6

Circuit depth optimization by BDD based function decomposition

Alexander I. Kornilov, Tatiana Yu. Isaeva
Research Institute of VLSI CAD Systems,
8a, Mazhorov per., Moscow E-023, 105023 RUSSIA,
phone: 7 (095) 369 19 75 fax: 7 (095) 369 51 07
e-mail: korn@sapran.msk.su

Abstract

We present a BDD-based algorithm for logic functions' decomposition that is aimed at speed-up of circuits constructed as straightforward mapping of BDDs. Existing algorithms (including BDD-based) aimed at circuits' size optimization do not solve this problem. We propose a novel decomposition formula; BDDs' graphical properties are explicitly used. This allows to reduce circuits' performance sufficiently. A class of BDDs occurring in arithmetic units' synthesis is described for which the algorithm can be applied iteratively.

Keywords

High-performance circuits, functions' decomposition, Binary-Decision Diagrams.

1 INTRODUCTION

This paper is the result of the research carried out in the field of high-performance combinational circuits synthesis. The proposed algorithm is based on Reduced Ordered Binary Decision Diagrams (further called BDD's for short) proposed by R.E.Bryant (Bryant,1986), that provide a convenient representation for most of the logic functions in practical use. We do not concern the problem of variable ordering since there exist several algorithms that find a good ordering if possible. This function representation is widely used in the field of logic synthesis. BDD's proved to be an effective structure for the implementation of the traditional synthesis methods such as function decomposition and factorization, ISOP form construction. Again several synthesis algorithms were developed that construct a network as the straightforward mapping of its BDD structure (e.g., in static CMOS base, multiplexor base or FPGA base). For such circuits the assumed BDD depth directly defines their performance. However, a number of familiar modern algorithms minimize BDD size only, leaving behind the problem of depth optimization.

G. Saucier et al. (eds.), *Logic and Architecture Synthesis*

Familiar methods of decomposition (including BDD-based decomposition) aimed at simplification of Boolean functions do not solve the problem at least for such important class of Boolean functions as carry functions in arithmetic units.

We made an attempt to take advantage of the fact that we use BDD structure both as the circuit model and as the function representation convenient for minimization. In this paper we describe a BDD based function decomposition algorithm aimed at the speed-up of the circuit constructed as the straightforward BDD implementation. The main goal of such decomposition is to assign a complicated function instead of a variable to the BDD root and hence reduce the BDD depth.

In the first section we remind the familiar Shannon's decomposition. The second section is devoted to the proposed algorithm description. In the third section we describe a class of BDD's connected with arithmetic units for which the algorithm is especially good due to the opportunity to apply it iteratively.

2 SHANNON'S DECOMPOSITION

Consider the Shannon's decomposition formula with respect to the first k variables.

$$(1)\quad f(x_1,\ldots,x_n) = \bigvee_{\forall (a_1,\ldots,a_k),\ a_i\in\{0,1\}} x_1^{a1} \,\&\, x_2^{a2} \,\&\ldots\&\, x_k^{ak} \,\&\, f(a_1,a_2,\ldots,a_k,x_{k+1},\ldots,x_n).$$

We can rewrite this equivalence in the following way:

$$(2)\quad f(x_1,\ldots,x_n) = \bigvee_{i\in I} g_i(x_1,\ldots,x_k) \,\&\, h_i(x_{k+1},\ldots,x_n).$$

Here $I=\{1,\ldots,w\}$, where w is the number of different cofactors, $H = \{h_i,\ i\in\}$ is the set of these cofactors and $G = \{g_i,\ i\in I\}$ is the set of corresponding characteristic functions:

$$g_i(a_1,\ldots,a_k).= \begin{cases} 1, & \text{if } h_i(x_{k+1},\ldots,x_n) \equiv f(a_1,a_2,\ldots,a_k,x_{k+1},\ldots,x_n) \\ 0, & \text{otherwise.} \end{cases}$$

Consider BDD implementing the function f for a certain variable ordering. For definiteness assume that the variables are ordered in the natural way: $x_1,x_2,\ldots,x_n$. Assume each internal BDD vertex v has an index of variable it is labeled, terminal vertices having index n+1. Let us denote this value index(v).

From formula (2) we immediately obtain a disjunctive decomposition of function f. According to this formula we can draw a crossing line that divides function f BDD into higher and lower parts, the former consisting of vertices v such that index(v)≤k, and the latter consisting of vertices v whose index(v)>k. Note that the lower part of the BDD implement the system of cofactors for a given k.

There exist several BDD patterns that at once lead to good function decomposition. E.g., if there are only two cofactors and one of them is constant, the function is implemented as disjunction (for constant 1) or conjunctions (for constant 0) of two functions. If the two cofactors are complementary functions the function is implemented as EXOR of two functions.

In general, if the number of cofactors is comparatively small we obtain a good function implementation. This case (disjunctive decomposition) was described in (Lai,1993), as well as a nondisjunctive decomposition algorithm for size optimization.

Function f may just be implemented as the independent implementation of system of characteristic functions G, system of cofactors H and formula (2). The total depth of this imple-

mentation is the sum of depths of formula (2) implementation ($\approx \log_2 w$, if it is implemented as a pyramidal structure) and maximal depth for the systems G and H (for these systems can be implemented in parallel). System H is implemented as the lower part of the initial BDD. Implementation for each of the functions g_i can be obtained from the upper part of the initial BDD. Thus we finally obtain implementation of less depth, not exceeding $\log_2 w + \max(k, n-k)$.

Unfortunately if the number of different cofactors is big this implementation is impractical both for size and depth minimization.

3 DECOMPOSITION ALGORITHM FOR DEPTH REDUCTION

In this section we describe the advanced decomposition algorithm for depth reduction.

Let us transform the formula (2) in the following way:

$$(3) \quad f(x_1,\ldots,x_n) = \bigvee_{i \in I_0,} g_i(x_1,\ldots,x_k) \,\&\, h_i(x_{k+1},\ldots,x_n) \vee g_0(x_1,\ldots,x_k) \,\&\, h_0(x_1,\ldots,x_n),$$

where I_0 is a subset of I, the function

$$g_0(x_1,\ldots,x_k) = \bigvee_{i \in I \setminus I_0,} g_i(x_1,\ldots,x_k)$$

and the function

$$h_0(x_1,\ldots,x_n) = \begin{cases} \bigvee_{i \in I \setminus I_0,} g_i(x_1,\ldots,x_k) \,\&\, h_i(x_{k+1},\ldots,x_n), & \text{if } g_0(x_1,\ldots,x_k) = 1; \\ \text{don't care,} & \text{otherwise.} \end{cases}$$

Note that unlike the previous case this formula contains only a subset of cofactors and characteristic functions; and the rest of them are gathered in the last term, that includes the special function h_0, depending on the whole set of input variables.

We choose the set of indexes I_0 so that to break only long paths in the BDD. Further we describe how we do this.

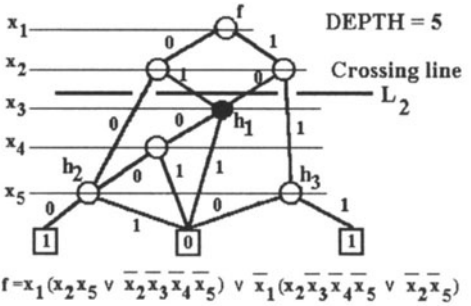

Figure 1. BDD example: k = 2; l_0 = 3; I = {1,2,3}; I_0 = {1}.

Definition. The edge cutset E_k of the BDD F at level k is the set of edges (v1,v2) for which index(v1)<=k, and index(v2)>k.

That is, if we draw a line dividing the BDD into upper and lower parts, the former consisting of the vertices with indexes less or equal than k, and the latter -- of the vertices with the indexes greater than k, Ek is the set of all the BDD edges, crossing that line.

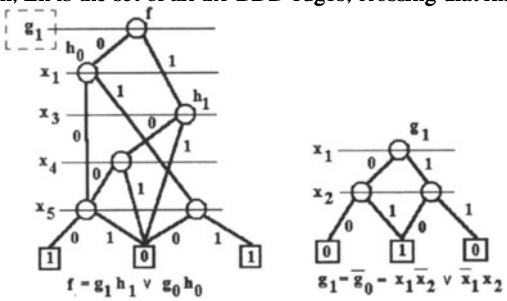

Figure. 2. Resulting implementation for the example function.

Given a set E of BDD edges denote Ends(E) the set of all the end vertices of the edges from E. Clear that the set Ends(E_k) is the set of all the BDD vertices corresponding to the functions from H. Let us set the critical length l_0

Definition. $E_k(l_0)$ is the set of edges from E_k, belonging to the paths that are longer than l_0.

Clear that Ends($E_k(l_0)$) $\subseteq$ Ends(E_k), and in many cases these sets are not the same. In Figure.1 an example of such a BDD is given. We choose I_0 as the set of indexes of the cofactors corresponding to the BDD vertices from Ends($E_k(l_0)$).

As in the previous case the system of cofactors $H_0 = \{h_i, i \in I_0\}$ is implemented by the lower part of BDD, and BDD's for the system of characteristic functions $G_0 = \{g_i, i \in I_0 \cup \{0\}\}$ can be built using the upper part of the BDD. Let us describe construction of these BDD's in more details.

Each of the characteristic functions can be considered as the reachability function (as defined by N. Ishiura in (Ishiura,1992)) between the initial BDD root and the BDD vertex representing the corresponding cofactor. We can build BDD for a function g_i ($i \neq 0$) in the following way:

- consider a subgraph consisting of all the paths connecting BDD root to the vertex representing cofactor h_i (preserving all the labels);
- consider the initial BDD root as the new BDD root;
- replace the vertex corresponding to the cofactor h_i with the terminal vertex 1;
- add missing complementary edges directing them to the terminal vertex 0.

Function g_0 is implemented as the other characteristic function save that we replace by terminal vertex 1 all the vertices corresponding to the cofactors h_i, $i \in I \setminus I_0$.

We can see that the depths of the BDD's for the cofactors and the characteristic functions are determined by the path's' lengths in the lower and upper BDD parts, and thus we can just estimate them as the number of input variables of these functions.

Unlike these functions, the special function h_0 depends on the whole set of input variables. Of course, some of them can be insufficient, but we don't know that beforehand. But for this function we can explicitly build BDD with depth not exceeding the chosen critical depth.

Consider all the paths in the initial BDD connecting its root to the terminal vertices passing through the BDD vertices from the set Ends(E_k)\Ends($E_k(l_0)$). These vertices correspond to the cofactors that are not implemented separately. These paths form a directed acyclic graph that is not a BDD for some of the complementary edges are missing, including the edges from $E_k(l_0)$. For every vertex with only one pull-down edge let us insert an edge directed to the same vertex as its complementary edge. Thus we obtain a new non reduced BDD implementing the func-

tion h_0 (setting its don't care values). Its depth does not exceed the critical depth because all the edges from the critical paths cutset are removed, and the inserted edges do not bring in long paths.

The total implementation depth is defined by the depth of the formula (3) implementation ($\approx\log_2|I_0|$); the depth of BDD's for the systems G_0 and H_0; the depth of function h_0 BDD. Thus it does not exceed $\log_2|I_0|+\max(l_0,k,n-k)$. If $k\approx n/2$ the depth is $\approx\log_2|I_0|+\max(l_0,n/2)$. Taking advantage of function h_0 don't care values often leads to the reduction by 1 of its BDD depth (See Figure.2).

It is important that for fixed crossing line level k and critical depth l_0 the time complexity of the proposed algorithm is linear on the number of BDD vertices. But of course to minimize the particular implementation depth a certain search may be carried out, e.g., a minimal cutting set may be found.

4 ALGORITHM APPLICATION FOR ARITHMETIC UNITS

There is a BDD type that often occurs in arithmetic units synthesis -- BDD's, that contain a single long path, the subBDD's at the ends of the complement edges along this path being of constant depth (e.g., subBDD's representing input variables and their complements). In this particular case the BDD edge cutset consists of a single edge, and the formula (3) can be rewritten as follows:

$$(4)\quad f(x_1,...,x_n) = g_1(x_1,...,x_k)\ \&\ h_1(x_{k+1},...,x_n)\ V\ \hat{}g_1(x_1,...,x_k)\ \&\ h_0(x_1,...,x_n)$$

BDD's for the functions h_1, g_1 and h_0 are of the same type as for the function f. This allows the iterative use of the method, thus yielding pyramidal structure instead of the initial linear structure. The structure depth is reduced from O(n) to $O(\log_2 n)$ while its size just slightly increases.

Basing on these principles we have developed a methodology to synthesize high-performance arithmetic units. We have designed a look ahead carry generator (LACG) as one of the applications of this methodology.

Let us consider a well known LACG (Motorola) based on the traditional 2-level synthesis principles for the purpose of comparison. To implement this generator in a basis of static CMOS circuits 116 transistors are required. Two 4-input NAND elements are connected sequentially in this circuit and produce the main part of the whole delay that is rather high.

Our LACG is based on the BDD technology principles (Kornilov,1988). To implement this generator in a basis of static CMOS circuits only 80 transistors are required. In this case the delay producing circuit consists of two sequentially connected passive multiplexors and two invertors with performance approximately equivalent to a performance of 3-inputs NAND gate (it also has the depth equal 3). Hence we can claim that the proposed generator is at least two times faster than first one and requires for its implementation approximately 30% less transistors.

Table 1: Comparison of the LACG implementations for different synthesis styles.

Two-level Synthesis		BDD-based Decomposition	
Output	Fall Delay/Rise Delay	Output	Fall Delay/Rise Delay
GE	4.60/4.90	AF	3.28/2.72
PR	1.06/4.30	MF	1.20/2.56
CO	5.30/5.72	CO	4.14/3.86

The developed methodology enables to construct high-performance arithmetic units with-

out using LACG.

This particular algorithm was described in (Kornilov,1990).

5 CONCLUSIONS

The proposed algorithm allows to reduce sufficiently the BDD representation depth without big size increase, and thus optimize performance of the circuits obtained by straightforward mapping algorithms. In some cases the structure depth may be reduced from linear to logarithmic in the number of input variables.

REFERENCES

Bryant R. E. (1986) Graph-Based Algorithms, *IEEE Trans. Comput., Vol. C-35, no.8, Aug. 1986, 677-691.*

Lai Yung-Te , Pedram Massound (1993) BDD Based Decomposition of Logic Function with Application to FPGA Synthesis, *Proceedings of 30th ACM/IEEE Design Automation Conference, 1993, 642-647.*

Ishiura N. (1992) Synthesis of Multi-Level Logic Circuits from Binary Decision Diagrams. *Synthesis and Simulation Meeting and International Interchange, Proccedings of SASIMI-92, 1992, 74-83.*

Motorola H4C Series Design Reference Guide, 7-165.

Kornilov A. I. et al. (1988) A Carry Forming Device, *Inventor's Certificate of the USSR No. 1608648, G 06 F 7/50.*

Kornilov A. I. (1990) High Performance Arithmetic Device Construction Using a Universal Logical Base, *Communication Facilities Engineering. Ser. Microelectronic Hardware, Vol. 1-2 (12-13), 1990, 41-47 (in Russian).*