Heuristic Synthesis of Multi-Terminal BDDs Based on Local Width/Cost Minimization

Petr Mikušek, Václav Dvořák
Brno University of Technology, CZ
{imikusek, dvorak}@fit.vutbr.cz

Abstract

Multi-terminal Binary Decision Diagrams (MTBDDs) are useful representation of multiple output Boolean functions. However, construction of such a diagram is a difficult task, especially when in some sense optimum diagram is sought. The paper presents an improved algorithm of MTBDD synthesis aiming at minimum MTBDD width or cost. The presented algorithm is a core of the upgraded version of a synthesis tool that accepts incompletely specified integer-valued functions of Boolean variables specified by possibly compatible cubes. The suggested technique is suitable for hardware (LUT cascades) or firmware implementation (branching microprograms).

Keywords: Incompletely specified functions, multi-terminal BDDs, LUT cascades, iterative disjunctive decomposition, functional decomposition.

1. Introduction

Design of digital systems is based on various specifications of Boolean functions, most often in a form of Boolean expressions or in cube notation (Espresso formats), [1]. Another popular machine representation of single-output Boolean functions uses binary decision diagrams (BDDs), which can have many forms, [2], [3].

Conversion of a single Boolean function into a BDD and related optimization problems were studied intensively [2]. As the variable ordering influences the cost and shape of the diagram, we should find one ordering of variables among all possible, that produces a diagram optimal in a certain sense (e.g. minimum cost, width or average path length).

Generalization of BDDs to multiple-output Boolean functions are so called word-level BDDs, among them e.g. multi-terminal BDDs (MTBDDs) or BDDs for characteristic function BDD_for_CF [3], [4]. The latter diagrams use both input and output variables at decision nodes what makes them more complex; their width can be minimized in some cases by the known algorithm [5]. On the other hand, optimum MTBDD synthesis, basically optimum ordering of variables with respect to a certain goal, is covered very little in the literature [3]; and yet, tools for BDDs synthesis [6] and manipulation cannot be used for MTBDDs, nor can be a MTBDD obtained from BDDs of component Boolean functions. Given the ordering of variables, the diagram may be obtained by decomposing the original function repeatedly, i.e. removing a group of 1 or more variables at each step.

In many cases integer values can be taken as identifiers for binary output vectors of multiple-output Boolean functions. This paper is more-less theoretical, presenting a heuristic technique of the MTBDD construction for incompletely specified integer functions of Boolean variables. The main contribution of the paper is the upgraded algorithm of iterative decomposition accepting incompletely specified functions.

The paper is structured as follows. Section 2 deals with the basic definitions and notions. MTBDD construction using a simple disjunctive decomposition iteratively is described in Section 3. Our heuristic approach to variable ordering is discussed in Section 4. The results and future research directions are commented on in Conclusion.

2. Basic definitions and notions

To begin our discussion, we define the following terminology. An integer function of n Boolean variables is defined as a set $\mathcal{F}$ of (n+1)-tuples, called function cubes, in which the first n components correspond to the binary inputs and the *single* integer component, to the output. Set $\mathcal{F}$ is only a shorthand description of a full function table, even though in the small example illustrated below (Table 1a, b) the full map looks smaller. The function F in Table 1 will be used as a running example in the sequel.

The value of symbol "-" is considered uncertain, whereas 0 and 1 are certain. An element c of $\{0, -, 1\}^n$ is called an input cube. We will be using notions of cube calculus [8].

Table 1. Integer function specification a) by the full map b) by cubes

↓ x3x4 →

x1x2	00	01	10	11
00	0	0	2	dc
01	0	dc	2	1
10	3	3	dc	3
11	3	1	2	1

dc = don´t care

	x1	x2	x3	x4	F
1	1	0	0	-	3
2	1	-	0	0	3
3	1	0	-	1	3
4	0	-	1	0	2
5	-	1	1	0	2
6	-	1	-	1	1
7	-	1	1	1	1
8	0	-	0	0	0
9	0	0	0	-	0

A set of (n+1)-tuples does not necessarily define an integer function, because it is possible to assign conflicting output values. Similarly to Boolean fr functions [1], we introduce *integer-valued fr functions*; they must satisfy the consistency condition, which guarantees that there are no contradictions; shortly, if two input cubes are compatible, their corresponding outputs must be identical.

Now we will define the basic notions related to MTBDDs (BDDs) and functional decomposition.

Def. 1. The *cost* of the MTBDD is given by the total number of true decision nodes, with outgoing edges directed to *different* nodes.

Def. 2. The *width* of the MTBDD at a certain height is the number of edges crossing the section of the MTBDD between adjacent levels of decision nodes, where the edges incident to the same node are counted as one.

Functional decomposition [3] of function F(X) is a serial disjunctive separation of F into two functions G (residual) and H (detached) such that

$$F(X) = H(U,G(V)). \qquad (1)$$

We want functions G and H to have strictly fewer inputs than F.

The advantage of MTBDDs over BDDs_for_CF is that while the former diagrams can be cut into slices of arbitrary size, the latter diagrams must be reconstructed after decomposition [5], because input and output variables are interleaved.

3. MTBDD construction based on the disjunctive iterative decomposition

Decomposition can be applied iteratively to a sequence of residual functions with a decreasing number of variables. In this section we will present a method of iterative disjunctive decomposition based on notion of blankets [8], modified and simplified for our case of integer functions. We will select always a single input variable (|U|=1), from now on denoted as a *detached variable*, that will be removed from a residual function in such a way that the width or cost of the diagram will be minimized locally. More general techniques like non-disjunctive decomposition or multi-variable decomposition (|U|>1) can be explored in future as well.

Instead of the exact formulation of a decomposition algorithm, we prefer to illustrate it on our running example. At the beginning we will select input variables for iterative decomposition simply in a natural sequence, with no optimization in mind. A single variable will be removed from the function in one decomposition step. Starting with variable x1 in our running example, we first create two-block blankets β2, β3, β4 for each input variable x2, x3, x4:

$$\beta2 = \{1, 2, 3, 4, 8, 9;\ 2, 4, 5, 6, 7, 8\}$$
$$\beta3 = \{1, 2, 3, 6, 8, 9;\ 3, 4, 5, 6, 7\} \qquad (2)$$
$$\beta4 = \{1, 2, 4, 5, 8, 9;\ 1, 3, 6, 7, 9\}.$$

Blankets consist of subsets (blocks) of cubes denoted by line numbers from Table 1b. The first block in each blanket includes cubes which contain "0" or "–" in place of variable x1, cubes in the second block have value "1" or "–" in place of variable x1. The input blanket for the subset (X\x1) is then obtained as an intersection of two-block blankets (2):

$$\beta = \{1, 2, 8, 9;\ 1, 3, 9;\ 4;\ 3;\ 2, 8;\ 6;\ 4, 5;\ 6, 7\}. \qquad (3)$$

Each block in blanket β can be assigned an ordered pair of function values

$$[F(0, x2, x3, x4), F(1, x2, x3, x4)]. \qquad (4)$$

There are three types of these output pairs:

a) **type [u, v]:** (true pair or decision node)
two values in the pair are different, u ≠ v; e.g. blocks (1,2,8,9), (1,3,9) and (2,8) in blanket (3) generate pair [0,3];

b) **type [u, u]:** (degenerated pair or decision node)
two values in the pair are identical; e.g. blocks (6) and (6,7) generate pair [1,1] and block (4,5) pair [2,2];

c) **type [u,-] or [-,u]:**
one of the values (4) doesn't exist in the list of cubes (it's don't care); e.g. block 4 generates pair [2,-] and block 3 generates [-,3]. Don't care value will be replaced by a particular value later to match case a) or b).

Now we can create compatible classes of these pairs with the goal to select a set of maximal classes, with minimal cardinality, that covers all the pairs; blocks with compatible output pairs get the same new id. In our example eight compatible classes in blanket β can be merged to three {1, 2, 3, 8, 9; 4, 5; 6, 7}, denoted with new id 0, 1, and 2, see Table 2. The minimal cardinality of merged blocks ensures that the number of outputs $\log_2|G|$ of the residual function G1(x2, x3, x4), is as small as possible. Finding maximal compatibility classes of pairs (4) is easy and can follow the algorithm below:

```
Algorithm 3.1. Assigning id's to output pairs.
1. List and enumerate all distinct output pairs with different certain values (type [u, v]);
2. continue in listing and enumeration of output pairs with the same certain values (type [u, u]);
3. output pairs with one certain value and one uncertain value (types [u,-] or [-, u]):
  if there is a compatible pair [u, u] in the list , use its id;
  else if there is a compatible pair, type [u,w] or [w,u], in the list , use its id;
  else (hardware optimization) if possible, join two output pairs [v, -] and [-,w] into a single pair [v, w] and assign it a next new id;
  else insert a new output pair with certain values [u, u] into the list and assign it a next new id.
```

Uncertain values are replaced by certain values in such a way as to reduce the number of new id numbers to a minimum. This is in fact utilization of don't cares for minimization. In more complex cases with ternary output vectors or when removing more than one variable at a time, finding the minimal cover must be done by more general methods, e.g. by graph coloring [8].

The detached function H1 is obtained, even though not in the cube form but in the integer form, by reading the first table in Table 2 backward,

H1(x1, new id) = [F(0, x2, x3, x4), F(1, x2, x3, x4)].

To obtain the residual function G1 is a bit trickier. The input blanket for the subset (X\x1) completed by function values "new id" contains redundant cubes which must be removed. On the other hand, all relevant minterms of G1 must be covered in the reduced cube set as well. Function G1 in our example is specified by five cubes, Table 2.

In the 2nd decomposition step we repeat the same procedure: the input blanket γ for the subset X\{x1,x2} consists of four blocks that can be merged to only three. Functions H2 and G2 are obtained as before, the rest of procedure is straightforward.

By now, we have obtained a sequence of detached functions H1 to H4 that can be implemented by four layers of a MTBDD. Construction of the MTBDD starts from leaves and goes left to the root, Fig. 1. The new id is used as a decision node label and integer values of the detached function $H|_{x=0}$, $H|_{x=1}$ for two values of the detached variable x are labels of successor nodes.

Table 2. Iterative decomposition procedure

β	x1 0 1	new id	
1,2,8,9	0, 3	0	H1
1,3,9	0, 3	0	
4	2, -	1	
3	-, 3	0	$\rightarrow$
2,8	0, 3	0	
6	1, 1	2	
4,5	2, 2	1	
6,7	1, 1	2	

	x2	x3	x4	G1
1	0	0	-	0
2	-	0	0	0
3	0	-	1	0
4	-	1	0	1
5	1	-	1	2

G1

γ	x2 0 1	new id	
1,2	0, 0	0	H2
1,3,5	0, 2	1	$\rightarrow$
4	1, 1	2	
3,5	0, 2	1	

	x3	x4	G2
1	0	0	0
2	-	1	1
3	1	0	2

δ	x3 0 1	new id	
1,3	0, 2	1	H3
2	1, 1	0	$\rightarrow$

	x4	G3
1	0	1
2	1	0

ε	x4 0 1	new id	
1,2	1, 0	0	H4

0

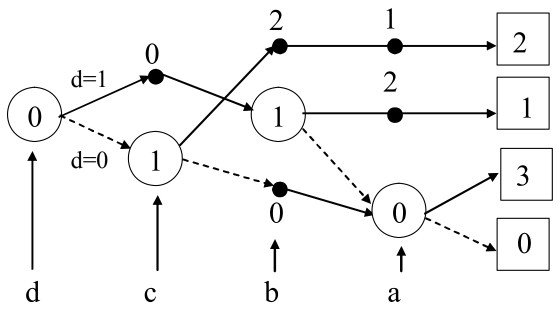

Fig. 1. MTBDD as a result of disjunctive iterative decomposition

4. Heuristic iterative decomposition

The remaining question not addressed as yet is, which variable should be used in any given step. We use a heuristics that strives to optimize one level of the MTBDD at a time. There are more sophisticated heuristics such as sifting where the window of several variables is moved from the root to leaves in order to optimize the position of several variables at a time [3]. However, if desired, our approach could be extended to more than one detached variables.

There are three parameters of MTBDDs that can be optimized: size (cost), width and an average path length (APL). For firmware implementation of MTBDDs, only cost minimization is of interest (degenerate nodes do not count). In our heuristics we minimize the number of regular nodes level by level, from leaves to the root. We expect that the total cost will be close to the minimum total cost. That is why we also talk about suboptimal MTBDD synthesis.

If hardware LUT cascades are to be generated, a slightly modified optimization criterion is required. The main concern is the cascade width defined as the number of binary rails connecting adjacent LUTs. It is related to the MTBDD width, the number of edges between adjacent levels of the diagram, as

$$\#\text{ rails} = \lceil \log_2 (\#\text{ edges}) \rceil. \quad (5)$$

The following algorithm minimizes the local width:

Algorithm 4.1. Selection of a variable to minimize the local width/cost of the MTBDD.

```
step ← 0;
do {
  repeat the simple disjunctive decomposition with
  each of (n – step) variables;
  apply the minimum local width criterion: select the
  variable that generates the minimum number of new
  id numbers (labels of decision nodes, LUT rows);
  in case of a tie and (firmware optimization only):
  apply the lowest cost criterion: select a variable pro-
  ducing the lowest number of output pairs of type [u,
  v] – true decision nodes;
  in case of a tie again, select one variable randomly;
  step ← step +1;
} while step < n-1.
```

To aid MTBDD synthesis, the program tool HIDET has been developed [7]. The first version accepted only completely specified integer-valued fr functions specified by disjoint cubes. The new version adopting the algorithm from Section 3 (now under construction) allows incomplete fr functions and compatible input cubes. As functions with many don’t cares are quite common, this is important innovation.

6. Conclusion

The presented method of heuristic MTBDD synthesis of multiple-output Boolean functions concentrated on theoretical basis of the more general key algorithm of HIDET accepting incomplete integer-valued fr functions specified by possibly compatible cubes. Decomposition used in the algorithm is based on notion of blankets. The procedure to minimize the local width or cost of diagrams has been formally presented. It is related to minimum cover of maximal compatibility classes that is in our case not too difficult to find.

Future research will be devoted to experiments with the upgraded version of HIDET and possible to its further extension, namely replacing integer values by ternary output cubes. This would make possible to clarify applications best suitable for MTBDDs on one hand and for BDD_for_CF diagrams on the other.

Acknowledgement

This research has been carried out under the financial support of the research grants “Design and hardware implementation of a patent-invention machine”, GA102/07/0850 (2007-9), “Safety and security of networked embedded system applications”, GA102/08/1429 (2008-10), "Mathematical and Engineering Approaches to Developing Reliable and Secure Concurrent and Distributed Computer Systems" GA 102/09/H042 (2009-12), all in care of Grant Agency of Czech Republic, and “Security-Oriented Research in Information Technology”, MSM 0021630528 (2007-13).

References

[1] M.J.S. Smith, Application-Specific Integrated Circuits. Addison-Wesley Longman Inc., New York, p.1026, 1997.

[2] R. Drechsler, B. Becker, Binary Decision Diagrams - Theory and Implementation. Springer, 1998.

[3] Yanushkevich, S. N., Miller, D., M., Shmerko, V. P., Stankovic, R. S., *Decision Diagram Techniques for Micro- and Nanoelectronic Design*. CRC Press, Taylor & Frasnis Group, Boica Raton, FL, 2006.

[4] T. Sasao, Y. Iguchi and M. Matsuura, "Comparison of decision diagrams for multiple-output logic functions," International Workshop on Logic and Synthesis (IWLS2002), New Orleans, Louisiana, June 4-7, 2002, pp.379-384.

[5] T. Sasao and M. Matsuura: BDD representation for incompletely specified multiple-output logic functions and its applications to functional decomposition, *Design Automation Conference*, pp.373-378, June 2005.

[6] http://tams-www.informatik.uni-hamburg.de/applets/

[7] Mikušek Petr, Dvořák Václav: On Lookup Table Cascade-Based Realizations of Arbiters, In: *11th EUROMICRO Conference on Digital System Design* DSD 2008, Parma, IT, IEEE CS, 2008, pp. 795-802.

[8] Brzozowski, J.A., Luba, T.: Decomposition of Boolean Functions Specified by Cubes. *Research report* CS-97-01, University of Waterloo, Canada, p.36, 1997.