Automation and Remote Control, Vol. 65, No. 6, 2004, pp. 1018–1032. Translated from Avtomatika i Telemekhanika, No. 6, 2004, pp. 201–217.

DISCRETE MODELING

Modeling Combinational Circuits Using Linear Word-level Structures

D. V. Popel* and S. N. Yanushkevich**

* *Computer Science Department, Baker University, Baldwin City, USA*
** *Electrical and Computer Engineering Department, University of Calgary, Calgary, Canada*

Received July 31, 2003

Abstract—In many applications of circuit design and synthesis, it is natural and in some instances essential to manipulate logic functions and model circuits using word-level representations and arithmetic operations in contrast to bit-level representations and logic operations. This paper reviews *linear word-level* structures and formulates their properties for combinational circuit modeling. The paper addresses the following problem: *given* a library of gates with their corresponding word-level representations such as linear arithmetic expressions or respective graph structures, *find* a word-level model of an arbitrary combinational circuit/netlist using that library of gates and minimizing memory allocation and time delay requirements. We present a comprehensive study on *linearization* assuming various circuit processing strategies. In particular, we develop a new approach to manipulate linear word-level representations by means of *cascades*. The practical applicability of linear structures and developed algorithms is strengthen by considering the problem of timing analysis. All this is supported by the experimental study on benchmark circuits.

1. INTRODUCTION

In the computer-aided design of integrated circuits, word-level structures are defined as groupings (corteges[1]) of uniform objects: bits, functions, etc. Thus, the hexadecimal representation of a binary number can be considered as a word-level structure obtained by decomposing the binary number into groups of four bits. Word-level structures are described by arithmetic expressions, tabular or graph representations, those parts are classified to be either linear or non-linear. Linearity of word-level structures is important while performing the linear transformation of variables [4], finding linear arithmetic expressions [3], and designing linear word-level decision diagrams [2]. Therefore, a special *modeling mechanism* is required to work with various linear word-level structures.

We observe that the linear word-level modeling can be accomplished in two possible ways. The first way is based on high-level abstract representations such as linear word-level diagrams [2]. Although finding linear sub-parts of a high-level representation is viable[2], there is a little practical impact of such modeling. It has been proved in [3] that a Boolean function can be represented by a set of linear arithmetic expressions. Using graph terminology, this statement means that an arbitrary word-level decision diagram is composed by a set of linear word-level diagrams. However, there is no obvious link between an abstract model like a word-level diagram and a respective circuit. The levelization of a circuit has been proposed in [2] to overcome this difficulty. The levelization helps to identify components which have unique representations in terms of linear word-level diagrams.

[1] The term "cortege" is used more frequently by eastern researches.

[2] Linear word-level diagrams is a border case of classical word-level decision diagrams—linear word-level diagrams can be found for the limited set of logic functions.

0005-1179/04/6506-1018

The second way is based on low-level structural representations and preserves interconnections of components along with the description of their desired behavior. Controversially, the low-level structures fail to describe the behavior of an entire circuit. The focus of our interest is a mixed high- and low-level representation. Within such a mixed-mode modeling, arithmetic expressions and graph structures become high-level abstractions, whereas the networks of gates (netlists) are employed for low-level representations. This paper emphasizes a tradeoff between abstract and structural word-level modeling contrasting novel low-level and mixed-mode representations with known word-level decision diagrams.

Even though our research is based on the theory presented in [3] for arithmetic expressions and results reported in [2] for word-level decision diagrams, we seek an answer to the question on what data structure would be the most efficient for mixed-mode linear modeling. It is also unknown what kind of grouping procedures are required within the framework of mixed-mode modeling. Thus, we distinguish three possible strategies to work with circuit components: gate based, level based, and cascade based. We outline a methodology how to determine various components and introduce the algorithms of circuit levelization and cascading. These strategies result in different characteristics of memory allocation and timing for linear word-level models. Both memory requirements and time delays are essential parameters for simulation and testing of combinational circuits.

The rest of the paper is organized as follows. In Section 2, we collect necessary definitions and basic terminology. Different word-level models for circuit processing are given in Section 3. The solution for the timing analysis problem based on linear word-level models is outlined in Section 4. Section 5 is dedicated to the experimental results on benchmark circuits. Section 6 concludes the paper and provides the directions of our future work.

2. COMBINATIONAL CIRCUITS AND THEIR WORD-LEVEL REPRESENTATIONS

In the following, we consider a circuit as a network of gates, which are solely combinational, and a corresponding multi-output Boolean function f as the mapping $B^n \to B^m$ over the variable set $X = \{x_1, \ldots, x_n\}$, where $B = \{0, 1\}$. Here, n is the number of inputs (variables), and m is the number of outputs. Below we discuss various word-level representations of the function f: (i) arithmetic expressions and their "border cases" of linear arithmetic expressions, and (ii) known word-level graph structures. Throughout the section we seek an answer to the question on what data structure would be the most efficient for the linear word-level modeling of combinational circuits.

2.1. Arithmetic Expressions

It is a well known fact that an arbitrary Boolean function can be represented by an *arithmetic expression*, also referred to as an *arithmetic polynomial* or *algebraic form*, where only arithmetic operations of addition and subtraction are permissible for all terms.

Example 1. Let us consider the half adder as a two-output circuit of two variables x_1 and x_2: $f_1 = x_1x_2$ (carry) and $f_2 = x_1 \oplus x_2$ (sum). The related arithmetic expressions are $f_1 = x_1x_2$ and $f_2 = x_1 + x_2 - 2x_1x_2$.

The fundamental advantage of word-level representations is their ability to describe a multi-output function by a single arithmetic expression. This is achieved by assigning weights to each output as a power of two integers: $f = 2^{m-1}f_1 + \ldots + 2^0 f_m$, where f_i, $i = 1, \ldots, m$, is a single-output function.

Example 2 (continuation of Example 1). Let us find the arithmetic expression for the multi-output function of the half adder. Combining arithmetic expressions $f_1 = x_1x_2$ and $f_2 = x_1 + x_2 -$

$2x_1x_2$ by weighting each output, we obtain a single word-level representation of the half adder: $f = 2f_1 + f_2 = x_1 + x_2$.

It was found for certain circuits, and the half adder is one of them, that their corresponding arithmetic expressions have a simplified format composed of terms with at most one literal. Such expressions are called linear arithmetic expressions and the process of synthesizing them is known as linearization.

2.2. Linear Arithmetic Expressions

As shown above, any function can be represented by either multiple arithmetic expressions (an arithmetic expression for each output) or by a single arithmetic expression (a weighted sum of arithmetic expressions for all outputs). *Linearization* can be defined as a process of transforming an arithmetic expression into a linear one where all product terms are comprised of at most one literal. In general, a *linear arithmetic expression* ζ is given by

$$\zeta = d_0 + d_1x_1 + \ldots + d_nx_n, \tag{1}$$

where $d_0, d_1, \ldots, d_n$ are integer-valued coefficients. Linear arithmetic expressions exploited in this paper have a number of useful properties.

Property 1. The linear arithmetic expression ζ takes only non-negative values. Therefore,

$$\begin{cases} d_0 \geq 0 \\ \left(d_0 + \sum\limits_{j=1}^{n} d_j\right) \geq 0, \quad \text{for} \qquad \forall d_j < 0. \end{cases}$$

Property 2. The linear arithmetic expression ζ has the maximum value of $d_0 + \sum\limits_{j=1}^{n} d_j$, for $\forall d_j > 0$.

It is necessary that the expression ζ carries all required information about the initial function f. Assigning proper values to all variables allows us to use this information for calculating the integer value of ζ, then getting certain bits from the integer value enables the restoration of the original function f. The idea of *masking* permits extracting of that information in the form of the linear expression ζ. Thus for the function f, the masking operator $\Xi^{\xi-\varrho}\{\zeta\}$ extracts the range of bits between ξ and ϱ, and results in $f = \Xi^{\xi-\varrho}\{\zeta\}$.

Example 3 (continuation of Example 2). Applying the masking operator $\Xi^2\{\zeta\}$ to the linear expression $\zeta = x_1 + x_2$ of the half-adder, we extract the most significant bits from the integer-valued vector [0112], and obtain the binary vector [0001] which reflects to f_1. The masking operator $\Xi^1\{\zeta\}$ extracts the binary vector [0110] which corresponds to the output f_2. So, a single linear arithmetic expression can be deployed to describe several single-output functions.

The following property characterizes the masking operator $\Xi^{\xi}\{\zeta\}$ for the single-output function f.

Property 3. For the single-output function f, the masking operator $\Xi^{\xi}\{\zeta\}$ selects ξ-th bit, $\xi = 1, \ldots, \left\lceil \log_2 \left(d_0 + \sum\limits_{j=1}^{n} d_j\right) \right\rceil$ for $\forall d_j > 0$, this operation is equivalent to bit-shifting ξ times.

As stated earlier, the process of linearization results in a single linear arithmetic expression ζ. However, the problem of finding the appropriate coefficients $d_0, d_1, \ldots, d_n$ is NP-hard. Below, we formulate a set of conditions to perform an efficient search for linear arithmetic expressions.

Lemma 1. *To avoid duplicated results while linearizing the single-output function f, the coefficients $d_0, d_1, \ldots, d_n$ should have values within the following bounds (intervals):*

$$\begin{cases} d_0 = 0 \ldots (2^n - 1) \\ d_i = -\left\lfloor \dfrac{2^n}{n} \right\rfloor \ldots \left\lfloor \dfrac{2^n}{n} \right\rfloor, \qquad i = 1, \ldots, n. \end{cases}$$

The coefficients which have values beyond these bounds would result in functions obtained by permutation of already generated ones.

Example 4 (continuation of Example 3). The carry function f_1 is extracted by applying the masking operator $\Xi^2\{\zeta\}$ to the linear arithmetic expression $\zeta = x_1 + x_2$. However, the same function f_1 can be derived from the following linear expressions and respective masking operators: $\Xi^3\{2x_1 + 2x_2\}$, $\Xi^3\{2 + x_1 + x_2\}$, $\Xi^4\{2 + 2x_1 + 2x_2\}$, $\Xi^4\{4x_1 + 4x_2\}$, etc.

Theorem 1. *For the linearization of the single-output function f, it is sufficient to have the coefficients $d_0, d_1, \ldots, d_n$ such that* $\left\lceil \log_2 \left(d_0 + \sum\limits_{j=1}^{n} d_j \right) \right\rceil \le n$, *for* $\forall d_j > 0$.

Proof. According to Lemma 1, the maximum value of the coefficient d_0 is $2^n - 1$, and the maximum values of other coefficients are $\left\lfloor \dfrac{2^n}{n} \right\rfloor$. Therefore, $\sum\limits_{j=1}^{n} d_j \le 2^n$. If $d_0 = 0$ then $\left\lceil \log_2 \left(d_0 + \sum\limits_{j=1}^{n} d_j \right) \right\rceil \le \log_2 2^n$, which results in $\left\lceil \log_2 \left(d_0 + \sum\limits_{j=1}^{n} d_j \right) \right\rceil \le n$.

Example 5. There are $2^{2^2} = 16$ single-output functions of two variables. All these functions can be generated by applying the masking operator Ξ^ξ, where $\xi = 1$ or $\xi = 2$, to the following list of linear arithmetic expressions: $x_1 + x_2$, $1 + x_1 + x_2$, $1 + x_1 - x_2$, $1 - x_1 + x_2$, $2 - x_1 - x_2$, $2 + x_1 - x_2$, $2 - x_1 + x_2$ and $3 - x_1 - x_2$.

This set of conditions can be generalized for multi-output functions. We use these conditions to perform a statistical experiment on limitations of linear arithmetic expressions.

2.3. Limitations of Linear Arithmetic Expressions

We conducted a statistical experiment to answer the question on how big is the set of functions which can be represented by linear arithmetic expressions and permissible masking operators. We considered the entire spectrum of $(2^{2^n})^m$ Boolean functions with up to eight input variables ($n = 8$) and up to eight outputs ($m = 8$). This experiment is based on *naive linearization* which generates coefficients d_i and permissible masking operators in a linear form. The naive linearization can handle only certain classes of functions.

Example 6 (continuation of Example 1). Let us change the order of the weighted outputs f_1 and f_2. The resulting arithmetic expression $f = 2f_2 + f_1 = 2x_1 + 2x_2 - 3x_1x_2$ is non-linear.

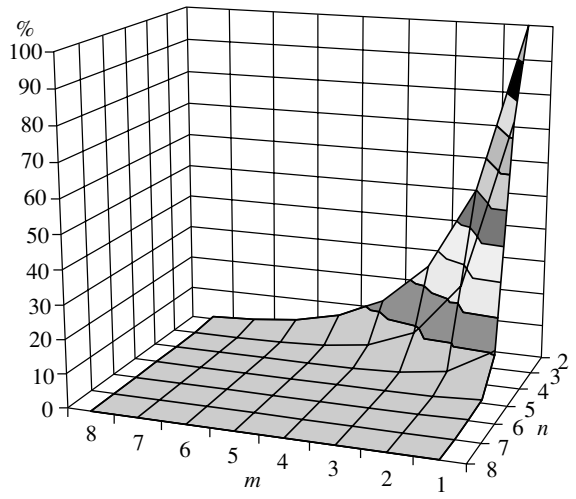

Fig. 1. A plot of the percentage of successfully linearized circuits with up to eight inputs and up to eight outputs, axes n and m respectively.

The search space of $(2^{2^n})^m$ is much larger than $m!$ for the naive linearization. "Success" denotes the case where a function extracted from the linear expression is identical to a given one. Figure 1 plots the percentage of successfully linearized functions by such an exhaustive search.

Observations. (1) All single output functions of two variables have been successfully linearized through an exhaustive search for linear expressions. It corresponds to the pick, i.e., 100% of success. For single output functions of three variables, the percentage of successfully linearized functions is about 74%.

(2) Allowing permutations for multi-output functions, we reveal that only 51% of two-input two-output functions are linearized. With the increase of the number of outputs, the percentage of successfully linearized functions vanishes. Besides, the run-time of such an exhaustive search for functions with more than four outputs is impractical.

(3) We unveiled that the linearization can be efficiently applied only to functions with less than four inputs and four outputs. This is the motivation of our interest to another approach introduced in [3].

Next we discuss linear word-level representations suitable for creation of netlist models.

3. LINEARIZATION OF NETLISTS

As demonstrated earlier (see, for example, Fig. 1), the linearization in general can be efficiently applied to either certain classes of circuits or any circuits with up to four inputs and four outputs. Following the problem stated above for low- or mixed-mode modeling of combinational circuits, we determine three linearization strategies based on gate, level, and cascade representations.

Gate based. This basic strategy is based on replacing gates in a given netlist by corresponding linear arithmetic expressions. Obviously, the resulting model includes the set of linear arithmetic expressions. Since the number of linear components is the same as the number of gates, this strategy explores limited possibilities of a word-level representation. However, it can be useful as an intermediate model in word-level modeling of small size designs.

Level based. This strategy uses the fact that an arbitrary function can be represented by a set of linear arithmetic expressions. Since practical combinational circuits have a multi-level structure, a linear model can be derived for each of its levels. Thus, a set of linear arithmetic expressions is as large as the number of levels in the circuit [2]. However, the problem of canonicity arises for

such a representation, where different sets of linear models can be synthesize for various partitions of a netlist.

Cascade based. This linearization strategy is based on cascading, where a reasonably small set of gates results in a linear model. However, with an uncertainty of partitioning/cascading process, the problem of canonicity arises as well.

The hypothesis of our study is that different strategies of grouping gates in a given circuit can significantly affect characteristics of a designing linear model. Further analysis of those strategies is based on the following theorem.

Theorem 2. *For an arbitrary variable x and a single output function f described by a linear arithmetic expression ζ and a masking operator Ξ ($f = \Xi\{\zeta\}$), linear expressions can be determined for the following functions:* (i) $f \wedge x$, (ii) $f \vee x$, (iii) $f \oplus x$, *and (iv)* $\overline{f}$.

The proof is given in [3].

Example 7. Let us consider the function f given by the linear arithmetic expression $\zeta = 1 - x_1 + x_2$ and the masking operator Ξ: $f = \Xi^2\{1 - x_1 + x_2\}$. The following set of linear expressions can be found for the basic operations between the function f and the variable x_3: (i) $f \wedge x_3 = \Xi^3\{1 - x_1 + x_2 + 2x_3\}$, (ii) $f \vee x_3 = \Xi^3\{3 - x_1 + x_2 + 2x_3\}$, (iii) $f \oplus x_3 = \Xi^2\{1 - x_1 + x_2 + 2x_3\}$, and (iv) $\overline{f} = \Xi^2\{2 + x_1 - x_2\}$.

3.1. Gate Based Linearization

The gate based linearization has a number of advantageous characteristics for the computer-aided design and simulation:

(i) Memory required for the gate representation is comparable to memory required for the storage of its linear arithmetic expression. Indeed, in a typical netlist, any gate is specified by its inputs, performed function, and the output, whereas a linear model of a gate is comprised of coefficients and a masking parameter.

(ii) This gate based model is the most appropriate for the analysis and simulation of small size circuits.

(iii) The gate based model is considered being flexible for future technologies, where gates are not associated with levels or partitions, but considered as nodes of a three dimensional structure.

All basic gates have their linear word-level counterparts. Moreover, this fact follows from the above theorem.

Corollary 1. *A linear arithmetic expression can be found for any k-input $\mathcal{AND}$, $\mathcal{OR}$ or $\mathcal{EXOR}$ gates.*

The proof is based on the principle of mathematical induction. Thus, n-input basic gates can be represented by the following linear arithmetic expressions [3]:

$$\mathcal{AND} \quad \bigwedge_{i=1}^{n} x_i^{\sigma_i} \quad \text{by} \quad \Xi^m \left\{ 2^{j-1} - n + \sum_{i=1}^{n} (\sigma_i + (-1)^{\sigma_i} x_i) \right\},$$

$$\mathcal{OR} \quad \bigvee_{i=1}^{n} x_i^{\sigma_i} \quad \text{by} \quad \Xi^m \left\{ 2^{j-1} - 1 + \sum_{i=1}^{n} (\sigma_i + (-1)^{\sigma_i} x_i) \right\},$$

$$\mathcal{EXOR} \quad \bigoplus_{i=1}^{n} x_i^{\sigma_i} \quad \text{by} \quad \Xi^1 \left\{ \sum_{i=1}^{n} (\sigma_i + (-1)^{\sigma_i} x_i) \right\},$$

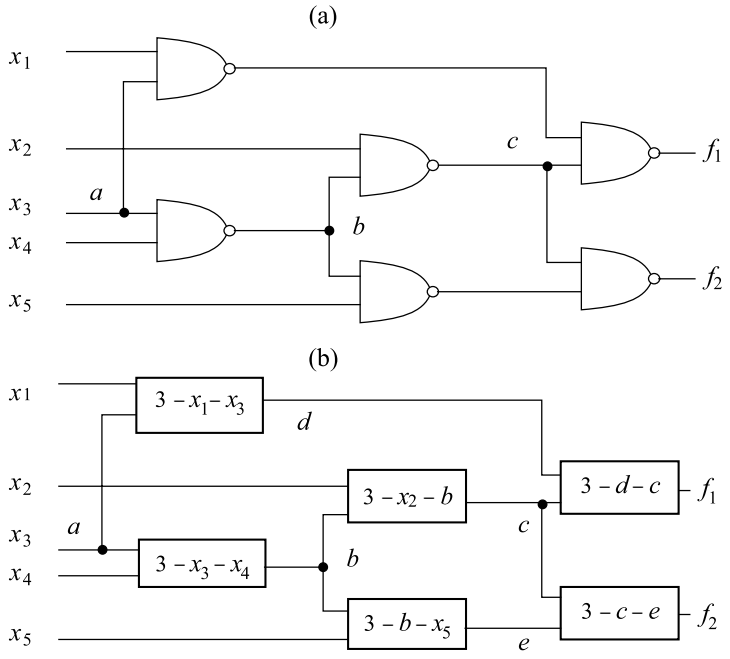

Fig. 2. The benchmark circuit C17 and its gate based linearization.

where

$$j = \lceil \log_2 n \rceil + 1, \quad \text{and} \quad x_i^{\sigma_i} = \begin{cases} x_i & \text{if} \quad \sigma_i = 0 \\ \overline{x}_i & \text{if} \quad \sigma_i = 1. \end{cases} \tag{2}$$

Note that for AND and OR gates, the output is extracted from the most significant bit m as indicated by the masking operator Ξ^m. The output of EXOR gate is extracted from the least significant bit, i.e., Ξ^1.

Although linear expressions can be uniquely assigned to all gates within a combinational circuit and the total memory allocation for the netlist is less than for the original circuit, the practical applicability of this assignment is restricted to the gate level only. The output of the gate based linearization can be used to speed up circuit simulation, since only arithmetic operations are required to obtain the output value.

Example 8. Let us consider the benchmark circuit C17 (Fig. 2a). The gate based linearization is depicted in Fig. 2b.

3.2. Level Based Linearization

The level based linearization has a number of attractive features for combinational circuit modeling, namely:

(i) The model is built directly from the given multi-level circuit; the number of linear arithmetic expressions is equal to the number of levels in a circuit.

(ii) Memory requirements are acceptable for large combinational and even multiple-valued circuits [2].

To implement a linearization algorithm in compliance with the level based processing, level labels must be assigned to all gates. We include an additional field *level* into the standard description of a gate $\mathcal{G}$ to keep the information about its level. Our levelization algorithm works as follows:

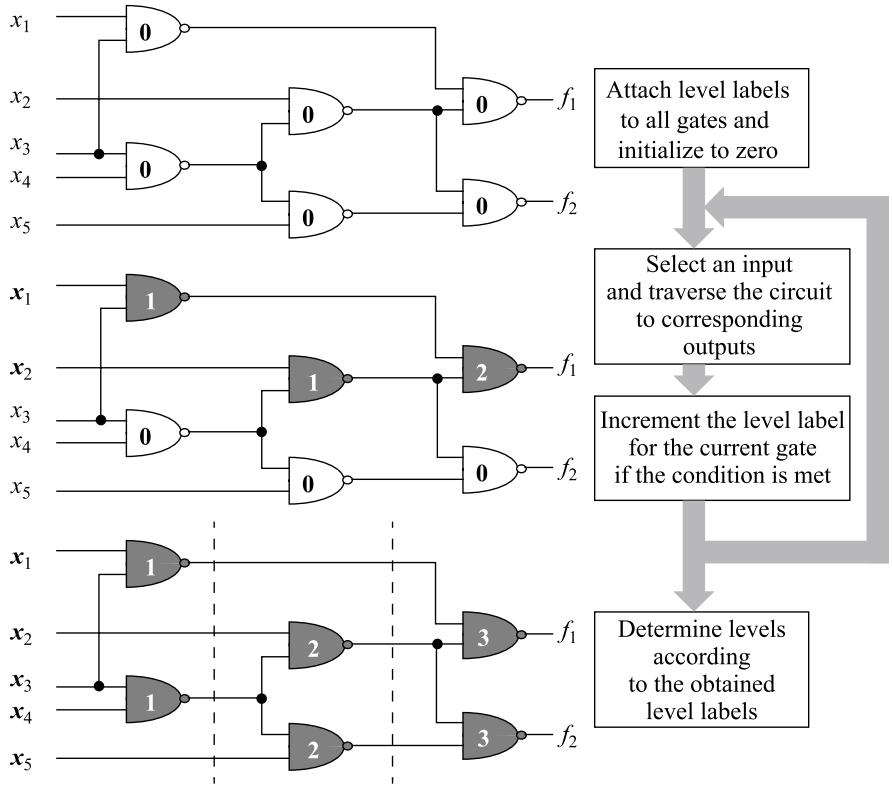

Fig. 3. Levelizing the benchmark circuit C17.

Step 1. Initialize level labels for all gates with zeros: $\mathcal{G}.level = 0$.

Step 2. Consider consequently all paths from each input to outputs. Traverse the circuit from the current input to corresponding outputs.

Step 3. Increment the current level label for a gate if the current level is less than or equal to the level of a previous gate: $\mathcal{G}.level = \mathcal{G}.level + 1$, if $\mathcal{G}.level \leq \mathcal{G}^{prev}.level$.

Step 4. Go to Step 2 until all paths are covered.

Example 9 (continuation of Example 8). Fig. 3 shows the levelization algorithm in action for the benchmark circuit C17. The linear model is depicted in Fig. 6a.

3.3. Cascade Based Linearization

As shown earlier, linear arithmetic expressions can be found for all basic gates. However, all linear expressions obtained for basic gates result in multi-output functions, and the applicability of the above theorem vanishes. Although extracting certain bits from their word-level representations through masking gives single output functions, positions of those are different. We distinguish two sets of basic gates: (i) the set $\{\mathcal{AND}, \mathcal{OR}, \mathcal{NAND}, \mathcal{NOR}\}$ has the masking operator extracting the most significant bits, and (ii) the set $\{\mathcal{EXOR}, \mathcal{NEXOR}\}$ has the masking operator extracting the least significant bits. This differentiation of basic gates allows us to formulate the following definition and a corollary.

Definition 1. A *single-rail cascade* with irredundant inputs is defined having (i) a single wire which connects adjacent gates, and (ii) a unique assignment of each variable to a gate input.

A recent survey of representation methods using cascades can be found in [5]. An example of a single-rail cascade is shown in Fig. 4a. We further classify single-rail cascades as (i) Ξ^2 cascades composed by the set of gates $\{\mathcal{AND}, \mathcal{OR}, \mathcal{NAND}, \mathcal{NOR}\}$, and (ii) Ξ^1 cascades made by the the set of gates $\{\mathcal{EXOR}, \mathcal{NEXOR}\}$.

Corollary 2. *A linear arithmetic expression can be found for either Ξ^1 or Ξ^2 cascade of k-input gates.*

The proof is based on Theorem 2 and the fact that Ξ^1 or Ξ^2 cascades are formed by grouping gates with the identical masking operators which extract either least or most significant bits.

Example 10 (continuation of Example 8). The benchmark function C17 has four single-rail Ξ^2 cascades, one of them is depicted in Fig. 4b. The corresponding linear arithmetic expression is $f_1 = \Xi^3\{4 + x_1 + a - 2c\}$.

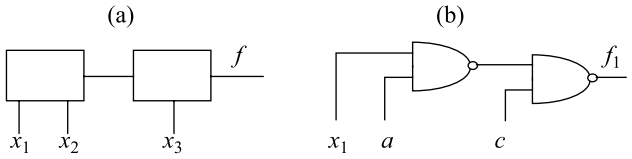

Fig. 4. A single-rail cascade and a Ξ^2 cascade from the benchmark circuit C17.

The ultimate goal of the cascading process is to obtain reasonable characteristics of a linear model, in particular, the minimal set of linear expressions, minimal values of weighted coefficients, the simple masking procedure, acceptable memory allocation and optimal time delays. To start a linearization algorithm through decomposing a circuit into a set of cascades, we integrate a cascade label *cascade* into the standard description of a gate $\mathcal{G}$. We propose the following cascading algorithm:

Step 1. Initialize cascade labels for all gates with zeros: $\mathcal{G}.cascade = 0$.
Initialize the cascade counter with one.

Step 2. Consider consequently all paths from each input to outputs. Traverse the circuit from the current input to corresponding outputs.

Step 3. Assign the current cascade counter to the cascade label of the current gate. Increment the cascade counter if the traversal reaches an output or $fanout > 1$.

Step 4. Go to Step 2 until all paths are covered.

Example 11 (continuation of Example 8). The results of the cascading algorithm for the benchmark circuit C17 are depicted in Fig. 5. The cascade based linear model is given in Fig. 6b.

Linear arithmetic expressions are obtained for each cascade by naive manipulations discussed in Section 2. For a netlist of cascades, the linear description of a cascade evaluated by these manipulations is substituted instead of the original cascade description.

Let us make preliminary evaluation of the level and cascade based strategies for the benchmark circuit C17 (Fig. 6a shows the result of the level based modeling, the cascade based model is given in Fig. 6b).

Observations. (1) The cascade based model results in four linear arithmetic expressions: $f_1 = \Xi^3\{4+x_1+a-2c\}$, $f_2 = \Xi^3\{4+x_5+b-2c\}$, $b = \Xi^2\{3-a-x_4\}$, and $c = \Xi^2\{3-b-x_2\}$. Whereas,

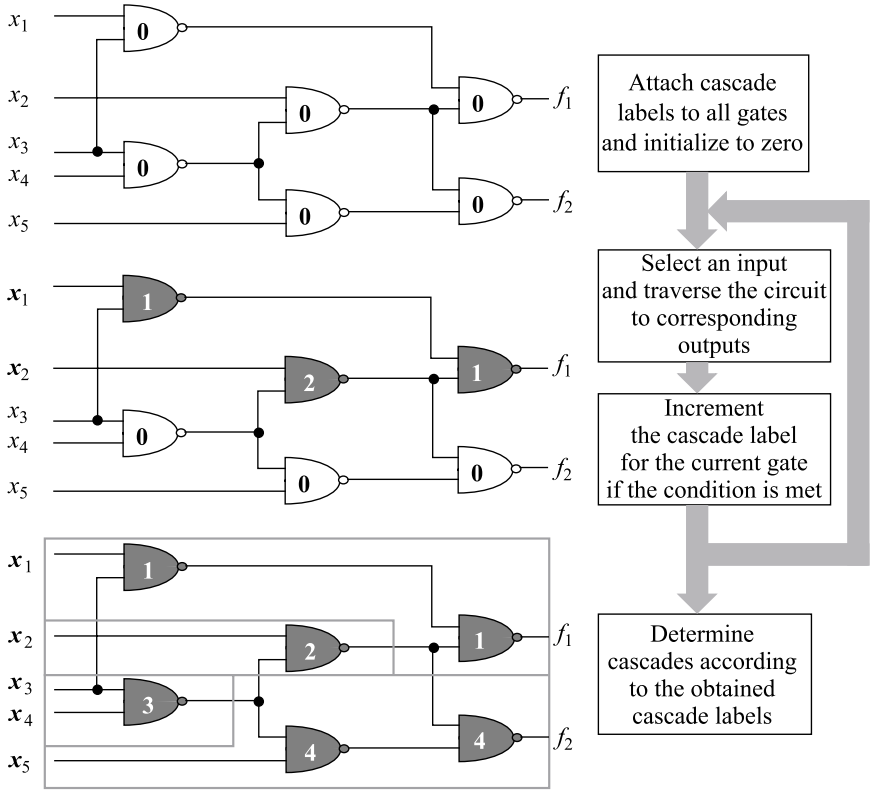

Fig. 5. Cascading the benchmark C17.

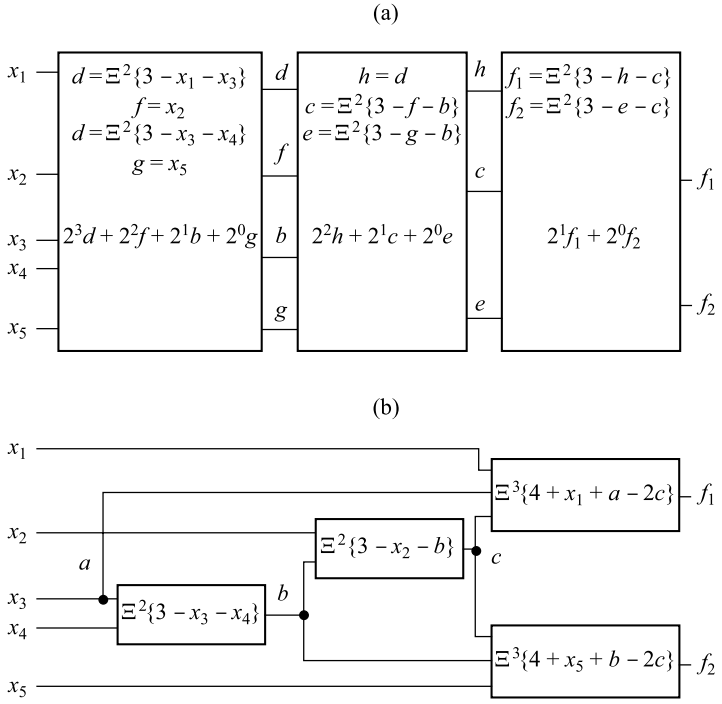

Fig. 6. Level and cascade based linearizations of the benchmark circuit C17.

the level based model is comprised of nine linear arithmetic expressions: $f_1 = \Xi^2\{3 - h - c\}$, $f_2 = \Xi^2\{3 - e - c\}$, $h = d$, $c = \Xi^2\{3 - f - b\}$, $e = \Xi^2\{3 - g - b\}$, $d = \Xi^2\{3 - x_1 - x_3\}$, $f = x_2$, $b = \Xi^2\{3 - x_3 - x_4\}$, and $g = x_5$.

(2) In cascade based model, 18 coefficients including the masking parameter should be stored, whereas the level based model requires 24 coefficients including the masking parameter, and gate based model needs 24 coefficients as well.

According to these preliminary evaluations, we envision that for large circuits reported effects will be more impressive (see Section 5 for the discussion of the experimental results).

4. TIMING ANALYSIS

We examine the linear word-level modeling through finding time delays in signal propagation. Topological timing analysis techniques are employed to make use of mixed-mode modeling and to supply approximate values of delays which usually suffice in the earlier stages of circuit design. Other methods with exact nature would be more appropriate to perform final tuning at the last stages of circuit design. Such methods are computationally expensive and not covered in this paper. Here, we use the fact that mixed-mode modeling reflects not only functional (high-level) behavior but also topological (low-level) structure. There are many other competing methods for an approximate timing analysis, for instance, a conditional delay method [6], and a methods based on algebraic decision diagrams [1]. This section covers two timing analysis approaches using linear word-level models.

Level based. The traditional problem of finding the longest topological path is considered. The output of the levelizing algorithm is used to determine the longest path under the fixed gate delay model [1].

Cascade based. Since the output of the cascading algorithm determines components that can be isolated as separate blocks and even simulated independently, the cascade-based technique can provide more accurate results. Thus, an approximation to the problem of a critical path can be given, where the critical path is the longest sensitizable path under the fixed cascade delay model.

Example 12 (continuation of Example 8). Let us compare the results of timing analysis for the benchmark circuit C17 by the level and cascade based modeling.

Level based. The longest topological path is equal to the number of levels, three for the benchmark circuit C17, under the fixed gate delay model. Since the circuit is composed of two-input NAND gates, the fixed gate delay model is considered to be accurate.

Cascade based. The longest sensitizable path in the circuit is determined by (1) building cascades, and (2) calculating the delay under the fixed cascade delay model. This delay for the benchmark circuit C17 is equal to three.

Our experimental results on timing analysis are reported below.

5. EXPERIMENTS

All algorithms were coded in C++ and integrated into a single program *neo*fractropy. The experimental results were obtained on a Pentium III 1.0 GHz workstation with 256 Mb of memory. We performed all experiments on the ISCAS85[3] benchmark set. This section shows the outcome of the levelization, cascading, and linearization algorithms through three sets of experiments, namely:

(i) comparing different representations based on gate, level or cascade processing for building word-level models;

[3] http://www.cbl.ncsu.edu/CBL_Docs/iscas85.html

(ii) building linear models and evaluating memory requirements for these models;

(iii) analyzing topological delays under these models.

5.1. Comparison of Different Netlist Representations

Table 1 outlines the characteristics of all combinational circuits from the ISCAS85 benchmark set. The column labelled Dataset gives the names of those circuits, and the column Function shows their functional characteristics. The number of inputs, outputs and gates of those circuits are given by the columns #Inputs, #Outputs and #Gates respectively. The rest of the table contains our experimental results. Thus, the columns labelled #Levels and #Cascades give the results of the levelizing and cascading algorithms obtained by our *neo*fractropy program.

In our experiments, we distinguish two types of cascades: ones composed of $\mathcal{AND}$, $\mathcal{OR}$, $\mathcal{NAND}$ and $\mathcal{NOR}$ gates; and cascades formed of $\mathcal{EXOR}$ and $\mathcal{NEXOR}$ gates and their variations. This is done because of necessity to find linear word-level models for separate cascades (see Section 3). Let us consider two benchmark circuits C17 and C432 in more detail.

The benchmark circuit C17 has five inputs, two outputs, and six NAND gates (Example 8). The levelizing algorithm determines three levels: two gates per level (Fig. 2a). The cascading algorithm finds four cascades with the density of one or two gates per cascade (Fig. 2b).

The benchmark circuit C432 has 36 inputs, 54 outputs, and 160 gates including $\mathcal{NOT}$, $\mathcal{AND}$, $\mathcal{NAND}$, $\mathcal{NOR}$ and $\mathcal{EXOR}$. The levelizing algorithm finds 17 levels with nine gates per level on average. The cascading algorithm determines 122 cascades with the number of gates per cascade ranging from one to four.

Observations. (1) It takes less than a second to levelize and cascade the entire set of benchmarks, because the computational complexity of both algorithms is in $O(\#\mathsf{Gates})$.

(2) These results reveal that in $(13274 - 8422)/8422 = 58\%$ cases cascades are formed of two and more gates (the values of 13 274 and 8422 are taken for the total number of gates and cascades respectively). This number can be further justified by the fact that the vast majority of gates in the selected benchmark set has the fanout greater than one. Such a condition terminates the process of forming cascades (see Section 3). Thus, for example, the benchmark circuit C17 has six gates, two gates have the $fanout = 2$, four others have the $fanout = 1$. Hence, only latter gates would be possible candidates for forming a cascade.

(3) The worst ratio gates per cascade of 1.248 is obtained for the multiplication circuit C6288: $ratio = 2416/1936 = 1.248$. The result for the ALU circuit C880 has the best ratio gates per

Table 1. Experimental results on circuit levelizing and cascading

Dataset	Function	#Inputs	#Outputs	#Gates	#Levels	#Cascades
C17	—	5	2	6	3	4
C432	Interrupt control	36	54	160	17	122
C499	Error detection	41	32	202	11	144
C880	ALU	60	32	383	24	215
C1355	Error detection	41	32	546	24	392
C1908	Error detection	33	25	880	40	506
C2670	ALU/control	233	108	1193	32	713
C3540	ALU	50	24	1669	47	961
C5315	ALU	178	163	2307	49	1397
C6288	Multiplication	32	32	2416	124	1936
C7552	Adder/comparator	207	109	3512	43	2032
Total				13 274	414	8422

cascade: $ratio = 383/215 = 1.78$. Thus, some improvements are expected in timing analysis for the benchmark circuit C880, whereas no improvements are envisioned for the benchmark circuit C6288 (see the columns #Gates and #Cascades from Table 1).

5.2. Evaluation of Memory Requirements

Table 2 gives the results of experiments on memory requirements for all linear models. The column ISCAS shows the memory allocation for the circuits in ISCAS85 format. Another widely used format EDIF requires much more memory, and therefore not covered by this study. The results reported in the next column LDD are taken from [2]. The rest of the table, labelled *neo*fractropy, contains the results of our experiments using gate based, level based, and cascade based strategies for circuit processing. To speed up the program, we created a database of linear representations for all basic gates ($\mathcal{AND}$, $\mathcal{OR}$, $\mathcal{NAND}$, $\mathcal{NOR}$, $\mathcal{EXOR}$, $\mathcal{NOT}$ and $\mathcal{BUFF}$) with up to eight inputs. For example, only two coefficients of an arithmetic expression are required to represent two-input $\mathcal{AND}$ gate (see Section 2 for detail study).

Table 2. Experimental results on memory requirements (in bytes) for linearized benchmark circuits

Dataset	ISCAS (original)	LDDs [2]	*neo*fractropy		
			gates	levels	cascades
C17	1303	201	174	150	135
C432	18 991	4601	5336	4066	3741
C499	21 989	5427	4410	2473	2897
C880	37 844	8306	12 001	10 496	7819
C1355	59 573	16 850	18 209	12 028	11 980
C1908	78 574	22 491	20 530	15 068	12 056
C2670	110 025	36 305	27 961	19 982	17 575
C3540	145 359	54 857	39 815	30 421	29 868
C5315	220 596	107 392	57 489	42 513	10 145
C6288	255 406	68 572	87 071	59 066	56 183
C7552	311 939	271 970	85 351	63 189	54 088
Total	1 261 599	596 972	358 347	259 452	206 487

Observations. (1) The linear word-level models built by our *neo*fractropy program using level based processing outperforms recent results reported in [2] and related to the linear decision diagram modeling.

(2) The cascade linear models require $(259\,452 - 206\,487)/259\,452 = 20\%$ less memory than their level counterparts. These results indicate that memory consumption can be reduced significantly, e.g. up to a factor of twenty for the benchmark circuit C5315, comparing to the original ISCAS85 format.

(3) The creation of gate, level and cascade linear models is computationally efficient. It takes less than an hour to build word-level models for the entire set of benchmark circuits. The computational complexity of the circuit linearization algorithm is in $O(\#\mathsf{Gates})$ for the gate based processing, in $O(\#\mathsf{Levels})$ for the level based processing, and in $O(\#\mathsf{Cascades})$ for the cascade based processing.

5.3. Timing Analysis

Table 3 compares the topological delays of level based and cascade based linear models with the known approximation methods: the conditional delay method [6][4], and the method based on

[4] Here, we compare the results with the best reported values for the conditional delay method.

Table 3. Experimental results on timing analysis

Dataset	delay [6]	delay [1]	*neo*fractropy delay	
			levels	cascades
C17	–	–	3	3
C432	17	23	17	15
C499	11	24	11	10
C880	24	15	24	16
C1355	24	–	24	23
C1908	37	–	40	28
C2670	30	26	32	28
C3540	46	–	47	35
C5315	47	34	49	32
C6288	124	–	124	124
C7552	–	38	43	30

algebraic decision diagrams [1]. Our program generates an expected output in terms of longest paths for the level based modeling (see Section 4).

Observations. (1) The cascade based modeling shows more accurate results for the longest sensitizable path under the fixed delay model. On the one hand, although these results are classified as approximate, they can be used as a lower bound for exact timing analysis methods. On the other hand, the results obtained through level based modeling determine upper bound values under the fixed delay model. Based on the results reported in Table 3, one can make a conclusion that the values taken from [6] are within intervals bounded by the outputs of cascade and level based modeling. For example, the delay for the benchmark circuit C5315 listed in [6] is 47, whereas the level based linear modeling results in 49, and the cascade based linear modeling estimates the delay in 32.

(2) Results of the cascade timing analysis largely depend on the ratio gates per cascade discussed previously. This fact is justified by the way how single-rail cascades are built (see Section 3). For the circuit where each cascade contains a single gate, the output of the level based modeling must be equal to the output of the cascade based modeling. Thus, the benchmark circuits C432, C499 and C6288 have low ratios gates per cascade, and the corresponding delays show no or very little improvements comparing to the level based modeling. For example, the multiplication circuit C6288 with the $ratio = 1.248$ has identical delay values of 124 for both cascade and level based modeling.

6. SUMMARY

The paper surveyed linear word-level structures and their properties for combinational circuit modeling. The problem of modeling was studied at both structural and behavioral levels. Different strategies of circuit processing was outlined taking advantage of linear word-level models. The practical problem of timing analysis was considered employing different linear word-level models.

Modeling approaches outlined above offer great potential for reducing design time and cost of simulation on general purpose hardware using software tools. However, the successful use of linear word-level structures depends on the effective utilization of hardware tools. In addition to software driven simulation described above, combinational circuits can be further modelled and simulated on inexpensive hardware: programmable logic devices. Currently, we focus on the problem of hardware support for modeling. We also foresee other practical applications of linear word-level models in circuit verification and testing.

REFERENCES

1. Bahar, R., Cho, H., Hachtel, G., Macii, E., and Somenzi, F., Timing Analysis of Combinational Circuits Using ADDs, *Proc. Eur. Design and Test Conf.*, *IEEE Computer Society Press*, 1994, pp. 625–629.

2. Dziurzanski, P., Malyugin, V., Shmerko, V., and Yanushkevich, S., Linear Models of Circuits Based on the Multivalued Components, *Avtom. Telemekh.*, 2002, no. 6., p. 99.

3. Malyugin, V., Realization of Boolean Function's Corteges by Means of Linear Arithmetical Polynomial, *Avtom. Telemekh.*, 1984, no. 2.

4. Rahardja, S. and Falkowski, B., Fast Linearly Independent Arithmetic Expansions, *IEEE Trans. Computers*, 1999, vol. 48, no. 9, pp. 991–999.

5. Sasao, T., Design Methods for Multi-rail Cascades, *Proc. Int. Workshop on Boolean Problems*, 2002.

6. Yalcin, H., Hayes, J., and Sakallah, K., An Approximate Timing Analysis Method for Datapath Circuits, *Proc. IEEE Int. Conf. on Computer Aided Design*, 1996, pp. 114–118.

This paper was recommended for publication by O.P. Kuznetsov, a member of the Editorial Board