ABSTRACT

We present for the first time methods to minimize *BMDs exploiting don’t care conditions. These minimization methods can be used during the verification of circuits by *BMDs. By changing function values for input vectors, which are in the don’t care set, smaller *BMDs can be computed to keep peak memory consumption during *BMD construction as low as possible. Preliminary experimental results prove the methods to be very effective in minimizing *BMDs sizes.

1. INTRODUCTION

One of the most important tasks during the design of Integrated Circuits is the verification of an implemented circuit, i.e., the check whether the implementation fulfills its specification.

In the last few years several methods based on Decision Diagrams (DDs) have been proposed [15, 3, 14] to perform verification. The idea is to transform both implementation and specification to DDs. Then, due to the canonicity of the DD representation, the equivalence check for verification of a combinational circuit into a DD reduces to the check whether the implementation fulfills its specification.

To the best of our knowledge the heuristics presented in this paper are the first solution to this problem. For the minimization of DDs under don’t care conditions there is a number of methods in the literature, e.g. [8, 7, 5, 18, 17, 11]. However for *BMDs the problem seems to be more difficult, since due to the Davio decomposition in *BMDs a change of the function value for a single input vector (exploiting a don’t care for this input vector) has not only a “local effect” in the Decision Diagram, but can affect larger parts of the *BMD (see Section 2). A paper which has some relations to our work in this sense is [20]. In that work FDDs [12] are minimized (which are also based on Davio decompositions). In fact our first method 1 to minimize *BMDs (which are representations of integer–valued functions) is somewhat similar to the minimization of FDDs in [20] (FDDs are representations of Boolean functions). Another related paper is [19], which minimizes Reed–Muller forms. However the method from [19], which decides, whether to flip the value for a subset of coefficients in the Reed–Muller spectrum from 0 to 1 (1 to 0) or not, with the goal to maximize the number of zeros in the Reed–Muller spectrum, is not applicable when the values are integers as for functions represented by *BMDs.

We developed two different methods for the minimization of *BMDs under don’t care conditions. After Section 2, which gives some basic definitions and notations, we present these methods in Section 3 and in Section 4 we give preliminary experimental results to evaluate the approaches. The minimization results are very promising. The first method was able to reduce *BMD sizes by 75% on the average, the second even by 79%. Finally, Section 5 concludes the paper and gives directions for future research.

2. PRELIMINARIES

In this section we give a brief review of BDDs [2], FDDs and *BMDs [4]. BDDs are used to represent Boolean functions $f : \{0, 1\}^n \rightarrow \mathbb{Z}$.

A BDD is a rooted directed acyclic graph $G = (V, E)$ with non-empty node set $V$ containing two types of nodes, non-terminal and terminal nodes. A non-terminal node $v$ has as label a variable $\text{index}(v) \in \{x_1, \ldots, x_n\}$ and two children $\text{low}(v), \text{high}(v) \in V$. We call $\text{low}(v)$ also 0-successor($v$) and $\text{high}(v)$ 1-successor($v$). The edge leading to $\text{low}(v)$ ($\text{high}(v)$) is called low (high) edge of $v$. BDDs are ordered [2]. A terminal node $v$ is labeled with a value $\text{value}(v) \in \{0, 1\}$ and has no outgoing edges. The Boolean function $f_v : \{0, 1\}^n \rightarrow \{0, 1\}$ defined by a BDD node $v$ is defined recursively: If $v$ is a terminal node with $\text{value}(v) = c \in \{0, 1\}$, then $f_v(x_1, \ldots, x_n) = c$ and if $v$ is a non-terminal node with $\text{index}(v) = x_i$, then $f_v(x_1, \ldots, x_n) = x_i \cdot f_{\text{low}(v)}(x_1, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n) + x_i \cdot f_{\text{high}(v)}(x_1, \ldots, x_n)$. (BDDs use the so-called Shannon

1 see Section 3
In contrast to BDDs, BMDs are at a node $v$ reduction rules to reduce the reached by $v$ say that the terminal 0 is reached by $v$. However due to the Davio decomposition we have the –node is the terminal BMD size of this polynomial, which is an upper bound on the vector $\vec{x}$. Then the function $f_B(x_1, \ldots, x_n)$ is the following. Since $f$ is a function, $f_B(x_1, \ldots, x_n)$ is the function representing a function $f: \{0, 1\}^n \rightarrow \mathbb{Z}$ and let $v$ be the $(\epsilon_1, \ldots, \epsilon_t)$–node $(t \leq n)$. Then the function $f_B(x_1, \ldots, x_n)$ represented by $v$ is equal to

$$f_B(x_1, \ldots, x_n) = \sum_{(\delta_1, \ldots, \delta_t) \leq (\epsilon_1, \ldots, \epsilon_t)} (-1)^{\sum_{i=1}^{t} (\delta_i - \epsilon_i)} f_B(x_{\delta_1+1}, \ldots, x_{\delta_t+1}).$$

(For $\delta, \epsilon \in \{0, 1\}^t: \delta \leq \epsilon$ iff $\delta_1 \leq \epsilon_1, \ldots, \delta_t \leq \epsilon_t$.)

Lemma 2.1 shows that the change of the function $f_B$ for a single input vector $e$, i.e. the change of cofactor $f_B|_{x_e=\bar{e}}$, has not only a “local effect” in the Decision Diagram, but affects all $\gamma$–nodes with $e \leq \gamma$.

BMDs were defined in [4] to further reduce the size of BMDs by increasing the amount of subgraph sharing. In *BMDs each edge has an additional multiplicative edge weight $m \in \mathbb{Z}$, such that an edge with edge weight $m$ leading to a node $v$ represents a function $m \cdot f_B$. Reduction rules guarantee that functions $e_1 g$ and $e_2 g (e_1, e_2 \in \mathbb{Z} \setminus \{0\})$ are represented by the same node (but by different edges).

**3. DON'T CARE ASSIGNMENT**

In the following we present a solution to the problem to minimize a *BMD by assigning values to don’t cares. We have to solve the following problem DC*BMD: Given: A *BMD $B$ representing a function $f: \{0, 1\}^n \rightarrow \mathbb{Z}$ and a BDD $C$ representing a function $f': \{0, 1\}^n \rightarrow \mathbb{Z}$. Find: A *BMD $B'$ representing a function $f': \{0, 1\}^n \rightarrow \mathbb{Z}$, such that $f \cdot c = f' \cdot c'$ and $B'$ has the minimum number of nodes among all *BMDs fulfilling the same property (and respecting the same variable order).

DC*BMD is a hard problem, more precisely we can prove the following theorem [16]:

**Theorem 3.1** DC*BMD is NP complete.

This is why we are looking for a heuristic solution of DC*BMD in the following.

**3.1. Method min_polynomial**

Our first method min_polynomial is motivated by the relationship between BMDs over variables $x_1, \ldots, x_n$ and polynomials over $x_1, x_2, \ldots, x_n$. The rule to evaluate BMDs directly implies a method to derive the polynomial representing the same function as the BMD. E.g. the function from Figure 1 is equal to $(x_1 + x_2) \cdot (x_2 + x_3)$. In general the polynomial contains the term $c : x_1^2 \cdot x_2^2 \cdot \ldots \cdot x_n^2$ ($x_1^2 = x_1$ and $x_2^2 = 1$) if and only if the node reached by $(\epsilon_1, \ldots, \epsilon_n)$ is terminal $c \neq 0$.

It is easy to see that the size of the BMD $B$ representing function $f_B$ is always less or equal to the size of the polynomial, which can be obtained from BMDs by reduction, this is clearly also true for *BMDs.

Our first method consists in a (heuristic) minimization of the size of this polynomial, which is an upper bound on the BMD and the *BMD size. For vectors $(\epsilon_1, \ldots, \epsilon_n)$, such that the terminal reached by $(\epsilon_1, \ldots, \epsilon_n)$ is $c \neq 0$, we try to use don’t cares to change the value of the terminal to zero. If $(\epsilon_1, \ldots, \epsilon_n)$ is a don’t care vector, i.e. $dc(\epsilon_1, \ldots, \epsilon_n) = 1$, we change the function value $f_B(\epsilon_1, \ldots, \epsilon_n)$ such that the terminal reached by $(\epsilon_1, \ldots, \epsilon_n)$ will

$\n\text{Example 2.1 Figure 1(a) shows an example of a BMD for function}$

$\n\text{with } f(0, 0) = 1, f(0, 1) = 6, f(1, 0) = 5$ and $f(1, 1) = 10.$

$\n\text{The (0, 0)–node is terminal 1, the (0, 1)–node is terminal 5,}$

$\text{(a) reduced (b) non-reduced}$

$\text{Figure 1: Example for a BMD.}$

$\text{the (1, 0)–node is terminal 4, but the (1, 1)–node is terminal 0,}$

$\text{since the high edge starting from the root leads to a terminal and}$

$\text{not to a node with label } x_2 \text{ and -- as shown in Figure 1(b) -- in the}$

$\text{non–reduced BMD vector (1, 1) leads to terminal 0.}$

$\text{Using (1) we can conclude the following lemma by induction:}$

$\text{Lemma 2.1 Let } B \text{ be a BMD representing a function } f_B: \{0, 1\}^n \rightarrow \mathbb{Z}$

$\text{and let } v \text{ be the } (\epsilon_1, \ldots, \epsilon_t)–node \text{ (} t \leq n \text{). Then the function}$

$\text{that is represented by its root node}$

$\text{is}$

$\text{where the labels are}$

$\text{1 For a function, } f: \{0, 1\}^n \rightarrow \mathbb{Z}, f(x_{n+1}) = 0 (f(x_{n+1}) = 1)$

$\text{and is called negative (positive) cofactor of } f \text{ with respect to } x_i.$

The size of a polynomial is defined as the number of constants, variable names and operators + and \cdot in the polynomial.
be 0. Using the formula of Lemma 2.1 it is clear that we just have to set for the changed function \( f_B \) 
\[
f_B(x_1, \ldots, x_n) = f_B(x_1, \ldots, x_n - c)
\]
to achieve this goal. After that we must not forget to adjust the values of other terminals according to this change of \( f_B(x_1, \ldots, x_n) \), since the value of \( f_B(x_1, \ldots, x_n) \) has an impact on all terminals, which are reached by vectors \( \gamma \geq \gamma \).

The main idea of our method \( \text{min}_\text{polynomial} \) is illustrated in Figure 2. Figure 2 shows a BMD for the function \( f : \{0, 1\}^2 \rightarrow \mathbb{Z} \) with polynomial \( 1 + 4x_2 + 3x_1 + 8x_1x_2 \). There are two don't care vectors: \( dc(0, 0) = dc(1, 1) = 1 \). The don't care values for \( (0, 0) \) and \( (1, 1) \) are represented in the BMD by the shaded boxes of terminals 1 and 8. At first, we set terminal 1, which is reached by \( (0, 0) \) to 0. To achieve this we make use of the don't care vector \( (0, 0) \) and change \( f(0, 0) \) by adding \(-1\). Then we have to propagate the change to all terminals which are reached by vectors \( \gamma \geq (0, 0) \). According to the formula of Lemma 2.1 we have to change terminal 4 by adding 1, terminal 3 by adding 1 and terminal 8 by adding \(-1\). The resulting values for the terminals are given in Figure 2 in the row 1st dc below the original terminals. Finally we make use of the don't care \( (1, 1) \) by adding \(-7\) to \( f(1, 1) \) resulting in a 0-terminated reached by \( (1, 1) \). Since there is no vector greater than \( (1, 1) \), we do not have to propagate the change in this case and the resulting terminals are shown in the second row 2nd dc below the original terminals. Finally, we obtain a changed function with polynomial \( 5x_2 + 4x_1 \). The reduced version of the resulting BMD is shown on the right hand side of Figure 2.

The order of processing the different don't care values in the example was not arbitrary: Since we process the terminals from left to right the propagation of changes due to other don't care assignments can not destroy the zeros we have already set. For this reason our recursive procedure processes the *BMD in a depth-first manner following low ends before high edges. Pseudo code of the resulting recursive procedure \( \text{min}_\text{polynomial} \) to minimize a *BMD \( B \) using don't cares specified by a BDD \( DC \) is given in Figure 3 (we omit details like computed table etc.). Note that in line 8 the propagation of the changes made to \( B_{\text{low}} \) is performed by adding \( B_{\text{low}} - B_{\text{low}} \) to \( B_{\text{high}} \) before applying \( \min\text{polynomial} \) to \( B_{\text{high}} \).

### 3.2. Method independent_dfs

The second method is motivated by the “matching siblings” heuristics from [18]. This heuristics was introduced to minimize BDDs in a recursive procedure. When the procedure processes a BDD node \( v \), it tries to assign don't cares in such a way that \( \text{low}(v) \) and \( \text{high}(v) \) become identical. If this is possible, we have to keep this subgraph only once and additionally – because of the BDD reduction rules – node \( v \) can be removed, because the subfunction is now independent from variable \( \text{index}(v) \).

Since BMDs use positive Davio decomposition instead of Shannon decomposition, the function represented by a node \( v \) can not be made independent from variable \( \text{index}(v) \) by changing \( \text{low}(v) \) and \( \text{high}(v) \) to make them identical. Here we try to make use of don't cares to change \( \text{high}(v) \) so that it beco\( \text{min} \text{polynomial}(v) \) and we can delete \( \text{high}(v) \) and (according to BMD reduction rules) also node \( v \).

Thus, we have to check for a node \( v \), which is reached by \( (x_1, \ldots, x_n) \), whether the node function can be made independent from variable \( x_2 \) by selecting don't cares from \( B_{\text{low}} \) to make the root function independent from \( x_2 \), since it was already used in the minimization of this node, such that it becomes \( \text{min}_\text{polynomial}(v) \). This check is used in a depth-first traversal of the *BMD. Whenever we reach a node which can be made independent from its top variable, we perform the modification and the effect of the change is propagated similar to procedure \( \text{min}_\text{polynomial} \).

### 4. EXPERIMENTAL RESULTS

We implemented the two methods for *BMD minimization based on \( v \), i.e., an experimental Word-Level DD package developed at the University of Freiburg [10] and performed experiments to compare the different approaches. The experiments were performed using a SPARC UltraII with a memory limit of 400 MB.

To generate incompletely specified functions from completely specified functions, we used a method proposed in [5]. We collapse each benchmark circuit to two-level form (sum-of-products form). Each cube in this two-level form is contained in the on-set of at least one output function. Now we consider the set of...
all these cubes and randomly select cubes with a probability of 40% to be included into the don’t care set. For the resulting don’t care set a BDD is computed. Then a *BMD for an integer−valued function representing the benchmark circuit is computed. Here output $f_j (0 < j < m − 1)$ is weighted by $2^j$, such that the function value of this integer−valued function $f$ for input vector $e$ is $F(e) = \sum_{j=0}^{m-1} f_j(2^j \cdot e)$. As variable order we use the order given in the benchmark specification. The results are summarized in Table 1. In the first column the benchmark circuit is given, in the second column the number of primary inputs and in the third column the number of primary outputs. Column 4 shows the number of BDD nodes needed to represent the don’t care set and in column 5 the number of nodes needed to represent the initial *BMD. Columns 6−8 give the *BMD sizes after minimization. Three different methods are compared: For comparison we give in column $az$ the simple method to set all don’t care input vectors to function value 0, which can be done by computing $f_2 \cdot \overline{dC}$. Column $mp$ gives the results for our procedure $\text{min\_polynomial}$ and column $dfs$ the results for our procedure $\text{independent\_dfs}$. Columns 9−11 give the ratios “size of minimized *BMD divided by size of initial *BMD”, again for the three different methods. Finally the corresponding CPU times are given in columns 12−14 in format minutes:seconds, rounded to seconds.

The results show that setting all don’t cares to 0 (columns $az$) is not a successful method. On the average the sizes even increase by 24.7%. In contrast, our two methods for don’t care minimization are both very effective in minimizing the *BMD sizes: Method $\text{min\_polynomial}$ (columns $mp$) is able to reduce *BMD sizes by 74.6% on the average and method $\text{independent\_dfs}$ (columns $dfs$) reduces the sizes even by 70.2%. Columns 13 and 14 show that these results can be achieved within a small amount of run time.

### Table 1: Results for don’t care minimization

| Circuit | $\#PI$ | $\#PO$ | $|dC|C|$ | $|BMD|$ | $|\text{independent\_dfs}|$ | Ratio | $|BMD|$ | $|\text{independent\_dfs}|$ | $|dfs|$ | $|az|$ | $|mp|$ | $dfs$ | $Time$ |
|--------|--------|--------|--------|--------|----------------|--------|--------|----------------|--------|--------|--------|--------|--------|
| S5p1   | 7      | 10     | 25     | 76     | 19 | 12 | 1.18 | 19 | 12 | 1.18 | 19 | 12 | 1.18 |
| Synma4 | 9      | 1      | 7      | 22     | 242 | 183 | 1.26 | 242 | 183 | 1.26 | 242 | 183 | 1.26 |
| apex   | 19     | 17     | 123    | 1183   | 2.055 | 1183 | 1.26 | 2.055 | 1183 | 1.26 | 2.055 | 1183 | 1.26 |
| c8     | 28     | 18     | 1,26   | 586    | 3.333 | 1183 | 1.26 | 3.333 | 1183 | 1.26 | 3.333 | 1183 | 1.26 |
| pcleb  | 27     | 17     | 65     | 84     | 1.388 | 142 | 1.26 | 1.388 | 142 | 1.26 | 1.388 | 142 | 1.26 |
| f67    | 6      | 4      | 6      | 196    | 1.060 | 68 | 1.26 | 1.060 | 68 | 1.26 | 1.060 | 68 | 1.26 |
| sarz   | 10     | 4      | 22     | 128    | 0.720 | 68 | 1.26 | 0.720 | 68 | 1.26 | 0.720 | 68 | 1.26 |
| zaiml  | 8      | 4      | 22     | 128    | 0.720 | 68 | 1.26 | 0.720 | 68 | 1.26 | 0.720 | 68 | 1.26 |

5. CONCLUSIONS AND FUTURE WORK

We presented two heuristic methods for don’t care minimization of *BMDs. Experimental results proved them to be very effective in reducing *BMD sizes within a small amount of CPU time. At the moment we are working on a modified version of method $\text{independent\_dfs}$, which is based on the observation that in contrast to BDDS [18] for *BMDs the order in which we process the nodes can influence the quality of the result due to the propagation of the change. Setting the high son of a node $\nu$ to 0 can destroy the possibility to set the high son of another node $\psi$ to 0. Since the subgraph of the high son of a node at a higher level in the *BMD will be larger on the average, we expect that the gain of setting the high son of such a node to 0 is also larger. Therefore nodes at higher levels should processed first leading to a breadth-first traversal of the *BMD instead of a depth-first traversal.

Moreover, we are working on an application of our *BMD minimization in the verification of Pentium style integer dividers to keep peak memory consumption small during backward construction [9]. Don’t cares are computed by an iterative image computation for the different add$\&$shift stages.

### 6. REFERENCES