Exact Minimum Logic Factoring via Quantified Boolean Satisfiability

Hiroaki Yoshida†, Makoto Ikeda‡ and Kunihiro Asada‡
†Department of Electronic Engineering
‡VLSI Design and Education Center(VDEC)
University of Tokyo
7-3-1 Hongo, Bunkyo-ku, Tokyo 113-8656, Japan
email: {hiroaki,ikeda,asada}@silicon.u-tokyo.ac.jp

Abstract— **This paper presents an exact method which finds the minimum factored form of an incompletely specified Boolean function. The problem is formulated as a Quantified Boolean Formula (QBF) and is solved by general-purpose QBF solver. We also propose a novel graph structure, called an X-B (eXchanger Binary) tree, which implicitly enumerates binary trees. Using this graph structure, the factoring problem is compactly transformed into a QBF and hence the size of solvable problems is extended. Experimental results show that the proposed method successfully finds the exact minimum solutions to the problems with up to 12 literals in ten minutes.**

I. Introduction

Logic factoring is an operation to find a factored form from a Boolean function. The factored form is known as one of the efficient representation styles of Boolean functions and forms a basis of multiple-level logic. Since the factored form corresponds to a static CMOS compound gate (Fig. 1), it has also been used for a number of applications such as primitive logic cell design and the estimation of circuit area required to implement a Boolean network.

Although logic factoring is one of the most fundamental problems in multiple-level logic synthesis, the exact solution to this problem is still challenging. Early works provided exact multiple-level logic minimization algorithms, however, the algorithms are very inefficient and apply to very small functions [1]-[3]. In addition to the fact, the algorithms of [2] [3] are designed to synthesize a circuit consisting of 2-input NAND gates and hence they cannot be applied to the minimization of the number of literals in the factored form.

Recently, Boolean satisfiability solvers have made a dramatic improvement [4][5] and have been successfully applied to industrial-scale EDA problems such as automatic test pattern generation [6] and symbolic model checking [7]. Quantified Boolean formula (QBF), which is a generalization of propositional logic, is known as a more natural way to model such problems. As a consequence, a number of efficient QBF decision algorithms have been proposed [8]-[11].

In this paper, we present an exact factoring method which transforms the factoring problem into a QBF and solves it using general-purpose QBF solver. Since the size of the QBF dominantly determines the runtime of the decision procedure, the QBF must be constructed as compact as possible. For this purpose, we propose a novel graph structure, called as an X-B tree, which compactly and implicitly enumerates all possible binary trees. Experimental results show that the proposed method generated the exact minimum solutions to the problems with up to 12 literals in ten minutes.

II. X-B Tree and Its Generation

A. X-B Tree

A *binary tree* is a graph with *internal nodes* and *leaf nodes* in which every internal node has two children. A *child* is either an internal node or a leaf node. An *X-B (eXchanger Binary) tree* is a rooted unordered binary tree with a new type of nodes, called an *exchanger node*. An exchanger node has the same number of inputs $\{i_1, ..., i_n\}$ and outputs $\{o_1, ..., o_n\}$. Only one of the children is

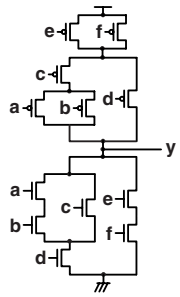


Fig. 1. A static CMOS compound gate implementing the complementation of a factored form expression $y = ((ab + c) * d) + ef$.

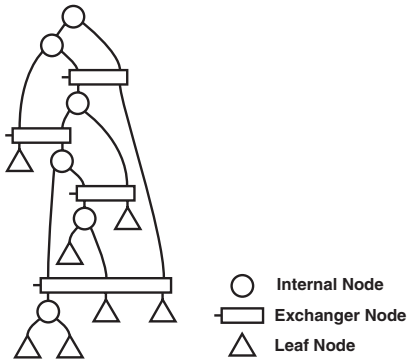


Fig. 2. An X-B tree with 7 leaf nodes.

an internal node, and the others are either a leaf node or an exchanger node. An exchanger node has an associated value, called an *exchange index* which determines how the inputs and the outputs are connected. An exchange index c_x is a positive integer $1 \leq c_x \leq n$ where n is the number of the inputs (outputs) of the exchanger. Then, the relation between the inputs and the outputs is given as follows:

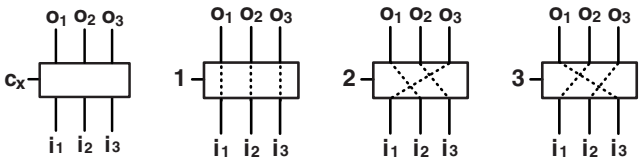
$$o_j = i_{((c_x+j-2) \bmod n)+1} \quad (1 \leq j \leq n).$$

where i_j is the j-th input and o_j is the j-th output. An exchanger node can be viewed as a bit shifter. An X-B tree with 7 leaf nodes is shown in Fig. 2 and a 3-input exchanger node is shown in Fig. 3.

Once the assignment of the exchange indices is determined, a binary tree is obtained. By exploring such assignments, all possible binary trees can be obtained. Note that there is not necessarily a one-to-one correspondence from an assignment of exchange indices to a binary tree, *i.e.*, different assignments of exchange indices can result in the same binary tree.

B. Signatures of Binary Trees

During the construction of X-B trees, it is necessary to check the isomorphism between two binary trees. We use the *bitstring representation* [12][13] as the signature of binary trees. Since the original bitstring representation is for ordered binary trees, we extend



(a) (b) $c_x = 1$ (c) $c_x = 2$ (d) $c_x = 3$

Fig. 3. An example of 3-input exchanger node.

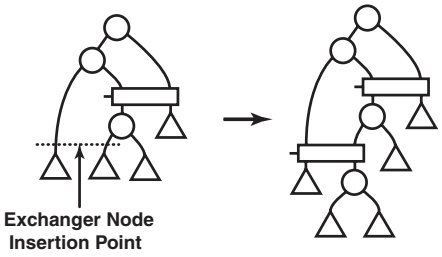


Fig. 4. Inserting an exchanger node.

it for unordered binary trees. The bitstring representation is a binary sequence $b_1 b_2 \ldots b_{2n}$ obtained recursively, as described in Fig. 5. In the procedure, the last bit is always 0 and hence is omitted.

C. Generating X-B Trees

Given the number of leaf nodes, there are many choices of X-B trees depending on how the exchanger nodes are connected. Since we are particularly interested in the minimum X-B tree, all possible X-B trees are first enumerated and the minimum one is chosen.

The proposed generation procedure is constructive in the sense that the X-B trees with L leaf nodes are constructed from the X-B trees with $L-1$ leaf nodes. The procedure starts with a binary tree with one internal node and two leaf nodes. Given an X-B tree, a set of leaf nodes and their parents is identified as an insertion point. Then, an exchanger node and an internal node are inserted at the point, as illustrated in Fig. 4.

The insertion points are computed as follows. First, a leaf node is replaced with an internal node and two leaf nodes. Then, all the signatures are computed by exploring all possible assignments of the exchange indices. After computing the signatures for all leaf nodes, a covering table is constructed where the rows correspond to the signatures and the columns to the leaf nodes. By solving the covering problem, the minimum set of leaf nodes is found and an exchanger node is inserted between the leaf nodes and their parents.

Table I shows the characteristics of the minimum X-B trees. In the QBF formulation described in the next section, an exchange index is represented as a bit-vector (*i.e.* a set of binary variables). The fourth column in the table corresponds to the total number of bits (binary variables) required to represent all the exchange indices in an X-B tree. An upper bound on the number of total exchange bits is $O(L \log_2 L)$. where L is the numbers of leaf nodes.

III. Exact Minimum Factoring

A. Problem Formulation

A *literal* is a variable or its negation. A *factored form* is a representation of a Boolean function and defined recursively as follows: 1) a literal is a factored form; 2) a sum of factored forms is a factored form; 3) a product of factored forms is a factored form. In general, the factored form of a Boolean function is not unique. For example, the following expressions; $abc + abd + cd$, $ab(c + d) + cd$ and $abc + (ab + c)d$ are all the factored forms of a Boolean function. A factored form is *minimum* if and only if the number of literals is the least among all possible factored forms.

```
Procedure ComputeSignature(Tree T)
    S=ComputeSignatureRecursive(GetRootVertex(T))
    return OmitLastBit(S)
end Procedure

Procedure ComputeSignatureRecursive(Vertex V)
    if V is a leaf node
        return "0"
    end if
    S_L=ComputeSignatureRecursive(GetLeftChild(V))
    S_R=ComputeSignatureRecursive(GetRightChild(V))
    if S_L > S_R
        return "1"+S_L+S_R
    else
        return "1"+S_R+S_L
    end if
end Procedure
```

Fig. 5. Basic procedure for signature computation.

TABLE I
Characteristics of minimum X-B trees.

#leaf nodes	*#internal nodes*	*#exchangers*	*#total exchange bits*	*#encoded binary trees*
2	1	0	0	1
3	2	0	0	1
4	3	1	1	2
5	4	2	2	3
6	5	3	3	6
7	6	4	5	11
8	7	5	7	23
9	8	6	9	46
10	9	7	11	98
11	10	8	13	207
12	11	9	15	451
13	12	10	17	983
14	13	11	20	2179
15	14	12	22	4850
16	15	13	25	10905

An AND/OR binary tree is a rooted binary tree where the type of each internal node is either a 2-input AND operator or a 2-input OR operator. By regarding leaf nodes as literals, an arbitrary factored form can be represented as an AND/OR binary tree.

The problem addressed in this paper can be formulated as follows: *Given an incompletely specified Boolean function* (f, d, r) *of variables* $V = \{v_1, \ldots, v_{|V|}\}$*, find a factored form with the minimum number of literals.* Alternatively, we can formulate it as follows: *Given an incompletely specified Boolean function* (f, d, r)*, find an AND/OR binary tree with the minimum number of leaf nodes which implements the Boolean function.*

B. Constructing a Quantified Boolean Formula

The problem is modeled as a miter structure [14] illustrated in Fig. 6. It checks the equivalence between the given Boolean function and an AND/OR X-B tree. An AND/OR X-B tree is an X-B tree with the following modifications. An internal node, called as an *operator node*, has its associated variable $c_o \in \{0, 1\}$ to specify whether the node type is AND or OR. Fig. 7 shows an operator node and its equivalent logic circuit. A leaf node, called as a *literal node*, has its associated variable $c_l \in \{1, \ldots, 2|V|\}$ to specify a literal $l \in \{v_1, \overline{v_1}, \ldots, v_{|V|}, \overline{v_{|V|}}\}$.

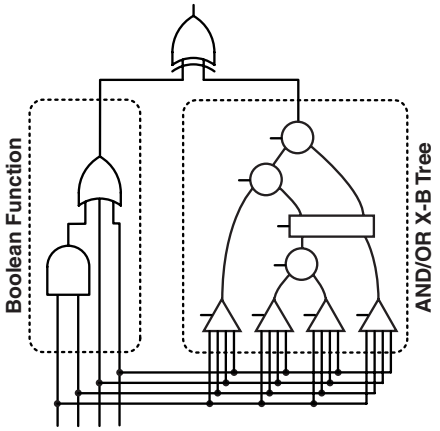


Fig. 6. A miter structure.

Fig. 8 shows a literal node and its equivalent logic circuit. The three types of variables, c_x, c_o and c_l, are called as *configuration variables* $C = \{c_1, \ldots, c_{|C|}\}$. An arbitrary AND/OR binary tree with a given number of leaf nodes can be represented by an AND/OR X-B tree with an assignment of the configuration variables.

A quantified Boolean formula is constructed based on this model. The clauses of the quantified Boolean formula consist of four categories: *function constraints*, *operator node constraints*, *exchanger node constraints* and *literal node constraints* where each constraint corresponds to a node in the miter structure.

1) Function Constraints: Let o_{root} be the variable corresponding to the output of the root operator node in the AND/OR X-B tree. The function constraints check the equivalence of the Boolean function and the AND/OR X-B tree:

$$\xi_f = (f \equiv o_{root}) + d$$

where f and d are the on set and the *don't care* set of the given Boolean function, respectively.

2) Operator Node Constraints: The operator node constraints represent the operator nodes in the AND/OR X-B tree. For each operator node, the following formula is constructed:

$$\xi_o = \overline{c_x}\Big(o \equiv (i_1 + i_2)\Big) + c_x\Big(o \equiv (i_1 \cdot i_2)\Big)$$

where i_1 and i_2 are the variables corresponding to the outputs of the child nodes of the operator node.

3) Exchanger Node Constraints: Let n be a positive integer $1 \leq n \leq m$. Then, a cube representation of n is defined as follows:

$$\mathcal{CUBE}(x, m, n) = \prod_{i=1}^{\lceil \log_2 m \rceil} (x_i \equiv b_i)$$

where $b_1 b_2 \ldots b_{\lceil \log_2 m \rceil}$ is a binary bit-vector representation of a decimal integer $n-1$. For example, $\mathcal{CUBE}(4,1) = \overline{x_1} \cdot \overline{x_2}$, $\mathcal{CUBE}(4,2) = x_1 \cdot \overline{x_2}$, $\mathcal{CUBE}(4,3) = \overline{x_1} \cdot x_2$ and $\mathcal{CUBE}(4,4) = x_1 \cdot x_2$.

The exchanger node constraints represent the exchanger nodes in the AND/OR X-B tree. For each exchanger node, the following formula is constructed:

$$\xi_x = \sum_{i=1}^{n}\Big(\mathcal{CUBE}(c_o, n, i) \prod_{j=1}^{n}(o_j \equiv i_{((i+j-2) \bmod n)+1})\Big) \overline{\sum_{i=n+1}^{2^{\lceil \log_2 n \rceil}} \mathcal{CUBE}(c_o, n, i)}$$

where n is the number of the inputs (outputs) of the exchanger node, o_j is the variable corresponding to the j-th output of the operator node, and i_j is the variable corresponding to the output of the j-th child node of the operator node.

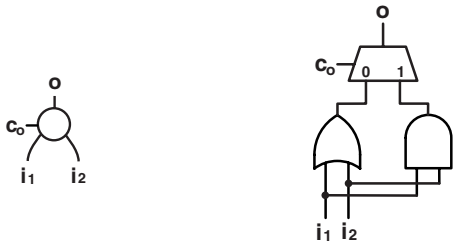


Fig. 7. (a) operator node and (b) its equivalent logic circuit.

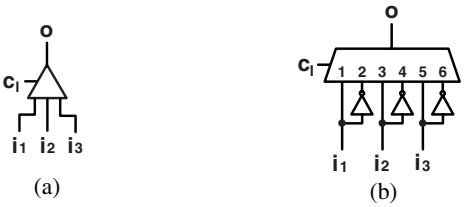


Fig. 8. (a) literal node and (b) its equivalent logic circuit.

4) Literal Node Constraints: The literal node constraints represent the literal nodes in the AND/OR X-B tree. For each literal node, the following formula is constructed:

$$\xi_l = \sum_{i=1}^{|V|}\Big(\mathcal{CUBE}(c_l, 2|V|, 2i-1)(o \equiv v_i) + \mathcal{CUBE}(c_l, 2|V|, 2i)(o \equiv \overline{v_i})\Big) \cdot \overline{\sum_{i=2|V|+1}^{2^{\lceil \log_2 2|V| \rceil}} \mathcal{CUBE}(c_l, 2|V|, i)}$$

Let $O = \{o_1, \ldots, o_{|O|}\}$ be the variables corresponding to the outputs of the exchanger, operator and literal nodes. Then, a quantified Boolean formula ξ is constructed by combining all the constraints and introducing existential and universal quantifiers:

$$\xi = \exists C \forall V \exists O \; \xi_f \left(\prod_{i=1}^{L-1} \xi_{o_i}\right)\left(\prod_{i=1}^{S} \xi_{s_i}\right)\left(\prod_{i=1}^{L} \xi_{l_i}\right)$$

where ξ_{o_i}, ξ_{s_i} and ξ_{l_i} are the constraints corresponding to the i-th node of each type, and L and S are the numbers of the leaf nodes and the exchanger nodes in the AND/OR X-B tree, respectively.

Upper bounds on the numbers of clauses and literals in ξ are given as $O(L^3)$ and $O(L^3 log_2 L)$, respectively. Due to space limitations, we do not provide the details of the derivations.

C. Finding the Minimum Factored Form

Given the number L of the leaf nodes in the AND/OR X-B tree, a QBF ξ is constructed as described in the previous section. If ξ is satisfiable, it implies that there is a factored form with L or less literals. To find the minimum factored form, we start with $L = |V|$ literals. If the QBF is satisfiable, the assignment of the configuration variables is computed and the minimum factored form is obtained. Otherwise, L is incremented by one and the procedure is repeated until the minimum factored form is obtained. Note that there does not exist any factored form with less than $|V|$ literals because every variable must appear in the factored form.

IV. Experimental Results

We have implemented the proposed method called *Exact Factor* in C++ on top of the logic manipulation class library *Logica* which we have recently developed. The platform was a Linux system on AMD Athlon 64 X2 4400 processor with 2 GB main memory. As a QBF

TABLE II

EXPERIMENTAL RESULTS ON BENCHMARK FUNCTIONS.

Function	*#inputs*	*ESPRESSO* [15]	*Good Factor* [16]	*Exact Factor*				
		#literals	*#literals*	*#literals*	*QBF*			*CPU time*
					#variables	*#clauses*	*#literals*	*[sec]*
majority	5	13	10	**9**	90	251	825	2.6
algebraic1	8	14	8	8	81	245	991	1.8
algebraic2	6	36	11	11	115	458	2014	483.9
algebraic3	6	15	11	**10**	103	409	1796	200.7
algebraic4	9	19	9	9	103	299	1046	12.0
boolean1	5	11	8	**6**	54	194	813	0.5
boolean2	6	26	16	**10**	103	413	1817	65.0
boolean3	5	13	10	**8**	78	287	1176	10.2
boolean4	6	22	18	**11**	115	458	2014	466.3
boolean5	6	30	20	**12**	127	383	1349	319.9

solver, we have examined a number of state-of-the-art QBF solvers: ssolve [8], SEMPROP [9], sKizzo [10], and Quantor [11]. Among these solvers, we chose sKizzo which solved our QBF problem instances in the shortest runtime.

We conducted experiments on a set of artificially-created benchmark functions and a function `majority` from MCNC91 benchmark suite because most of the well-known benchmark circuits are too big for our experiments. The artificial benchmark functions are categorized into two groups: *algebraic group* and *Boolean group*. The functions in the algebraic group are the functions of which the minimum factored form can be obtained without the specific features of Boolean algebra. In contrast, the functions in the Boolean group are the functions of which the minimum factored form can be obtained only if the specific features of Boolean algebra are used.

The results are shown in Table II. As a reference, we present the results of ESPRESSO two-level minimizer [15] and Good Factor factoring algorithm [16]. In the table, the first two columns give the name of the function and the number of the input variables, respectively. Columns 3, 4 and 5 show the numbers of literals of the sum-of-products form generated by ESPRESSO, the factored form generated by Good Factor, and the factored form generated by the proposed method. Columns 6, 7 and 8 show the numbers of variables, clauses and literals of the final satisfiable QBF. The last column shows the CPU time in seconds.

Since it is guarateed that the proposed method provides the minimum factored form, we focus particularly on the quality of the results and the runtime. As can be observed, the Good Factor provides near-minimum results on the functions in the algebraic group. However, on the functions in the Boolean group, the results of the Good Factor are far from the minimum solution. For example, the resulting expressions of `boolean4` are as follows:

ESPRESSO: $ab\overline{c}\,\overline{d} + ab\overline{e} + \overline{a}\,\overline{b}cd + \overline{a}\,\overline{b}ef + cd\overline{e} + \overline{c}\,\overline{d}ef$

Good Factor: $cd\overline{e} + \overline{c}\,\overline{d}ef + \overline{a}\,\overline{b}(ef + cd) + ab(\overline{e} + \overline{c}\,\overline{d})$

Exact Factor: $(ab + cd + ef)(\overline{a}\,\overline{b} + \overline{c}\,\overline{d} + \overline{e})$

It can also be observed that the size of QBFs does not simply follow the number of variables or literals. The reason is that our implementation of the logic factoring method performs some simple reductions on the QBF size as a preprocessing step if a redundancy is found in a given problem (*e.g.* a given Boolean function is unate). The proposed method could not solve the problems bigger than 12 literals in *ten minutes* mainly due to large CPU time of QBF satisfiability checking. QBF decision algorithm is under a heavy development and is still improving further. Hence it should be expected that bigger problems are solved in the near future.

V. CONCLUSIONS

Logic factoring is a fundamental but still challenging problem in multiple-level logic synthesis. In this paper, we presented an exact method which finds the minimum factored form of an incompletely specified Boolean function. The problem is formulated as a quantified Boolean formula and is solved by general-purpose QBF solver. We also proposed a novel graph structure, called an X-B tree, which enables to represent the factoring problem in a compact QBF. Experimental results showed that the proposed method successfully found the exact minimum solutions to the problems with up to 12 literals. Even though the size of solvable problems is limited, the proposed method is still useful for a number of practical applications such as primitive logic cell design.

REFERENCES

[1] E. L. Lawler, "An Approach to Multilevel Boolean Minimization," *Journal of ACM*, pp. 283–295, 1964.

[2] E. Davidson, "An Algorithm for NAND Decomposition under Network Constraints," *IEEE Trans. on Computers*, vol. 18, pp. 1098–1109, 1969.

[3] R. Drechsler and W. Gunther, "Exact Circuit Synthesis," *Int. Workshop on Logic Synthesis*, 1998.

[4] J. P. M. Silva and K. A. Sakallah, "GRASP—A New Search Algorithm for Satisfiability," in *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 220–227, Nov. 1997.

[5] M. Moskewicz *et al.*, "Chaff: Engineering an Efficient SAT Solver," in *Proc. of ACM/IEEE Design Automation Conference*, pp. 530–535, Jun. 2001.

[6] T. Larrabee, "Test Pattern Generation Using Boolean Satisfiability," *IEEE Trans. on Computer-Aided Design*, vol. 11, no. 1, pp. 4–15, Jan. 1992.

[7] A. Biere *et al.*, "Symbolic Model Checking without BDDs," in *Proc. of ACM/IEEE Design Automation Conference*, pp. 193-207, Jun. 1999.

[8] R. Feldmann *et al.*, "A Distributed Algorithm to Evaluate Quantified Boolean Formulas," in *Proc. National Conf. on Artificial Intelligence*, pp. 285–290, 2000.

[9] R. Letz, "Lemma and Model Caching in Decision Procedures for Quantified Boolean Formulas," in *Proc. TABLEAUX2002, vol. 2381 of LNAI*, pp. 160–175, 2002.

[10] M. Benedetti, "sKizzo: a QBF decision procedure based on Propositional Skolemization and Symbolic Reasoning," *Tech. Rep. TR04-11-03*, ITC-Irst, Nov. 2004.

[11] A. Biere, "Resolve and Expand," in *Proc. Intl. Conf. Theory and Applications of Satisfiability Testing*, LNCS, Springer 2005.

[12] A. Proskurowski, "On the Generation of Binary Trees," *Journal of ACM*, vol. 27, no. 1, pp. 1–2, Jan. 1980.

[13] S. Zaks, "Lexicographic Generation of Ordered Trees," *Theoretical Computing Science*, vol. 10, pp. 63–82, 1980.

[14] D. Brand *et al.*, "Verification of Large Synthesized Designs," in *Proc. IEEE Int. Conf. Computer-Aided Design*, pp. 534–537, Nov. 1993.

[15] R. K. Brayton *et al.*, Sangiovanni-Vincentelli, *Logic Minimization Algorithms for VLSI Synthesis*, Kluwer Academic Publishers, Boston, 1984.

[16] R. K. Brayton *et al.*, "MIS: A Multiple-level Logic Optimization System," *IEEE Trans. on Computer-Aided Design*, vol. 6, no. 6, pp. 1062–1081, Nov. 1987.