Combinational Verification based on High-Level Functional Specifications*

Evguenii I. Goldberg† Yuji Kukimoto‡ Robert K. Brayton‡

Cadence Berkeley Laboratories
Berkeley, CA 94704†
Department of Electrical Engineering and Computer Sciences
University of California, Berkeley, CA 94720‡

{eugeneg,kukimoto,brayton}@ic.eecs.berkeley.edu

Abstract

We present a new combinational verification technique where the functional specification of a circuit under verification is utilized to simplify the verification task. The main idea is to assign to each primary input a general function, called a *coordinate function*, instead of a single variable function as in most BDD-based techniques. BDDs of intermediate nodes are then constructed based on these coordinate functions in a topological order from primary inputs to primary outputs. Coordinate functions depend on primary input variables and extra variables. Therefore combinational verification is performed not over the set of primary input variables but over the extended set of variables. Coordinate functions are chosen in such a way that in the process of computing intermediate functions the dependency on the primary input variables is gradually replaced with that on the extra variables, thereby making boolean functions associated with primary outputs simple functions only in terms of the extra variables. We show that such a smart choice of coordinate functions is possible with the help of the high-level functional specification of the circuit.

1 Introduction

Implementation verification is to verify whether a gate-level circuit implements its functional specification given in a more abstract level. In practice implementation verification of a gate-level circuit is often performed by checking its equivalence with another gate-level circuit whose correctness has been already established. One class of combinational verification methods is to use BDDs or their derivatives [2]. A drawback of such approaches is blow-ups of BDDs. Another class of methods is based on exploiting structural similarity between two circuits. Although such methods can verify examples for which BDDs are prohibitively large, they cannot solve the problem completely since they rely on a very restrictive assumption on structural similarity. Circuits are considered structurally similar if they contain a considerable number of functionally equivalent points. However, a simple transformation on a circuit can yield another circuit where no internal node is functionally equivalent to any node in the original circuit. It is more natural to consider two circuits structurally similar if they are produced from the same high-level functional description by different sequences of local transformations. The problem, however, is that after reducing the original implementation verification problem into equivalence checking of two gate-level networks the original high-level information is completely lost. In this paper we present a new approach to combinational verification where this high-level functional specification is utilized to simplify equivalence checking.

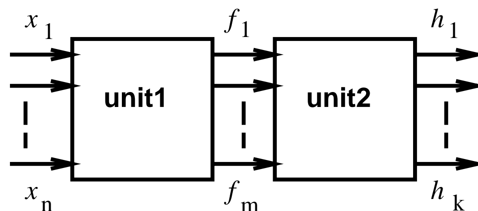

Figure 1: High-level functional specification

We explain our approach by the example of verifying a cascade circuit composed of two large combinational blocks in Figure 1. For the sake of simplicity suppose that

*This work was done when the first author was with Academy of Sciences of Belarus, Minsk and University of California, Berkeley.

$m \leq n$ holds and for any vector $\mathbf{y} = (y_1, \ldots, y_m) \in \{0,1\}^m$ there exists $\mathbf{x} = (x_1, \ldots, x_n) \in \{0,1\}^n$ such that $unit_1(\mathbf{x}) = \mathbf{y}$. Note that even if there exist small-size BDDs for $unit_1$ and $unit_2$ separately in terms of their input variables, a compact BDD may not exist for the entire circuit under any ordering of $\mathbf{x}$.

Our approach is still based on BDDs, but unlike most of the existing techniques the function associated with each primary input x_i is a general function X_i called a *coordinate function* instead of a single variable x_i. BDD construction is done starting from the coordinate functions at the primary inputs. The coordinate functions depend on primary input variables $x_1, \ldots, x_n$ and extra variables $y_1, \ldots, y_m$. Intuitively each extra variable y_i corresponds to output f_i as we will see later. As a result of this generalization the verification problem of functions $h_1(\mathbf{x}), \ldots, h_k(\mathbf{x})$ is translated into that of functions $h_1(\mathbf{X}), \ldots, h_k(\mathbf{X})$, where $\mathbf{X} = (X_1(\mathbf{x},\mathbf{y}), \ldots, X_n(\mathbf{x},\mathbf{y}))$. To make the verification of $h_i(\mathbf{x})$ equivalent to that of $h_i(\mathbf{X})$, $\mathbf{X}$ must be a surjective mapping from the boolean space $\{0,1\}^{n+m}$ to $\{0,1\}^n$, i.e.

$$\forall \mathbf{x}, \exists (\mathbf{x}', \mathbf{y}') \text{ s.t. } \mathbf{X}(\mathbf{x}', \mathbf{y}') = \mathbf{x}. \quad (1)$$

As it will be shown later, in case all signal patterns are observable at $f_1, \ldots, f_m$, we can always construct functions X_i so that

$$\forall i, f_i(X_1, \ldots, X_n) = y_i. \quad (2)$$

We are interested in finding functions X_i satisfying (1) and (2) that have small BDD representations. Although finding such functions is not easy in general, the knowledge on the high-level functional specification of $unit_1$ considerably simplifies this process as we will see later.

Let us illustrate how the idea of coordinate functions helps verify the equivalence between an implementation shown in Figure 2 and its high-level functional specification in Figure 1.

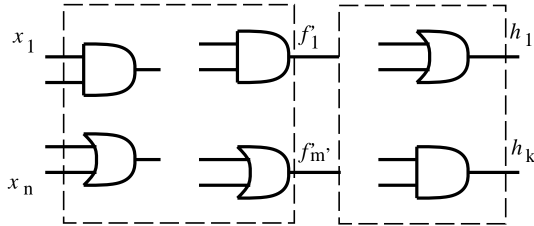

Figure 2: An implementation of the high-level functional specification

We make the following basic assumption about implementation circuits.

> In any implementation circuit N there exists a set of m' gates forming a cut C of N such that the function $f'_i(x_1, \ldots, x_n)$ realized by the i-th gate of C can be represented as a simple composition of functions $f_1, \ldots, f_m$ and $x_1, \ldots, x_n$, i.e.

$$f'_i(x_1, \ldots, x_n) = \textit{simple_function}_i(f_1, \ldots, f_m, x_1, \ldots, x_n) \quad (i = 1, \ldots, m') \quad (3)$$

As explained below one needs this assumption to guarantee that introducing extra variables simplifies the representation of functions at internal nodes.

Let N_0 be a gate-level circuit which is obtained directly by replacing the high-level functional descriptions of modules $unit_1$ and $unit_2$ with their gate-level implementations. Since an implementation N is typically generated by optimizing N_0, we cannot guarantee that the original m boundary points are completely preserved in terms of functionality in N. However, by our assumption we can roughly divide the set of gates in N into two subsets $UNIT_1$ and $UNIT_2$ corresponding to $unit_1$ and $unit_2$ respectively so that functions realized at the border of the two subcircuits are "close" to $f_1, \ldots, f_m$. From (2) and (3) it follows that

$$f'_i(x_1, \ldots, x_n) = \textit{simple_function}_i(y_1, \ldots, y_m, x_1, \ldots, x_n) \quad (i = 1, \ldots, m') \quad (4)$$

Expression (4) shows that introducing extra variables $y_1, \ldots, y_m$ is effective in reducing the complexity of function representations. Note that implementations N' and N'' satisfying (3) may not have any functionally equivalent internal nodes. So the class of implementation circuits satisfying the basic assumption is much broader than that of circuits verifiable by methods exploiting functional equivalence of internal nodes.

The verification procedure is organized as follows. First, functions $X_1, \ldots, X_n$ are represented as BDDs over variables $\mathbf{x}$ and $\mathbf{y}$. Then intermediate functions implemented by the gates of the circuit are computed in a topological order by function composition until primary outputs are reached. While the size of BDDs over variables $\mathbf{x}$ typically increases as we move from inputs to outputs, the size of BDDs over variables $\mathbf{x}$ and $\mathbf{y}$ behaves differently; it has the first peak somewhere in $UNIT_1$, but starts decreasing as we approach the boundary between $UNIT_1$ and $UNIT_2$. Once the boundary is passed, the size increases again to primary outputs. Intuitively the introduction of coordinate functions $X_1, \ldots, X_n$ replaces the dependency on $\mathbf{x}$ with the dependency on $\mathbf{y}$ in boundary functions.

This paper is organized as follows. In Section 2 we describe the relationship between the proposed method and other verification methods based on domain transformation. We then prove the correctness of the verification formally in Section 3. Section 4 discusses how to construct coordinate functions. In Section 5 we consider as an example the special case where $unit_1$ is an adder and show

how effective coordinate functions can be constructed from the high-level functional specification of $unit_1$. Section 6 gives experimental results and Section 7 concludes the paper.

2 Combinational Verification using Domain Transformations

The method presented in this paper can be classified as a domain transformation method, originally proposed by Meinel *et al.* [1]. The basic idea of domain transformations is to transform a given function to a "simpler" function and represent the transformed function using BDDs. Meinel showed that some functions whose BDD sizes are proven to be exponential under any variable ordering have polynomial-size BDDs after carefully constructed transformations.

Let $h(x_1, \ldots, x_n)$ be a completely specified function. [1] proposed to use a bijective transformation $\mathbf{f} = (f_1(\mathbf{x}), \ldots, f_n(\mathbf{x})) : \{0,1\}^n \mapsto \{0,1\}^n$. Let $\mathbf{z} = (z_1, \ldots, z_n)$ be the variables corresponding to the transformed domain. Given a transformation $\mathbf{f}$, $h(\mathbf{x})$ is transformed into $H(\mathbf{z}) = h(\mathbf{f}^{-1}(\mathbf{z}))$. By choosing an appropriate transformation, it is possible to switch over to a simpler function with a compact BDD.

An advantage of this type of transformations is that a transformation maps a function $h(\mathbf{x})$ to another completely specified function $H(\mathbf{z})$. Since H is uniquely determined by $\mathbf{f}$, any canonical representation of H serves as a canonical representation of h. However, so far only local domain transformations have been investigated [4] since the bijective restriction on transformations sets the number of transformation functions to the number of primary inputs. On the other hand, intuitively good "global" transformation functions can be found only when a structural representation of h is investigated. Therefore, the number of transformation functions should depend on the structure of h rather than the number of primary inputs.

In [3] we considered the case where the number of transformation functions $\mathbf{f} = (f_1, \ldots, f_m)$ is greater than n. Transformation functions give an injective mapping from $\{0,1\}^n \mapsto \{0,1\}^m$. The main idea of [3] is that a high-level functional specification of h typically has some auxiliary functions to simplify the description and that those functions are good candidates for transformation functions. In particular we investigated transformations of the form $\mathbf{f} = (x_1, \ldots, x_n, g_{n+1}, \ldots, g_m)$, where $g_{n+1}(\mathbf{x}), \ldots, g_m(\mathbf{x})$ are auxiliary functions used in the functional specification of h.

Since $\mathbf{f}^{-1}$ is not fully specified in this case, a function $h(\mathbf{x})$ is transformed into an incompletely specified function $H(\mathbf{z})$. More specifically $H(\mathbf{z})$ is not specified if vector $\mathbf{z}$ is not satisfiable, i.e. if there exists no $\mathbf{x}$ such that $\mathbf{f}(\mathbf{x}) = \mathbf{z}$. Instead of choosing a completely specified function compatible with H we represent H itself canonically using a three-terminal BDD where the third terminal node represents a don't care. We further showed that the class of functions having polynomial size BDD representations of H subsumes the classes of functions having polynomial size representations in other canonical BDD extensions.

On a practical side, however, experiments on benchmark circuits showed that if one uses intermediate functions of existing nodes in a network as transformation functions, the size of the three-valued BDD for H is often larger than that of the BDD for h over the original set of input variables.

In this paper we look for coordinate functions $\mathbf{X}(\mathbf{z})$ specifying an "inverse" mapping: $\{0,1\}^m \mapsto \{0,1\}^n$. By choosing an inverse mapping directly we select a unique completely specified function $H'(\mathbf{z}) = h(\mathbf{X}(\mathbf{z}))$. Therefore any canonical representation of H' is a canonical representation of h.

3 Correctness of the Verification Method

Theorem 1 *If condition (1) holds, verification of functions $h_i(\mathbf{x})$ is equivalent to that of functions $h_i(\mathbf{X})$.*

Proof. Let N_1 and N_2 be two circuits to be compared implementing functions $h_1(\mathbf{x}), \ldots, h_k(\mathbf{x})$ and $h'_1(\mathbf{x}), \ldots, h'_k(\mathbf{x})$ respectively. If there exist $\mathbf{x}'$ and i such that $h_i(\mathbf{x}') \neq h'_i(\mathbf{x}')$, then $\exists(\mathbf{x}^*, \mathbf{y}^*), h(\mathbf{X}(\mathbf{x}^*, \mathbf{y}^*)) \neq h'(\mathbf{X}(\mathbf{x}^*, \mathbf{y}^*))$, where $\mathbf{x}' = \mathbf{X}(\mathbf{x}^*, \mathbf{y}^*)$. If, on the other hand, there exist $(\mathbf{x}^*, \mathbf{y}^*)$ and i such that $h_i(\mathbf{X}(\mathbf{x}^*, \mathbf{y}^*)) \neq h'_i(\mathbf{X}(\mathbf{x}^*, \mathbf{y}^*))$ then $h_i(\mathbf{x}') \neq h'_i(\mathbf{x}')$, where $\mathbf{x}' = \mathbf{X}(\mathbf{x}^*, \mathbf{y}^*)$. Therefore the equivalence of the two verification problems holds. □

4 Finding Coordinate Functions

Let

$$X_i(\mathbf{x}, \mathbf{y}) = \begin{cases} x_i & \text{if } unit_1(\mathbf{x}) = \mathbf{y} \\ p_i(\mathbf{x}, \mathbf{y}) & \text{otherwise} \end{cases} \tag{5}$$

where $p_i(\mathbf{x}, \mathbf{y})$ is an arbitrary function. (Vector $(\mathbf{x}, \mathbf{y})$ is said to be satisfiable if $unit_1(\mathbf{x}) = \mathbf{y}$.) It follows that coordinate functions defined by (5) satisfy (1) since $\mathbf{X}(\mathbf{x}, unit_1(\mathbf{x})) = \mathbf{x}$. We still need to guarantee that $\mathbf{X}(\mathbf{x}, \mathbf{y})$ have polynomial size BDDs.

First we consider the case when all outputs of $unit_1$ are observable. Define X_i in the following way.

$$\mathbf{X}(\mathbf{x},\mathbf{y}) = \begin{cases} \mathbf{x} & \text{if } unit_1(\mathbf{x}) = \mathbf{y} \\ \mathbf{x}' & \text{otherwise} \end{cases} \quad (6)$$

where $\mathbf{x}'$ is a vector such that $unit_1(\mathbf{x}') = \mathbf{y}$.

Since $unit_1$ is surjective, we can always find the $\mathbf{x}'$ defined above. Under this particular choice of $\mathbf{X}$, $unit_1(\mathbf{X}(\mathbf{x},\mathbf{y})) = \mathbf{y}$ holds. It is easy to see this by simple case analysis. If $unit_1(\mathbf{x}) = \mathbf{y}$, $\mathbf{X}(\mathbf{x},\mathbf{y}) = \mathbf{x}$ from the definition of $\mathbf{X}$, and it follows that $unit_1(\mathbf{X}(\mathbf{x},\mathbf{y})) = unit_1(\mathbf{x}) = \mathbf{y}$. If $unit_1(\mathbf{x}) \neq \mathbf{y}$, $\mathbf{X}(\mathbf{x},\mathbf{y}) = \mathbf{x}'$ and by the definition of $\mathbf{X}$ $unit_1(\mathbf{x}') = \mathbf{y}$. So again $unit_1(\mathbf{X}(\mathbf{x},\mathbf{y})) = unit_1(\mathbf{x}') = \mathbf{y}$.

An important property of coordinate functions X_i specified by (6) is that $h_i(\mathbf{X})$ does not depend on $\mathbf{x}$ since at the boundary of $unit_1$ and $unit_2$ the dependency on $\mathbf{x}$ is completely removed. Thus functions X_i can be seen as a way of making h_i depend on variables $\mathbf{y}$ instead of $\mathbf{x}$.

Note that it is very difficult to find fully automatically functions X_i satisfying (6) without any use of the high-level specification for the following two reasons. First, for each pair $(\mathbf{x},\mathbf{y})$ such that $unit_1(\mathbf{x}) \neq \mathbf{y}$, one needs to find $\mathbf{x}'$ such that $unit_1(\mathbf{x}') = \mathbf{y}$. This problem is equivalent to the satisfiability problem. Second, we are only interested in functions X_i whose BDDs are compact. To overcome the first obstacle one needs an extremely fast satisfiability problem solver. To surmount the second obstacle the solver must provide "close" solutions for "close" instances of the satisfiability problem.

However, as demonstrated in Section 5, making use of the high-level specification of $unit_1$ can drastically simplify this step.

Now assume that there are unobservable outputs of $unit_1$, i.e. $\exists \mathbf{y}$ such that $\forall \mathbf{x}.unit_1(\mathbf{x}) \neq \mathbf{y}$. Then definition in (6) does not apply since the existence of $\mathbf{x}'$ is not guaranteed. However, the basic idea of replacing the dependency on $\mathbf{x}$ with that on $\mathbf{y}$ still works. We cannot find coordinate functions that $unit_1(\mathbf{X}(\mathbf{x},\mathbf{y})) = \mathbf{y}$. Instead, we will look for $\mathbf{X}(\mathbf{x},\mathbf{y})$ such that $unit_1(\mathbf{X}(\mathbf{x},\mathbf{y}))$ are simple functions of $\mathbf{y}$. Once such coordinate functions are found, we still have simple functions at the boundary between $UNIT_1$ and $UNIT_2$ as long as an implementation circuit satisfies the basic assumption (3). A possible solution is to select $\mathbf{X}(\mathbf{x},\mathbf{y})$ so that it is equal to $\mathbf{x}''$, where $\mathbf{y}'' = unit_1(\mathbf{x}'')$ is as "close" as possible to $\mathbf{y}$. For example one can take as $\mathbf{y}''$ the closest in Hamming distance to $\mathbf{y}$. This minimizes the number of cut functions $f_i(\mathbf{X})$ for which $f_i(\mathbf{X}(\mathbf{x},\mathbf{y})) \neq y_i$. Also, since vector $\mathbf{x}''$ is selected independently of the value of $\mathbf{x}$ in $(\mathbf{x},\mathbf{y})$, functions $unit_1(\mathbf{x},\mathbf{y})$ are only dependent on $\mathbf{y}$.

5 Example

Consider the case where $unit_1$ in Figure 1 is an n-bit adder; $sum(\mathbf{a},\mathbf{b}) = (s_1(\mathbf{a},\mathbf{b}),\ldots,s_{n+1}(\mathbf{a},\mathbf{b}))$, where $\mathbf{a} = (a_1,\ldots,a_n)$ and $\mathbf{b} = (b_1,\ldots,b_n)$ are input operands with a_n, b_n, s_{n+1} being the most significant bits. Let 1^n denote a vector consisting of n 1's. Note that 1^{n+1} is the only unobservable vector at the outputs of the n-bit adder.

Let coordinate functions $\mathbf{A} = (A_1,\ldots,A_n)$ and $\mathbf{B} = (B_1,\ldots,B_n)$, where A_i and B_i depend on $a_1,\ldots,a_n,b_1,\ldots,b_n,y_1,\ldots,y_{n+1}$, be specified in the following way

$$(\mathbf{A},\mathbf{B})(\mathbf{a},\mathbf{b},\mathbf{y}) = \begin{cases} (\mathbf{a},\mathbf{b}) & \text{if } sum(\mathbf{a},\mathbf{b}) = \mathbf{y} \\ (\mathbf{a}',\mathbf{b}') & \text{else if } \mathbf{y} \neq 1^{n+1} \\ & \quad \text{where } sum(\mathbf{a}',\mathbf{b}') = \mathbf{y} \\ (1^n,1^n) & \text{otherwise} \end{cases} \quad (7)$$

The last part of the definition is the only difference from the definition of $\mathbf{X}$ in (6). Since $\mathbf{y} = 1^{n+1}$ never appears at the outputs of the adder, $\mathbf{x}'$ in (6) does not exist. We then find a satisfiable vector $\mathbf{y}''$ whose Hamming distance from $\mathbf{y}$ is minimum. One such vector is 01^n, where the first bit is the least significant bit. The input producing this output is $\mathbf{a} = \mathbf{b} = 1^n$. Therefore, the return values of $(\mathbf{A},\mathbf{B})$ for this case is $(1^n,1^n)$.

It is easy to check that $s_i(\mathbf{A},\mathbf{B}) = y_i$ if $i = 2,\ldots,n+1$ and $s_1(\mathbf{A},\mathbf{B}) = y_1 \cdot (\overline{y}_2 + \ldots + \overline{y}_{n+1})$. Notice that except when $\mathbf{y} = 1^{n+1}$, $sum(\mathbf{A},\mathbf{B}) = \mathbf{y}$. The dependency on $\mathbf{a}$ and $\mathbf{b}$ is completely replaced with that on $\mathbf{y}$.

Consider how to choose vector $(\mathbf{a}',\mathbf{b}')$ when $sum(\mathbf{a},\mathbf{b}) \neq \mathbf{y}$. and $\mathbf{y} \neq 1^{n+1}$. Let $int(\mathbf{a})$ denote the integer specified by $\mathbf{a}$. One simple way is to define $\mathbf{a}'$ and $\mathbf{b}'$ so that $int(\mathbf{a}') = int(\mathbf{b}') = int(\mathbf{y})/2$ if $int(\mathbf{y})$ is even and $int(\mathbf{a}') = \lfloor int(\mathbf{y})/2 \rfloor$ and $int(\mathbf{b}') = \lfloor int(\mathbf{y})/2 \rfloor + 1$ if $\mathbf{y}$ is odd.

> Note that when finding $\mathbf{x}' = (\mathbf{a}',\mathbf{b}')$ we exploit high-level information that $unit_1$ is an adder. Based on this we have a systematic way of obtaining $\mathbf{x}'$ for a given unsatisfiable vector without solving the satisfiability problem.

Under this particular choice of $\mathbf{a}'$ and $\mathbf{b}'$ $a'_i = y_{i+1}$. $b'_i = y_{i+1}$ if $int(\mathbf{y})$ is even and $b'_i = [incr(shift(\mathbf{y}))]_i$ otherwise, where $shift$ is an n-bit-input $(n-1)$-bit-output function shifting $\mathbf{y}$ one position to the left (i.e. dividing $int(\mathbf{y})$ by 2), and $incr$ is an $(n-1)$-bit-input n-bit-output function adding 1 to $int(shift(\mathbf{y}))$.

The BDD of the satisfiability function $unit_1(\mathbf{x}) \equiv \mathbf{y}$ for the n-bit adder has a linear size in n under the following variable ordering $y_1 < a_1 < b_1 < y_2 < a_2 < b_2 < \ldots < y_n < a_n < b_n < y_{n+1}$. We confirmed experimentally that

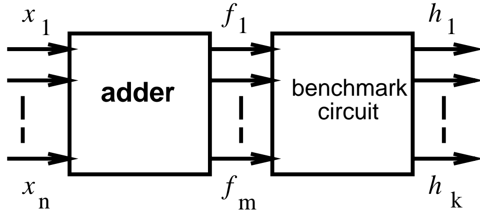

Figure 3: Cascade circuit

the BDDs of A_i and B_i also grow linearly in n under this variable ordering.

These coordinate functions $\mathbf{A}, \mathbf{B}$ can be used in any situation where an adder feeds another unit.

6 Experimental Results

Unfortunately combinational benchmark circuits currently available are specified at the gate level directly and do not come with any high-level functional specification. To create circuits with a known high-level structure from benchmark networks we constructed artificial networks shown in Figure 3.

Each circuit consists of two blocks: the first block is an n-bit adder whose outputs are connected to the inputs of the second block, which is a benchmark circuit. The value n was chosen so that the number of outputs of the adder $n+1$ is equal to the number of inputs of the benchmark circuit. Each composite circuit was optimized by `script.rugged` in SIS and was verified using two methods: 1) the method described in this paper and 2) a standard BDD-based approach where output BDDs are computed in terms of input variables. To demonstrate that the proposed verification method does not depend on the way the adder is implemented we used three different implementations of the adder: a ripple-carry adder, a carry-skip adder and a carry-select adder. The results of the experiments are summarized in Table 1. Each column of the table contains the following information.

- The name of a benchmark circuit used as $unit_2$.

- The number of inputs and outputs of the composite circuit and the number of gates in the optimized circuit.

- The results of the BDD-based verification: the total number of nodes in the BDDs for outputs under variable ordering $a_1 < b_1 < \ldots < a_n < b_n$ and CPU time in seconds on DEC AlphaServer 8400 5/300. The BDD-based verification was done only to composite circuits with ripple-carry adders.

- The results of the verification based on coordinate functions described in Section 5: the total number of nodes in BDDs representing the outputs in terms of the extended set of variables $(\mathbf{y}, \mathbf{a}, \mathbf{b})$ under variable ordering $y_1 < a_1 < b_1 < \ldots < y_n < a_n < b_n < y_{n+1}$ and CPU time for three different versions of adders.

The BDDs in terms of the extended set of variables is much smaller than the BDDs in terms of primary input variables. CPU time reduction is also considerable.

7 Concluding Remarks

We presented a new combinational verification method using high-level functional specifications. The effectiveness of this approach was demonstrated by taking cascade circuits as an example. This technique can be extended to networks with more complex topologies.

Although our approach requires designer's intervention, this human interaction makes it possible to verify a broader class of circuits than by existing techniques. Roughly speaking there are two approaches to solving an NP-complete or harder problem. The first is to focus on a specific class of instances and develop algorithms by taking advantages of the properties of the class. Verification methods exploiting structural similarity fall into this category. The other approach is based on algorithms with user interaction. Such algorithms use high-level information that cannot be recovered by brute-force computations. The proposed method is an example of such an approach since the use of coordinate functions can be considered as a way of "pumping" high-level information into algorithms.

References

[1] J. Bern, C. Meinel, and A. Slobodová. Efficient OBDD-based Boolean manipulation in CAD beyond current limits. In *Proceedings of 32nd Design Automation Conference*, pages 408–413, June 1995.

[2] R. E. Bryant. Graph-based algorithms for Boolean function manipulation. *IEEE Transactions on Computers*, C-35(8):677–691, August 1986.

[3] E. Goldberg, Y. Kukimoto, and R. K. Brayton. Canonical TBDD's and their applications to combinational verification. In *ACM/IEEE International Workshop on Logic Synthesis*, May 1997.

[4] C. Meinel and T. Theobald. Local encoding transformations for optimizing OBDD-representations of finite state machines. In *Proceedings of First International Conference on Formal Methods in Computer-*

circuit: adder +	inputs	outputs	gates	BDD-based approach		Our method			
							time		
				nodes	time	nodes	ripple	skip	select
C1355	80	32	273	227464	16.30	45922	2.94	2.98	3.01
C3540	98	22	382	4138802	557.38	604559	62.86	61.57	59.15
C880	118	26	239	2242888	164.20	346660	22.53	22.94	24.32
k2	88	45	441	191823	17.59	28336	3.16	3.23	3.59
C432	70	7	157	11631	1.01	1852	0.81	0.84	0.86
des	510	245	1343	2285948	249.29	73919	90.00	93.31	95.32
C1908	64	25	251	181862	26.68	36129	4.08	4.09	4.65
pair	344	137	790	3652319	86.28	67685	37.30	38.83	53.10
too_large	74	3	165	40545	2.63	7096	1.18	1.11	2.73

Table 1: Experimental results

Aided Design (FMCAD96), pages 404–418, November 1996.