MORE: An Alternative Implementation of BDD Packages by Multi-Operand Synthesis [*]

Andreas Hett Rolf Drechsler Bernd Becker

Institute of Computer Science
Albert-Ludwigs-University
79110 Freiburg im Breisgau, Germany
email: <name>@informatik.uni-freiburg.de
WWW: http://www.informatik.uni-freiburg.de/FREAK

Abstract

In this paper we present a new approach for the realization of a BDD package. This approach does not depend on recursive synthesis operations, i.e. the ternary If-Then-Else-operator (ITE), to perform manipulations of Boolean functions; instead our basic operation MORE is based on exchanges of neighbouring variables and existential quantification. It is capable of combining an arbitrary number of Boolean functions in parallel. We discuss the difference between MORE and ITE and give experimental results to show the advantages of our implementation approach with respect to size and runtime.

1. Introduction

Ordered Binary Decision Diagrams (OBDDs) as introduced by Bryant in 1986 [4] are the state-of-the-art data structure in CAD. They have been successfully used in many applications, like verification and logic synthesis [12, 7, 5]. In the meantime they are also integrated in commercial tools for verification and FPGA design [1, 11].

One of the main advantages of OBDDs is that they can efficiently be implemented (see e.g. [2, 13]). All these packages are based on recursive synthesis algorithms similar to the *apply*-operation proposed in [4]. In [2] the *If-Then-Else*-operator (ITE) has been used, since all binary synthesis operations can easily be described by the use of ITE. Since ITE works well in many applications no alternative concepts have been proposed.

In this paper we present a completely new approach for the implementation of an OBDD package. Our method does not use recursive synthesis algorithms like ITE. It even does not use a *Computed Table* (CT). Instead our approach is based on an algorithm that uses exchange of neighbouring variables as the basic operation (which is also the basic operation for dynamic variable ordering). On the functional level the method is based on existential quantification, i.e. the approach uses *Multi-operand synthesis OR-operations based on Existential quantification* (MORE).

We first present the new method and then discuss the resulting advantages in comparison to ITE based OBDD packages. Our experimental results show the advantages of MORE: Using MORE it is possible to build OBDD representations for circuits for which ITE fails within reasonable upper bounds. Additionally, algorithms for dynamic minimization (see e.g. [8, 15, 14]) can easily be included, since MORE supports fast exchange of neighbouring variables as the basic operation. Experiments show that MORE is not only capable of handling symbolic simulation with a smaller number of nodes, even the runtimes are improved.

2. Ordered Binary Decision Diagrams

In the following we briefly review the essential definitions and properties of BDDs. For more details see [4, 5].

As well-known each Boolean function $f : \mathbf{B}^n \rightarrow \mathbf{B}$ can be represented by a *Binary Decision Diagram* (BDD) [4], i.e. a directed acyclic graph where a Shannon decomposition is carried out in each node.

A BDD is called *ordered* if each variable is encountered at most once on each path from the root to a terminal and if the variables are encountered in the same order on all such paths. A BDD is called *reduced* if it does not contain vertices either with isomorphic sub-graphs or with both edges pointing to the same node.

[*]This work was supported in part by DFG grant Be 1176/8-2

For functions represented by reduced, ordered BDDs efficient manipulations are possible [4]. Furthermore, the size[1] of a reduced OBDD can be reduced if *Complement Edges* (CEs) are used [2]. Then a node is used to represent a function and its complement at the same time. (The use of CEs implies that the computation of the complement can be performed in constant time.) In the following only reduced OBDDs are considered and for briefness these graphs are called BDDs.

2.1. The If-Then-Else-Operator

The ternary *If-Then-Else*-operator (ITE) [2] forms the core of recursion based synthesis operations for BDDs. ITE is a Boolean function defined for three operands as follows:

$$ite(F, G, H) = F \cdot G + \overline{F} \cdot H$$

Since ITE can be used to implement all two-variable Boolean operations every recursive binary synthesis operation is a spezialized descendant of the ternary operation, e.g. $F + G = ite(F, 1, G)$. The recursive formulation

$$ite(F, G, H) = (v, ite(F_v, G_v, H_v), ite(F_{\overline{v}}, G_{\overline{v}}, H_{\overline{v}}))$$

where F_v and $F_{\overline{v}}$ denote F evaluated at $v = 1$ and $v = 0$, respectively, (the same holds for G and H) provides an algorithm for the computation of the operation. For more details see [2].

2.2. The Computed Table

The *Computed Table* (CT), mostly a hash-based cache [2], maps the three operands F, G and H of an ITE-function call to the resulting node $ite(F, G, H)$ once this result has been computed. In [2, 13] it has been shown that it is possible to implement a DD-package that realizes lookup and insert processes in the CT and *Unique Table* (UT)[2] in constant time. Thus, the time complexity for the ITE-function can be given as follows:

> Observing that ITE can be called at most once for each combination of nodes in F, G and H due to the storage of calculated operations in an ideal[3] CT the overall time complexity of ITE is $O(|F| \cdot |G| \cdot |H|)$. For binary operations only two operands are non-terminals. Thus, a binary operation has the time complexity $O(|F| \cdot |G|)$.

Circuit	Time of ITE [s]	
	With CT	Without CT
c0017	0.0	0.0
c0432	18.3	172.1
c0880	1.0	149.2
c1908	35.5	2940.5
c3540	218.2	628.2
Other c... benchmarks	...	> 1 CPU hour

Table 1. Construction time for BDDs using ITE with and without a CT

The influence of the CT is shown in Table 1. One can observe that only a minority of the ISCAS'85 [3] benchmarks can be constructed as BDDs in appropriate time without a CT.

2.3. Dynamic Variable Reordering

In this section we briefly review the basics of a dynamic reordering method called *Level Exchange* (LE) which was introduced in [8], since this is the basic operation for our new synthesis method in the next section.

In order to modify the variable ordering of BDDs we can exchange variables in adjacent levels. Since an exchange of neighbouring variables is a local operation [8, 15] consisting only of the relinking of nodes in these two levels, this can be done very efficiently. In [15] it was shown that it is even possible to perform LEs as local operations when using BDDs with CEs.

3. MORE

We introduce the *Multi-Operand synthesis OR-operation based on Existential quantification* (MORE). First the principle idea will be explained by the synthesis of two BDDs. Then it is shown that the operation can easily be expanded to an *arbitrary* number of operands.

3.1. The Basic Idea

Assume that two BDDs F and G of size $|F|$ and $|G|$ defined over a set of variables $X_n := \{x_1, \cdots, x_n\}$ for the Boolean functions f and g are given. They are to be combined to result in the BDD R for the Boolean function $r = f + g$. We therefore introduce a new variable c that is called *coding variable* and construct the BDD R' as shown for two examplary functions f

[1]The *size* of a BDD F denoted by $|F|$ equals the number of its non-terminal nodes.

[2]The *UT* is essential for the storage and access of BDD nodes. To allow efficient level exchange operations (see also Subsection 2.3) the table gets divided in segments for each variable, suggesting the expression '*UTs*'.

[3]An ideal *Computed Table* can store every operation result and thus every operation needs to be calculated only once. However, since for a CT only a limited number of calculated operations can be stored due to memory limitations, older entries must be deleted to free resources for newer ones. Thus, it may occur that operations have to be recalculated since the appropriate entry is no longer present.

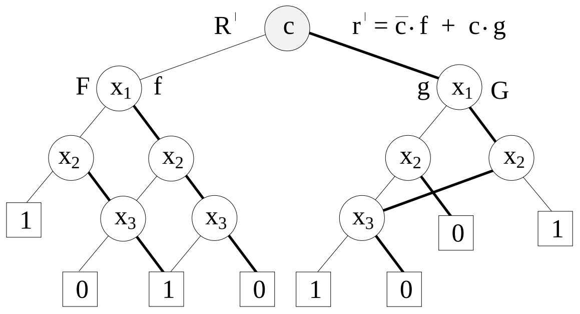

Figure 1. Introduction of *coding variable c* **to represent the Boolean function** $r' = \overline{c} \cdot f + c \cdot g$

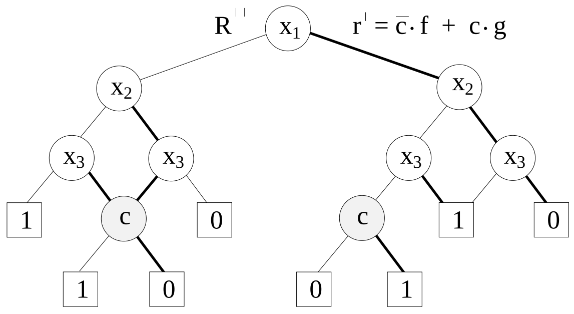

Figure 2. *Coding variable* **shifted into the bottom level**

and g in Figure 1[4] representing the function $r' = \overline{c} \cdot f + c \cdot g$. The construction of R' can be done in constant time if the BDDs F and G are given. The existential quantification

$$\begin{aligned} \exists c : r' &= (\overline{c} \cdot f + c \cdot g)_{c=0} + (\overline{c} \cdot f + c \cdot g)_{c=1} \\ &= f + g \\ &= r \end{aligned}$$

obviously leads to the desired function r although at this point we would have achieved no improvement since the calculation still requires the recursive OR-synthesis. However performing an existential quantification can be very simple if we behold the following facts:

- We can always change the variable ordering of a BDD without changing the Boolean function it represents. Thus, it is possible by means of LEs that are local operations (see Section 2.3) to shift the *coding variable* c into the bottom level of BDD R' and still represent function r' (see Figure 2).
- Each path p from the root of the BDD R'' to a terminal vertex represents a Boolean function $cube_p = m_p \cdot z$ where $m_p \in \{x_{p_1}{}^{a_{p_1}} \cdot \ldots \cdot x_{p_k}{}^{a_{p_k}} | x_{p_j} \in X_n, a_{p_j} \in \{0,1\}, 1 \leq j \leq k, 1 \leq k \leq n\}$ and $z \in \{c, \overline{c}, 1\}$.
- The Boolean function represented by BDD R'' equals the Boolean sum of all these cubes.
- The existential quantification $\exists c : cube_p = \exists c : (m_p \cdot z)$ equals m_p for all paths p.
- The Boolean sum of $\exists c : cube_p$ for all paths p equals the Boolean function $f + g$.

Due to the shift of the *coding variable* c to the bottom level it is always present last (if at all) in a path

[4]Note: Since the reduction rules identify isomorphic subgraphs, the depicted BDDs actually share some of the represented nodes. However, for a clear separation of the two operands f and g they are presented without sharing. Note also that bold edges denote the high edges.

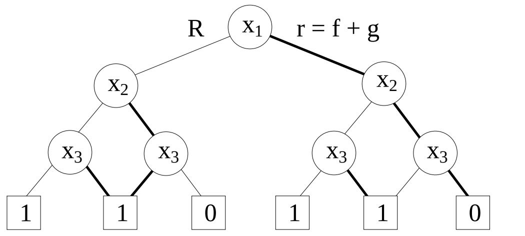

Figure 3. Existential quantification $\exists c : r' = f + g = r$ **done in bottom level**

and thus the existential quantification for each $cube_p$ can be performed by simply linking all edges that point to a node with variable c to the terminal one-node resulting in the representation of cube m_p. Therefore the following holds:

> The binary OR-synthesis of f and g can be performed by inserting the *coding variable* c at the top. Then c is shifted to the bottom level and a relinking of all edges that point to a node with variable c to the terminal one-node is performed which can be done by a traversal of the BDD. The resulting BDD R represents $f + g$ as is shown in Figure 3.

Since we want to maintain canonicity of the data structure, a reduction of the resulting BDD may be needed which can be performed in $O(|R|)$ [16]. The resulting BDD is shown in Figure 4.

MORE is capable of handling any binary synthesis operation: As is well-known, the negation of a Boolean function represented by a BDD G can be performed in O($|G|$). When using CEs it can even be done in constant time [2]. Since the NOT- and OR-operation are sufficient to perform any binary synthesis operation and CEs are applied in our BDD package, MORE is capable of replacing *all* recursive synthesis operations.

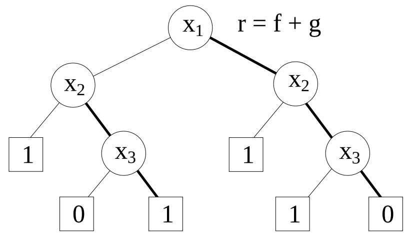

Figure 4. Reduction rules applied to regain a canonical representation of r

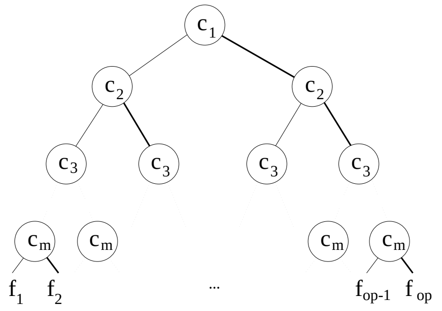

Figure 5. *Coding Tree* **with** $\lceil log_2(op) \rceil$ **coding variables** $\{c_1, \ldots, c_m\}$ **to combine** op **Boolean functions**

3.2. Expanding MORE

We can easily expand the number of operands for MORE by introducing $\lceil log_2(op) \rceil$ *coding variables* for the combination of *op* BDDs of arbitrary size using a *Coding Tree* as shown in Figure 5. The algorithm keeps as simple as for a single *coding variable* since we can shift each of them towards the bottom level until they are in the variable ordering below each variable of X_n. A sketch of the algorithm is given in Figure 6.

The power of MORE gets more obvious for an increasing number of operands since the runtime amortizes when handling more complex operations while recursive synthesis operations grow steadily, i.e. MORE needs $\lceil log_2(op) \rceil$ shift operations while ITE has to perform $(op - 1)$ synthesis operations.

3.3. MORE than ITE?

In this subsection we discuss the characteristics of MORE and compare our method to the recursive approach.

Since ITE works very efficiently in combination with a well-organized CT there seems to be no need for an

```
MORE(operand_list consisting of op BDDs)
{
  Build Coding Tree by use of $\lceil log_2(op) \rceil$ coding variables;

  for each coding variable do
  {
    Shift coding variable into bottom level
        by means of Level Exchanges;

    // Traverse BDD and eliminate coding variable
    Perform existential quantification; Reduce result tree;
  }

  return result tree;
}
```

Figure 6. MORE algorithm

alternative approach. However, a very important drawback gets obvious if we behold the following facts:

- Since physical resources for any program are limited, there exists an upper size limit for any BDD package (e.g. the number of existing nodes is limited to 1,000,000).
- Since the crossing of this physical boundary is forbidden, the current ITE-operation has to be negated if the crossing of this boundary is impending.
- It is not possible to interrupt an ITE-operation in order to change the variable ordering without the loss of subgraphs that were calculated during its recursive process to gain the final result.
- The number of existing nodes during an ITE-operation is higher (especially for large circuits) than the number of nodes needed to represent the resulting function. (For some of the ISCAS'85 [3] benchmarks we measured the maximal number of additional nodes needed (at the same time) during ITE-operations while constructing the BDDs. Experimental results are shown in Section 4.)
- The runtime of ITE in comparison to MORE is often much larger (see Section 4).

To handle an impending boundary crossing, ITE must protocol every step taken to perform a synthesis operation in order to make a reversion of them possible. After this reversion a minimization of the BDDs (e.g. by means of dynamic variable reordering algorithms) can be done and the ITE-operation can be retried in hope that it will succeed this time.

Since MORE is based on LEs, an impending boundary crossing can be handled simply by interrupting the shift process of a *coding variable* and rearranging the variables, e.g. minimizing the BDDs by optimizing the variable ordering for the blocks of variables above and below the level where the shifted *coding variable* is currently positioned (see Figure 7). Then MORE will continue right at the point where it was suspended. Thus,

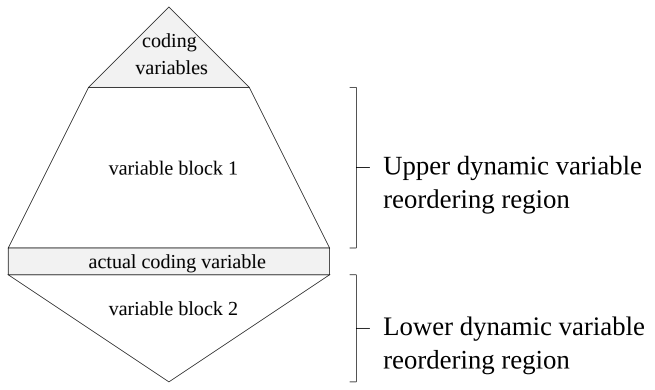
coding
variables
variable block 1
actual coding variable
variable block 2
Upper dynamic variable reordering region
Lower dynamic variable reordering region

Figure 7. Interruption of MORE and optimization of the variable ordering

MORE works with higher granularity allowing to construct BDDs with a better controllability of resources.

We have presented experimental results (see Section 2.2) to show that a *Computed Table* is essential to gain a good performance of ITE.

The performance of MORE does not depend on any assistent data structure except the *Unique Table*, thus spares the resources (usually some megabytes are needed to maintain an efficient CT for large BDDs) and gets an upper size limit bonus of approximately 30,000 nodes[5] for free.

4. Experimental Results

In this section we present a set of experimental results. All experiments were carried out on a *SUN SPARCstation 20* workstation with a time limit of 1 CPU hour in the package environment of [10]. The variable ordering was determined by a method similar to *Interleaving* [6]. In the case of ITE a CT with 10,000 anchors and a maximal chain-length of 3 was applied (i.e. 30,000 entries at a time can be stored). In Table 2 the runtime for ITE and MORE (denoted by *Time* and given in CPU seconds) for the ISCAS'85 [3] benchmarks are shown. Additionally the size of the resulting graphs are given in column *Size*[6]. In the columns *Add.Nodes* the number of additional nodes needed during a synthesis operation is stated for ITE and MORE, respectively (see also Figure 8).

Due to the number of nodes needed for intermediate results, i.e. BDDs that have to be kept in storage until a recursion is solved in ITE, the amount of additional nodes turned out to be generally higher for ITE than

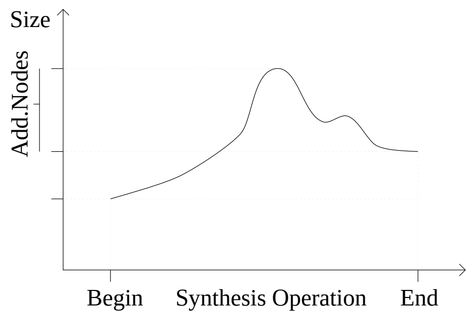

Figure 8. Size modification during synthesis operations

for MORE, where multi-operand synthesis operations are applied. The crossing of an upper size limit is therefore more likely. The consequences in this case were already stated in Section 3.3. Note that the presented results in the columns *Add.Nodes* are highly sensitive to strategies used during symbolic simulations.

Also with respect to runtime MORE has several advantages in comparison to ITE:

- Even with a non-optimized approach of MORE (i.e. a terminal case handling (see below) is currently not fully implemented) the synthesis can be performed within the same runtime in comparison to ITE. In the best cases MORE even performed with a time factor bonus of more than 20 (e.g. 'c5315') and could even solve the cases were ITE failed (i.e. needs more than 1 CPU hour) in less than one CPU minute.
- There exist several cases were the CT failed to assist ITE or (more often) the BDDs that have to be combined are simply too large and their logical structure is unsuitable and therefore the calculations of ITE consume very much time[7].

5. Conclusions and Future Work

In this paper we presented a new approach for the realization of BDD packages that do not depend on recursive synthesis operations; instead our basic operation MORE is based on exchanges of neighbouring variables and existential quantification. It is capable of combining an arbitrary number of Boolean functions in *parallel*.

All facts taken into account we conclude that MORE offers an easy to implement and easy to handle approach based on level exchange, i.e. a basic routine that

[5]This value largely depends on the basic data structure of the package implementation.

[6]Obviously the size of the resulting graph is equal for ITE and MORE.

[7]In the case of the benchmarks $'c499'$ and $'c1355'$ we also tested the construction without *Interleaving*. Altough the construction of the benchmark is much quicker in this case (less than a minute), MORE yields even smaller runtimes.

Circuit	Size	ITE		MORE	
		Time [s]	Add.Nodes	Time [s]	Add.Nodes
c0432	31177	18.3	21949	21.5	6869
c0499	40657	> 1 hour	no result	**12.6**	4946
c0880	8654	1.0	3536	1.4	3540
c1355	57945	> 1 hour	no result	**27.1**	6419
c1908	14073	35.5	5316	**12.9**	3261
c2670	9771	1.6	4052	5.1	4052
c3540	150672	218.2	56841	**114.1**	56837
c5315	84267	800.4	14873	**31.5**	14873
c7552	8481	5.1	2544	8.5	2006
s1196	4132	0.4	1017	0.8	698
s1423	14873	3.1	1411	**2.8**	1082
dalu	2392	1.1	513	2.4	330
rot	10497	2.7	13913	3.8	2844
frg2	1979	0.4	183	2.0	81

Table 2. Measurements for MORE and ITE

is available in all BDD packages. It is a resource sparing and interruptable alternative to manipulate BDDs. Our experimental results show that MORE does not only need a smaller amount of nodes for synthesis operations, even the runtimes are improved.

MORE offers a large variety of expansions for further improvement, which is part of our future work: The handling of terminal cases seems to be promising, i.e. considering terminal cases during the sifting of the coding variable. Since (in comparison to ITE) MORE can make profit out of more complex operations, a structural analysis of the circuit topology and the fusing of neighbouring gates should be capable of further reducing the runtime complexity. First promising results can be found in [9].

Since MORE works with a local core operation, i.e. the level exchange, the concept of shifting the *coding variables* to the bottom level can be easily adapted to be handled in a pipeline. Thus, a parallel BDD package that distributes variable level windows to the network computers should expand the horizon of BDD computations. Therefore it is the focus of our current work.

References

[1] D. Appenzeller and A. Kuehlmann. Formal verification of a PowerPC microprocessor. In *Int'l Conf. on Comp. Design*, pages 79–84, 1995.

[2] K. Brace, R. Rudell, and R. Bryant. Efficient implementation of a BDD package. In *Design Automation Conf.*, pages 40–45, 1990.

[3] F. Brglez and H. Fujiwara. A neutral netlist of 10 combinational circuits and a target translator in fortran. In *Int'l Symp. Circ. and Systems, Special Sess. on ATPG and Fault Simulation*, pages 663–698, 1985.

[4] R. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 8:677–691, 1986.

[5] R. Bryant. Symbolic boolean manipulation with ordered binary decision diagrams. *ACM, Comp. Surveys*, 24:293–318, 1992.

[6] H. Fujii, G. Ootomo, and C. Hori. Interleaving based variable ordering methods for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 38–41, 1993.

[7] M. Fujita, H. Fujisawa, and N. Kawato. Evaluation and improvements of boolean comparison method based on binary decision diagrams. In *Int'l Conf. on CAD*, pages 2–5, 1988.

[8] M. Fujita, Y. Matsunga, and T. Kakuda. On variable ordering of binary decision diagrams for the application of multi-level synthesis. In *European Conf. on Design Automation*, pages 50–54, 1991.

[9] A. Hett, R. Drechsler, and B.Becker. MORE optimization techniques. Technical report, Albert-Ludwigs-University, Freiburg, 1996.

[10] A. Hett, R. Drechsler, and B. Becker. *The DD Package PUMA - An Online Documentation.* URL www.informatik.uni-freiburg.de/FREAK/, 1996.

[11] V. Le, T. Besson, A. Abbara, D. Brasen, H. Bogushevitsh, G. Saucier, and M. Crastes. ASIC prototyping with area oriented mapping for ALTERA/FLEX devices. In *SASIMI*, pages 176–183, 1995.

[12] S. Malik, A. Wang, R. Brayton, and A. Sangiovanni-Vincentelli. Logic verification using binary decision diagrams in a logic synthesis environment. In *Int'l Conf. on CAD*, pages 6–9, 1988.

[13] S. Minato, N. Ishiura, and S. Yajima. Shared binary decision diagrams with attributed edges for efficient boolean function manipulation. In *Design Automation Conf.*, pages 52–57, 1990.

[14] S. Panda and F. Somenzi. What are the variables in your neighborhood. In *Int'l Workshop on Logic Synth.*, pages 5b:5.11–5.20, 1995.

[15] R. Rudell. Dynamic variable ordering for ordered binary decision diagrams. In *Int'l Conf. on CAD*, pages 42–47, 1993.

[16] D. Sieling and I. Wegener. Reduction of BDDs in linear time. *Information Processing Letters*, 48(3):139–144, 11 1993.