Preprint from Proceedings of International Symposium on Circuits and Systems, Sydney, Australia, May 2001

EXPLOITING DON'T CARES TO MINIMIZE *BMDS

Christoph Scholl, Marc Herbstritt and Bernd Becker

Institute of Computer Science, Albert–Ludwigs–University,
D 79110 Freiburg im Breisgau, Germany
<scholl/herbstri/becker>@informatik.uni-freiburg.de

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* *BMD *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 of a combinational circuit into a DD. Then, due to the canonicity of the DD representation, the equivalence check for specification and implementation reduces to the check whether the corresponding DDs are identical.

The most popular data structure in this context were *Binary Decision Diagrams* (BDDs) [2]. They were applied successfully e.g. to the verification of control logic and integer adders. But there are functions of high practical relevance (e.g. integer multipliers), which can not be represented efficiently by BDDs. To overcome the limitations of BDDs other types of DDs were defined, e.g. *Binary Moment Diagrams* (BMDs) and *Multiplicative* BMDs (*BMDs) [4], which are able to represent integer–valued pseudo Boolean functions $f : \{0,1\}^n \rightarrow \mathbb{Z}$ and which are especially suited for arithmetic functions.

When a circuit consists of several modules or subcircuits, existing methods to compute the *BMD representing the overall circuit compute *BMDs for the modules and combine these *BMDs to a *BMD for the overall circuit by *BMD operations [6]. Other methods use backward construction [9, 13] from the circuit outputs towards the inputs and compose step by step the *BMD for a gate of the current cut front into the *BMD for the intermediate result.

A potential, which has not been used in this process so far, is the knowledge that certain input combinations can not be applied to subcircuits/modules. Input combinations, which can not be applied to subcircuits, can be given as don't care informations in the circuit specification or can be computed as satisfiability don't cares by image computations [1]. These don't cares can be used to minimize *BMDs – either before combining the *BMDs for submodules by *BMD operations or in the backward construction method when the processing of a submodule, for which don't care informations are at hand, is finished. In this context the minimization of *BMDs by exploiting don't care informations aims at reducing the *BMD sizes to keep peak memory consumption as low as possible.

The problem we have to solve is to minimize a *BMD B for a function f_B under don't care conditions given by a characteristic function dc ($dc(x) = 1$, if x is a don't care vector, i.e. x can not be applied to the subcircuit realizing f_B). Since dc is a Boolean function, we assume that it is represented by a BDD. Our task is to compute a *BMD B' realizing a function $f_{B'}$, such that $f_B(x) = f_{B'}(x)$ for all x with $dc(x) = 0$ and B' has a (nearly) minimum number of nodes among all *BMDs fulfilling this property.

To the best of our knowledge the heuristics presented in this paper are the first solution to this problem. For the minimization of BDDs 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.

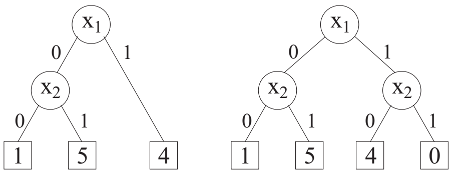
We developed two different methods for the minimization of *BMDs under don't care conditions. After Section 2, which gives some basic definitions and notations, we present these methods in Section 3 and in Section 4 we give preliminary experimental results to evaluate the approaches. The minimization 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], BMDs and *BMDs [4]. BDDs are used to represent Boolean functions $f : \{0,1\}^n \rightarrow \{0,1\}$, and both BMDs and *BMDs represent integer–valued pseudo 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 $index(v) \in \{x_1, \ldots, x_n\}$ and two children $low(v), high(v) \in V$. We call $low(v)$ also 0–*successor*(v) and $high(v)$ 1–*successor*(v). The edge leading to $low(v)$ ($high(v)$) is called low (high) edge of v. BDDs are ordered [2]. A terminal node v is labeled with a value $value(v) \in \{0,1\}$ 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 $value(v) = c \in \{0,1\}$, then $f_v(x_1, \ldots, x_n) = c$ and if v is a non-terminal node with $index(v) = x_i$, then $f_v(x_1, \ldots, x_n) = \overline{x_i} \cdot f_{low(v)}(x_1, \ldots, x_n) + x_i \cdot f_{high(v)}(x_1, \ldots, x_n)$. (BDDs use the so-called Shannon

[1] see Section 3



(a) reduced (b) non-reduced

Figure 1: Example for a BMD.

decomposition.) The function represented by a BDD B is equal to the function represented by its root node v_{root}.

Like BDDs BMDs are based on a rooted directed acyclic graph. In contrast to BDDs the terminal nodes v are labeled with values $value(v) \in \mathbb{Z}$. The recursive definition of the pseudo Boolean function $f_v : \{0,1\}^n \to \mathbb{Z}$ represented by a BMD node v differs from BDDs: If v is a terminal node with $value(v) = c \in \mathbb{Z}$, then $f_v(x_1,\dots,x_n) = c$ and if v is a non-terminal node with $index(v) = x_i$, then $f_v(x_1,\dots,x_n) = f_{low(v)}(x_1,\dots,x_n) + x_i \cdot f_{high(v)}(x_1,\dots,x_n)$. BMDs use the so-called *positive Davio decomposition*. It follows from this recursive definition that the function represented by $low(v)$ is equal to $f_v|_{x_i=0}$[†], but in contrast to Shannon decomposition the function represented by $high(v)$ is

$$f_v|_{x_i=1} - f_v|_{x_i=0}. \tag{1}$$

Since BMDs use another decomposition type than BDDs (positive Davio decomposition instead of Shannon decomposition), the reduction rules to reduce the BMD sizes and to make BMDs a canonical data structure have to be changed compared to BDDs: As in the case of BDDs, if for terminal nodes v and $v' \in V$ $value(v) = value(v')$ or if for non-terminal nodes v and v' $index(v) = index(v')$, $low(v) = low(v')$ and $high(v) = high(v')$ then $v = v'$. However due to the Davio decomposition we have the reduction rule that in a reduced BMD there is no node $v \in V$ with $high(v) = t$, t terminal node with $value(t) = 0$.

For simplicity we assume in the following that the variables occur in the fixed order $x_1,\dots,x_n$.

To give a relation between nodes of a BMD B and cofactors of the function f_B represented by B, we define "the node which is reached by $(\epsilon_1,\dots,\epsilon_l) \in \{0,1\}^l$ $(l \le n)$":

To determine the node reached by $(\epsilon_1,\dots,\epsilon_l)$ we start at the root node and follow the edges according to $(\epsilon_1,\dots,\epsilon_l)$. If we are at a node v labeled with x_i and $\epsilon_i = 0$, then we follow the edge to $low(v)$ and if $\epsilon_i = 1$, we go to $high(v)$. Special attention has to be paid to the case, when ϵ_i*–successor*(v) has not label x_{i+1}. If in this case ϵ_i*–successor*(v) is a non–terminal, choose k with $x_k = index(\epsilon_i\text{–}successor(v))$ and if ϵ_i*–successor*(v) is a terminal choose $k = n+1$. Then we have to take into account, that in an non–reduced version of the BMD the edge leading to ϵ_i*–successor*(v) would be replaced by a path of nodes leading to ϵ_i*–successor*(v) where the labels are $x_{i+1},\dots,x_{k-1}$ and the high edges lead to the constant 0, respectively. Therefore we go to ϵ_i*–successor*(v) only if $\epsilon_{i+1} = \dots = \epsilon_{k-1} = 0$, otherwise we say that the terminal 0 is reached by $(\epsilon_1,\dots,\epsilon_l)$ (since 0 would be reached in a non–reduced version of the BMD). We call the node reached by $(\epsilon_1,\dots,\epsilon_l)$ also $(\epsilon_1,\dots,\epsilon_l)$–node and the function represented by this node $f_B^{(\epsilon_1,\dots,\epsilon_l)}$.

Example 2.1 *Figure 1(a) shows an example of a* BMD *for function f with $f(0,0) = 1$, $f(0,1) = 6$, $f(1,0) = 5$ and $f(1,1) = 10$. The $(0,0)$–node is the terminal 1, the $(0,1)$–node is terminal 5, the $(1,0)$–node is terminal 4, but the $(1,1)$–node is terminal 0, since the high edge starting from the root leads to a terminal and not to a node with label x_2 and – as shown in Figure 1(b) – in the non–reduced* BMD *vector $(1,1)$ leads to terminal 0.*

Using (1) we can conclude the following lemma by induction:

Lemma 2.1 *Let B be a* BMD *representing a function $f_B : \{0,1\}^n \to \mathbb{Z}$ and let v be the $(\epsilon_1,\dots,\epsilon_l)$–node ($l \le n$). Then the function $f_B^{(\epsilon_1,\dots,\epsilon_l)}$ represented by v is equal to*

$$f_B^{(\epsilon_1,\dots,\epsilon_l)} = \sum_{\substack{(\delta_1,\dots,\delta_l)\le \\ (\epsilon_1,\dots,\epsilon_l)}} (-1)^{\sum_{i=1}^{l}(\epsilon_i-\delta_i)} f_B|_{x_1=\delta_1,\dots,x_l=\delta_l}. \tag{2}$$

(For $\delta,\epsilon \in \{0,1\}^l : \delta \le \epsilon$ iff $\delta_i \le \epsilon_i\ \forall 1 \le i \le l$.)

Lemma 2.1 shows that the change of the function f_B for a single input vector ϵ, i.e. the change of cofactor $f_B|_{x=\epsilon}$, has not only a "local effect" in the Decision Diagram, but affects all γ–nodes with $\epsilon \le \gamma$.

*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 $m \cdot f_v$. Reduction rules guarantee that functions $c_1 \cdot g$ and $c_2 \cdot g$ ($c_1, c_2 \in \mathbb{Z} \setminus \{0\}$) 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 $f : \{0,1\}^n \to \mathbb{Z}$ and a BDD C representing a function $c : \{0,1\}^n \to \{0,1\}$.

Find: A *BMD B' representing a function $f' : \{0,1\}^n \to \mathbb{Z}$, 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.*

That 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 $x_1,\dots,x_n$ and polynomials over $x_1,\dots,x_n$: 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 $(1 + x_2 \cdot 5) + x_1 \cdot 4 = 1 + 5x_2 + 4x_1$. In general the polynomial contains the term $c \cdot x_1^{\epsilon_1} \cdot \ldots \cdot x_n^{\epsilon_n}$ ($x_i^1 = x_1$ and $x_i^0 = 1$) if and only if the node reached by $(\epsilon_1,\dots,\epsilon_n)$ is terminal $c \ne 0$.

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[3] representing f_B. Since *BMDs 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 $(\epsilon_1,\dots,\epsilon_n)$, such that the terminal reached by $(\epsilon_1,\dots,\epsilon_n)$ is $c \ne 0$, we try to use don't cares to change the value of the terminal to zero. If $(\epsilon_1,\dots,\epsilon_n)$ is a don't care vector, i.e. $dc(\epsilon_1,\dots,\epsilon_n) = 1$, we change the function value $f_B(\epsilon_1,\dots,\epsilon_n)$ such that the terminal reached by $(\epsilon_1,\dots,\epsilon_n)$ will

[†]For a function, $f : \{0,1\}^n \to \mathbb{Z}$ $f_{x_i=0}$ ($f_{x_i=1}$) is the function which results from a substitution of x_i by constant 0 (1) and is called negative (positive) cofactor of f with respect to x_i.

[3]The size of a polynomial is defined as the number of constants, variable names and operators + and · in the polynomial.

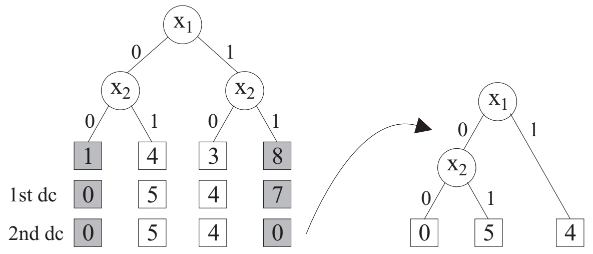


Figure 2: Example: BMD minimization by *min_polynomial*.

```
1  *BMD function min_polynomial(*BMD $B$, BDD $DC$)
2  if $DC = \boxed{1}$ then return $\boxed{0}$ fi; if $DC = \boxed{0}$ then return $B$ fi
3  if $B = constant$ then return $B$ fi
4  Let $v$ be top variable of $B$ and $DC$,
5  $B_{low} = B|_{v=0}, B_{high} = B|_{v=1} - B|_{v=0}$,
6  $DC_{low} = DC|_{v=0}, DC_{high} = DC|_{v=1}$
7  $B'_{low} := min\_polynomial(B_{low}, DC_{low})$
8  $B'_{high} := min\_polynomial(B_{high} + (B_{low} - B'_{low}), DC_{high})$
9  $B' = B'_{low} + v \cdot B'_{high}$
10 if $size(B') < size(B)$ return $B'$ else return $B$ fi
```

Figure 3: Pseudo code for *min_polynomial*.

be 0. Using the formula of Lemma 2.1 it is clear that we just have to set for the changed function $f_{B'}$

$$f_{B'}(\epsilon_1, \ldots, \epsilon_n) = f_B(\epsilon_1, \ldots, \epsilon_n) - c$$

to achieve this goal. After that we must not forget to adjust the values of other terminals according to this change of $f_B(\epsilon_1, \ldots, \epsilon_n)$, since the value of $f_B(\epsilon_1, \ldots, \epsilon_n)$ has an impact on all terminals, which are reached by vectors $\gamma \geq \epsilon$.

The main idea of our method *min_polynomial* is illustrated in Figure 2. Figure 2 shows a BMD for the function $f : \{0,1\}^2 \to \mathbb{Z}$ 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 $> (0,0)$. 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–terminal 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 edges before high edges. Pseudo code of the resulting recursive procedure *min_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_{low} is performed by adding $B_{low} - B'_{low}$ to B_{high} before applying *min_polynomial* to B_{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 $low(v)$ and $high(v)$ 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 $index(v)$.

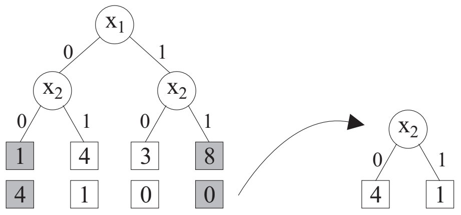


Figure 4: Example: BMD minimization by *independent_dfs*.

Since BMDs use positive Davio decomposition instead of Shannon decomposition, the function represented by a node v can not be made independent from variable $index(v)$ by changing $low(v)$ and $high(v)$ to make them identical. Here we try to make use of don't cares to change $high(v)$, such that it becomes 0. Then, the function represented by v is independent from $index(v)$ and we can delete $high(v)$ and (according to BMD reduction rules) also node v.

Thus, we have to check for a node v, which is reached by $(\epsilon_1, \ldots, \epsilon_l)$, whether the node function can be made independent from variable x_{l+1} by exploiting don't cares from $dc|_{x_1=\epsilon_1,\ldots,x_l=\epsilon_l}$. Figure 4 illustrates the method using the same example as in Figure 2. At the beginning we check whether the root node v can be made independent from x_1 by using don't cares, which is equivalent to the question, if we can set $high(v)$ to zero. To do this we can exploit don't cares both from $dc|_{x_1=0}$ and from $dc|_{x_1=1}$, i.e. both the don't cares at $(0,0)$ and $(1,1)$ in this example. The terminal reached by $(1,0)$ can not be set to 0 using don't cares from $dc|_{x_1=1}$, but it is possible to use don't care $(0,0)$ (adding 3 to $f(0,0)$) to set this terminal to 0. Then we use don't care $(1,1)$ to set the terminal reached by $(1,1)$ to 0 and in fact, it is possible to make the root node independent from x_1. The changed values for the terminals are given in Figure 4 in the row below the original terminals. The reduced BMD is given on the right hand side of Figure 4. It is easy to see that it is not possible to make the remaining node independent from x_2, since there are no don't cares which could be exploited. (Note that also the don't care $(0,0)$ must not be used in the minimization of this node, since it was already used to make the root function independent from x_1. Exploitation of don't care $(0,0)$ could make the function depend on x_1 again.)
The check, whether a function of a node v, which is reached by $(\epsilon_1, \ldots, \epsilon_l)$, can be made independent from variable x_{l+1} using $dc|_{x_1=\epsilon_1,\ldots,x_l=\epsilon_l}$ can be formulated as a recursive procedure, which checks first if the low son can be set to 0 and then if the high son can be set to 0; details can be found in [16]. 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 *min_polynomial*.

4. EXPERIMENTAL RESULTS

We implemented the two methods for *BMD minimization based on `wld`, an experimental Word-Level DD package developed at 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 onset of at least one output function. Now we consider the set of

Circuit	#PI	#PO	$\|DC\|$	$\|{*}\mathrm{BMD}\|$	$\|{*}\mathrm{BMD}_{min}\|$			ratio $\frac{\|{*}\mathrm{BMD}_{min}\|}{\|{*}\mathrm{BMD}\|}$			Time		
					az	mp	dfs	az	mp	dfs	az	mp	dfs
5xp1	7	10	15	76	19	12	3	0.250	0.157	0.039	0:00	0:00	0:00
9symml	9	1	97	223	242	183	182	1.085	0.820	0.816	0:09	0:00	0:00
alu2	10	6	91	401	372	139	147	0.927	0.346	0.366	0:30	0:01	0:01
apex7	49	37	120	1390	2305	118	49	1.658	0.084	0.035	0:08	1:28	3:27
c8	28	18	126	346	336	17	13	0.971	0.049	0.037	0:02	1:23	0:02
mux	21	1	5798	60	47	34	34	0.783	0.566	0.566	0:00	0:06	0:18
pcler8	27	17	34	44	61	32	21	1.386	0.727	0.477	0:01	0:00	0:09
rd73	7	3	36	89	87	43	36	0.977	0.483	0.404	0:02	0:00	0:00
rd84	8	4	65	196	200	114	81	1.020	0.581	0.413	0:15	0:00	0:00
sao2	10	4	52	128	96	47	37	0.750	0.367	0.289	0:01	0:00	0:00
z4ml	7	4	30	69	87	30	26	1.260	0.434	0.376	0:00	0:00	0:00
			$\sum$	3022	3852	769	629	1.247	0.254	0.208			

Table 1: Results for don't care minimization.

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_i ($0 \leq i \leq m-1$) is weighted by 2^i, such that the function value of this integer–valued function f for input vector ϵ is $f(\epsilon) = \sum_{i=0}^{m-1} 2^i \cdot f_i(\epsilon)$. As variable order we used the initial 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 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 $f_B \cdot \overline{dc}$. Column *mp* gives the results for our procedure *min_polynomial* and column *dfs* the results for our procedure *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 *min_polynomial* (columns *mp*) is able to reduce *BMD sizes by 74.6% on the average and method *independent_dfs* (columns *dfs*) reduces the sizes even by 79.2%. Columns 13 and 14 show that these results can be achieved within a small amount of run time.

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 *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 v to 0 can destroy the possibility to set the high son of another node v' 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

[1] K. Bartlett, R. K. Brayton, G. Hachtel, R. M. Jacoby, C. R. Morrison, R. Rudell, A. L. Sangiovanni-Vincentelli, and A. R. Wang. Multilevel logic minimization using implicit don't cares. *IEEE Trans. on CAD*, 7(6):723–740, 1988.

[2] R.E. Bryant. Graph - based algorithms for Boolean function manipulation. *IEEE Trans. on Comp.*, 35(8):677–691, 1986.

[3] R.E. Bryant. Binary decision diagrams and beyond: Enabeling techniques for formal verification. In *Int'l Conf. on CAD*, pages 236–243, 1995.

[4] R.E. Bryant and Y.-A. Chen. Verification of arithmetic functions with binary moment diagrams. In *Design Automation Conf.*, pages 535–541, 1995.

[5] S. Chang, D. Cheng, and M. Marek-Sadowska. Minimizing ROBDD size of incompletely specified multiple output functions. In *European Design & Test Conf.*, pages 620–624, 1994.

[6] Y.-A. Chen and R.E. Bryant. ACV: an arithmetic circuit verifier. In *Int'l Conf. on CAD*, pages 361–365, 1996.

[7] O. Coudert, C. Berthet, and J.C. Madre. Verification of sequential machines based on symbolic execution. In *Automatic Verification Methods for Finite State Systems, LNCS 407*, pages 365–373, 1989.

[8] O. Coudert, C. Berthet, and J.C. Madre. Verification of sequential machines using Boolean functional vectors. In *Proceedings IFIP International Workshop on Applied Formal Methods for Correct VLSI Design*, pages 111–128, 1989.

[9] K. Hamaguchi, A. Morita, and S. Yajima. Efficient construction of binary moment diagrams for verifying arithmetic circuits. In *Int'l Conf. on CAD*, pages 78–82, 1995.

[10] M. Herbstritt. Erfüllbarkeitsprobleme bei Word-Level Decision Diagrams. Master's thesis, University Freiburg, April 2000.

[11] Y. Hong, P.A. Beerel, J.R. Burch, and K.L. McMillan. Safe BDD minimization using don't cares. In *Design Automation Conf.*, pages 208–213, 1997.

[12] U. Kebschull, E. Schubert, and W. Rosenstiel. Multilevel logic synthesis based on functional decision diagrams. In *European Conf. on Design Automation*, pages 43–47, 1992.

[13] M. Keim, M. Martin, B. Becker, R. Drechsler, and P. Molitor. Polynomial formal verification of multipliers. In *VLSI Test Symp.*, pages 150–155, 1997.

[14] A. Kuehlmann and F. Krohm. Equivalence checking using cuts and heaps. In *Design Automation Conf.*, pages 263–268, 1997.

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

[16] C. Scholl, M. Herbstritt, and B. Becker. Exploiting don't cares to minimize *BMDs. Technical report, Albert-Ludwigs-University, Freiburg, September 2000.

[17] C. Scholl, S. Melchior, G. Hotz, and P. Molitor. Minimizing ROBDD sizes of incompletely specified functions by exploiting strong symmetries. In *European Design & Test Conf.*, pages 229–234, 1997.

[18] T.R. Shiple, R. Hojati, A.L. Sangiovanni-Vincentelli, and R.K. Brayton. Heuristic minimization of BDDs using don't cares. In *Design Automation Conf.*, pages 225–231, 1994.

[19] D. Varma and E.A. Trachtenberg. Computation of reed–muller expansions of incompletely specified boolean functions from reduced representations. *IEE Proceedings*, 138(2):85–92, 1991.

[20] Z. Zilic and K. Radecka. Don't care FDD minimization by interpolation. In *Int'l Workshop on Logic Synth.*, pages 353–356, 1998.