitable prefixes can be found. Thereafter the iteration

proceeds the tails and constructs the blocks of the MTBDD. The non-trivial tails as well as the remainder serve as inputs to the next iterations of the algorithm.

1: L = Empty list of pairs Prefix, Tail
2: Let R = Set of cubes, Y = vector of integer outputs.
3: Let B = Basic Prefix, T_F = its family, $T_T = T_F \backslash S$ = its tail
4: $L = [L, (B, T_T)]$
5: $T_U = R \backslash (T_F \cup T_R)$
6: $C = \{t : t \in R \text{ and } t \cdot B \equiv 0\}$
7: $T_R = \{t : t \in R \backslash (C \cup T_F)\}$
8: C = COMMON_PARTS(C)
9: **while** $|C| > 0$, **do**
10: Let S =Secondary Prefix
11: T_F = its family
12: $T_T = T_F \backslash S$ = its tail
13: $L = [L, (S, T_T)]$
14: $T_O = \{t : t \in C \text{ and } t \cdot S \equiv 0\}$
15: $T_{NO} = \{t : \ t \in C \backslash T_O\}$
16: $C = T_O$
17: $T_R = T_R \cup T_{NO}$
18: **end while**
Classify and enumerate the tails in L
Construct the Block's BDD from L
19: $L_T = \{l : l \in L, \ \text{Tail } (l) \text{ is trivial } \}$
20: $L_{NT} = \{l : l \in L \backslash L_T\}$
21: **if** the remainder is trivial, **then**
22: $LT = [L_T, \mathit{Remainder}]$
23: **else**
24: $LNT = [L_{NT}, \mathit{Remainder}]$
25: **end if**
26: **for all** the list elements in LT **do**
27: Implement the tail
28: **end for**
29: **for all** the list elements in LNT **do**
30: Recursively call this procedure
31: **end for**

References

[1] M. G. Karpovsky, R. S. Stanković, and J. T. Astola, "Reduction of sizes of decision diagrams by autocorrelation functions," *IEEE Trans. on Computers*, vol. 52, no. 5, pp. 592–606, 2003.

[2] O. Keren, I. Levin, and R. S. Stanković, "Reduction of the number of paths in binary decision diagrams by linear transformation of variables," in *Proc. of 7th International Workshop on Boolean Problems*, 2006, pp. 79–84.

[3] ——, "Linearization of functions represented as a set of disjoint cubes at the autocorrelation domain," in *Proc. of 7th International Workshop on Boolean Problems*, Freiberg, Sept. 2006, pp. 137–144.

[4] D. M. Miller, R. Drechsler, and M. A. Thornton, *Spectral Techniques in VLSI CAD*. Kluwer Academic Pub., 2001.

[5] R. S. Stanković and J. T. Astola, *Spectral Interpretation of Decision Diagrams*. New York, Secaucus, NJ: Springer-Verlag Inc, 2003.

[6] I. Levin, O. Keren, V. Ostrovsky, and G. Kolotov, "Concurrent decomposition of multi-terminal bdds." in *Proc. of 7th International Workshop on Boolean Problems*, Freiberg, Sept. 2006, pp. 129–136.

[7] A. Zakrevskij, *A Common Logic Approach to Data Mining and Pattern Recognition*. Springer, 2006, pp. 1–44.

[8] S. Baranov, *Logic Synthesis for Control Automata*. Dordrecht, Boston, London: Kluwer Academic Publisher, 1994.

[9] R. Drechsler and B. Becker, *Binary Decision Diagrams, Theory and Implementation*. Kluwer Academic, 1998.

[10] M. Fujita, P. McGeer, and J.-Y. Yang, "Formal methods in system design," in *Special Issue on Multi-Terminal Binary Decision Diagrams*. Springer Netherlands, Kluver Academic, Feb. 1997, vol. 10, no. 1, pp. 149–169.

[11] G. D. Micheli, *Synthesis and optimization of digital circuits*. New York, Toronto, London: McGraw-Hill, 1994.

[12] P. C. McGeer, K. L. McMillan, A. Saldanha, and A. Sangiovanni-Vincentelli, "Fast discrete function evaluation using decision diagrams," in *Proceedings of International Conference on Computer Aided Design*, Nov. 1995, pp. 402–407.

[13] I. Levin, O. Keren, G. Kolotov, and M. Karpovsky, "Piecewise linearization of multi-output functions," in *Proc. of the 2006 International Workshop on Spectral Methods and Multirate Signal Processing*, Florence, Italy, Sept.2-3, 2006, pp. 345–353.

[14] J. P. Hayes, *Introduction to Digital Logic Design*. Prentice Hall, 1993.

[15] M. G. Karpovsky, *Finite Orthogonal Series in the Design of Digital Devices*. New York: Jon Wiley, 1976.