IEEE International Conference on Computer Design (ICCD'00)
Austin, September 2000

Efficient Dynamic Minimization of Word-Level DDs based on Lower Bound Computation

Wolfgang Günther Rolf Drechsler

Institute of Computer Science
Albert-Ludwigs-University
Freiburg im Breisgau, Germany
{guenther,drechsle}@informatik.uni-freiburg.de

Stefan Höreth

Siemens AG
Corporate Technology
Munich, Germany
Stefan.Hoereth@mchp.siemens.de

Abstract

Word-Level Decision Diagrams (WLDDs), like ∗BMDs or K∗BMDs, have been introduced to overcome the limitations of Binary Decision Diagrams (BDDs), which are the state-of-the-art data structure to represent and manipulate Boolean functions. However, the size of these graph types largely depends on the variable ordering, i.e. it may vary from linear to exponential. In the meantime, dynamic approaches to find a good variable ordering are also known for WLDDs.

In this paper we show how these approaches can be accelerated significantly using a combination of a lower bound computation and synthesis operations. In the experiments it turned out that by this technique, the runtime for dynamic minimization can be reduced by more than 40% on average without loss of quality.

1 Introduction

Formal verification has become one of the most important steps in circuit design. Since modern circuit designs can contain several million transistors, also verification of such large designs becomes more and more difficult. Pure simulation cannot guarantee the correct behavior and exhaustive simulation is often impossible.

However, many designs have a very regular structure, like ALUs, that can be described easily on a higher level of abstraction. For example, describing and verifying an integer multiplier on the bit-level is very difficult, while the verification becomes easy if the outputs are grouped to a bit-string [9]. Recently, several approaches to formal circuit verification have been proposed that make use of these regularities [1, 15, 3]. All these approaches have in common that they are based on *Word-Level Decision Diagrams* (WLDDs), i.e. a graph based representation of functions (similar to BDDs [8]) that allow to represent functions with a Boolean range and an integer domain. Examples of WLDDs are EVBDDs [25], MTBDDs [12, 2], *BMDs [9], HDDs [13], and K*BMDs [18]. In the meantime WLDDs have been integrated in verifications tools [1, 11] and are also used for symbolic model checking [14, 10, 16].

BDDs and WLDDs have in common that the underlying variable ordering determines the size of the diagram and by this the memory needed to represent a function. Therefore, finding a good variable ordering is very important for WLDDs. For BDDs, the best results were obtained by *sifting* [27], measured in the number of nodes of the resulting BDD. Sifting can be used for WLDDs, too, and it can also be extended to improve the *Decomposition Type List* (DTL) of WLDDs simultaneously [23]. (This is also known as DTL-sifting [17].) However, especially for larger functions, dynamic minimization can become very time consuming. In [24], it is reported that for automatic verification of a BCD-to-binary converter 60% of the runtime was used for dynamic minimization. Therefore, methods to speed up sifting are increasingly important.

In this paper we investigate the use of lower bound techniques during dynamic minimization of WLDDs. We prove for the first time lower bounds for WLDDs. Similar lower bounds have already successfully been used for BDDs [19]. Furthermore, we show why dynamically applying the approach from [19] in the word-level domain cannot reduce the runtime much and how these problems can be solved. Finally, we give experimental results to show the efficiency of our approach. It is shown that runtime for both sifting and DTL-sifting can be reduced by more than 40% using these methods.

The paper is structured as follows: first, the concept of WLDDs is reviewed and dynamic variable reordering techniques are described. In Section 3, the proof for the lower bounds is given. A method to incorporate these lower bounds to DTL-sifting is outlined in Section 4. Experimental results are given in Section 5. Finally, the paper is summarized.

2 Preliminaries

2.1 Word-Level Decision Diagrams (WLDDs)

Decision Diagrams (DDs) are a graph-based representation for functions $f : \mathbf{B}^n \rightarrow \mathbf{B}$ (bit-level DDs) or $f : \mathbf{B}^n \rightarrow \mathbf{Z}$ (word-level DDs, WLDDs). At each (non-terminal) node v labeled with a variable x, a decomposition of function f represented by v into two sub-functions $f_{(x,low)}$ and $f_{(x,high)}$ is performed.

For bit-level DDs the following three decompositions have been considered [17]:

$$\begin{array}{rcll} f & = & \overline{x}\, f_{(x,low)} \oplus x\, f_{(x,high)} & \text{Shannon } (S) \\ f & = & f_{(x,low)} \oplus x\, f_{(x,high)} & \text{positive Davio } (pD) \\ f & = & f_{(x,low)} \oplus \overline{x}\, f_{(x,high)} & \text{negative Davio } (nD) \end{array}$$

Function f is represented at node v, while $f_{(x,low)}$ ($f_{(x,high)}$) denotes the function represented by the *low*-edge (*high*-edge) of v. $\oplus$ is the Boolean Exclusive OR operation. The recursion stops at terminal nodes labeled with 0 or 1.

If a decomposition of type S (pD, nD) is carried out at some node v, then v is called a S-node (a pD, nD-node). If only decompositions of type S are applied the resulting DD is a BDD [8], while in OKFDDs [20] all three are allowed.

In this paper, we consider the same three decompositions for WLDDs:

$$\begin{array}{rcl} f & = & (1-x)\cdot f_{(x,low)} + \quad x \cdot f_{(x,high)} \\ f & = & f_{(x,low)} + \quad x \cdot f_{(x,high)} \\ f & = & f_{(x,low)} + (1-x)\cdot f_{(x,high)} \end{array}$$

The notation S, pD, and nD is used analogously to the bit-level. x still denotes a Boolean variable, but the values of the functions are integer numbers and they are combined with the usual operations (addition, subtraction, and multiplication) in the ring **Z** of integers.

The functions $f_{(x,low)}$ and $f_{(x,high)}$ can be computed as follows ($f_{x=c}$ denotes the *cofactor* of f to $x = c$):

$$\begin{array}{llll} f_{(x,low)} = f_{x=0} & f_{(x,high)} = f_{x=1} & S \\ f_{(x,low)} = f_{x=0} & f_{(x,high)} = f_{x=1} - f_{x=0} & pD \\ f_{(x,low)} = f_{x=1} & f_{(x,high)} = f_{x=0} - f_{x=1} & nD \end{array}$$

For two variables x, y and $a, b \in \{low, high\}$, we define

$$f_{(x,a),(y,b)} := (f_{(x,a)})_{(y,b)}.$$

Decomposition types are associated with the n Boolean variables $x_1, x_2, \ldots, x_n$ using a *Decomposition Type List* (DTL) $d := (d_1, \ldots, d_n)$ where $d_i \in \{S, pD, nD\}$, i.e. d_i provides the decomposition type for variable x_i ($i = 1, \ldots, n$).

While it turned out that *complement edges* are a good extension to BDDs [7], for WLDDs also additive and multiplicative edge values were introduced. For K$*$BMDs which are the most general WLDD the function represented by a node v and edge values a and m is obtained by

$$f = a + m f_v,$$

where a and m are integer numbers and f_v is the function represented by node v.

Of course, the size of the representation for a given function f depends on the chosen variable ordering and DTL. There are examples of parameterized functions which provide exponential gaps between representations with differing variable order and DTL [8, 4]. On the other hand, finding good variable orderings and DTLs turns out to be difficult [6] and therefore heuristics have been developed to find at least reasonable orderings and DTLs. The most successful approaches are based on *Dynamic Variable Ordering* (DVO) [27, 23] and DVO with DTL change [17].

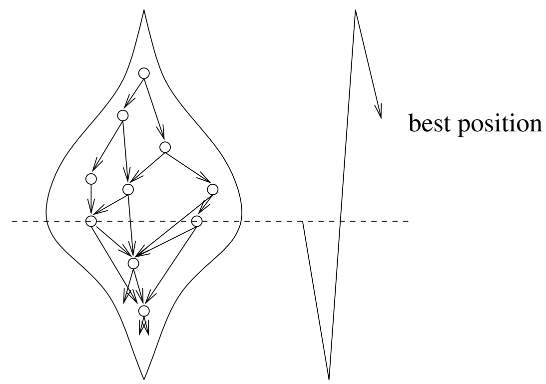

Figure 1. Sifting one variable

2.2 Exchange of Neighboring Variables

Most DVO methods are based on exchanging adjacent variables of the variable ordering [21, 27]. If the exchange is a local operation, it can be performed very quickly since only edges must be redirected within both levels, and a complete reconstruction of the DD is not necessary. For BDDs it has been shown that level exchange is a local operation [21]. For WLDDs, the level exchange is not in all cases a local operation, but several restrictions / extensions allow to handle it efficiently [23]. In case of K$*$BMDs, it has been shown that one repair run after sifting is sufficient to restore canonicity, as only the sign of the multiplicative edge values may be affected. For more details see [23].

2.3 Sifting

The most popular DVO algorithm is *sifting* [27]. All variables of a given DD are successively considered. When a variable is chosen, the goal is to find the best local position of the variable, assuming that the relative order of all other variables remains the same. In a first step, the order in which the variables are considered is determined. This is done by sorting the levels according to their size with largest level first. To find the best position, the variable is moved across the whole DD. In [27], this is done in three steps (see Figure 1):

1. The variable is exchanged with its successor variable until it is the last variable.
2. The variable is exchanged with its predecessor until it is the topmost variable.
3. The variable is moved back to the closest position which led to the minimal size of the DD.

2.4 DTL-Sifting

The idea of DTL-sifting is to alter the DTL if it is easy to do so, i.e. when the variable is in the lowest level of the DD. For any node of the lowest level, both outgoing

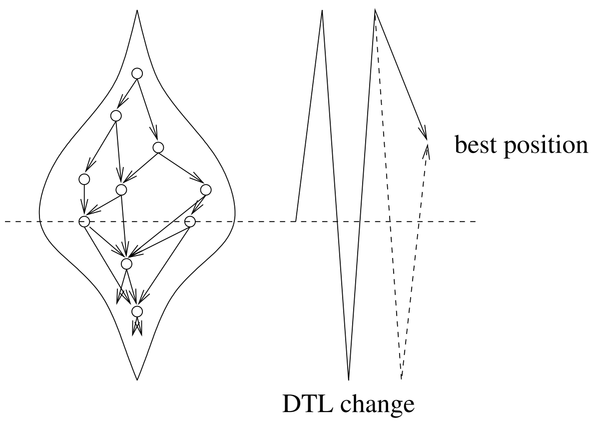

Figure 2. DTL-sifting for one variable

edges are constants and thus the decomposition type can easily be modified. An additional repair run is necessary only if a transformation to (or from) nD is performed [17]. Therefore, nD has not been considered in [23] as a decomposition type, i.e. only S and pD are used. The following procedure has been proposed (see Figure 2):

1. The variable is exchanged with its predecessor variable until it is the topmost variable.
2. The variable is exchanged with its successor until it is the last variable.
3. The decomposition type of the variable is modified, i.e. if the decomposition type was S, it is set to pD, and vice versa.
4. The variable is exchanged with its predecessor until it is the topmost variable.
5. The variable is moved back to the closest position which led to the minimal size of the DD. If the decomposition type before has led to a better size, then the type is modified again, i.e. the variable is again sifted down, the type is exchanged, and finally the variable is moved to the optimal position.

3 Lower Bounds on WLDD Sizes

The idea of using lower bounds during sifting is that the process of moving a variable into some direction can be stopped as soon as it is known that no further improvement is possible. If the computation of the lower bound is less expensive than actually moving the variable, then the overall computation time can be reduced without loss of quality.

In the approach proposed in this paper a lower bound that can be computed efficiently is used. It specifies how much the size can be reduced at most if a given variable is moved in the variable order. In fact this is the same lower bound that is already known as a theoretical result for BDDs (see [5]) and that was already used to accelerate sifting of BDDs [19]. However, it is not obvious at all that this lower bound transfers to WLDDs.

3.1 Bounds for Level Exchanges

In the following, the node increase for the exchange of neighboring levels is examined. In the whole section it is assumed that the function essentially depends on all its variables and that the variable ordering before the exchange operation is $(x_1, x_2, \ldots, x_n)$. The number of nodes labeled with x_i is denoted by $\text{label}(x_i)$. Furthermore, we restrict to the case of K$*$BMDs since this is the most general data structure.

Theorem 1 Let $f : \mathbf{B}^n \rightarrow \mathbf{Z}$ be represented by a K$*$BMD and let $1 \leq i < n$. If the levels i and $i+1$ are exchanged, the resulting level sizes $\text{label}'(\cdot)$ are bounded by

$$1 \leq \text{label}'(x_i) \leq 2 \cdot \text{label}(x_i)$$

and

$$\tfrac{1}{2} \cdot \text{label}(x_{i+1}) \leq \text{label}'(x_{i+1}) \leq \text{label}(x_i) + \text{label}(x_{i+1}).$$

All other levels remain unchanged.

Proof: Since a level exchange is a local operation up to the sign of the multiplicative weight [23], the level sizes of all other levels remain unchanged. To prove the inequations, we consider four cases:

Case 1. $1 \leq \text{label}'(x_i)$,
Case 2. $\text{label}'(x_i) \leq 2 \cdot \text{label}(x_i)$,
Case 3. $\frac{1}{2} \cdot \text{label}(x_{i+1}) \leq \text{label}'(x_{i+1})$,
Case 4. $\text{label}'(x_{i+1}) \leq \text{label}(x_i) + \text{label}(x_{i+1})$.

Cases 1 and 4 directly transfer from BDDs to WLDDs [5]. Case 3 follows from Case 2 since exchanging two levels twice leads to the original representation. To prove Case 2, more detailed observations have to be made.

Level i has to represent the following set of functions:

$$C = \{c \mid \exists e_1, \ldots, e_{i-1} \in \{low, high\} : \\ c = f_{(x_1,e_1),\ldots,(x_{i-1},e_{i-1})} \text{ and } c \text{ depends on } x_i\}.$$

It is represented by a set of nodes V, such that

$$\forall c \in C\ \exists v \in V\ \exists m, a \in \mathbf{Z} : c = mv + a.$$

If the levels i and $i+1$ are exchanged, the set

$$C' = \{c' \mid \exists e_1, \ldots, e_{n-1}, e \in \{low, high\} : \\ c' = f_{(x_1,e_1),\ldots,(x_{i-1},e_{i-1}),(x_{i+1},e)} \\ \text{and } c' \text{ depends on } x_i\}.$$

has to be represented in level $i+1$ by a set of nodes V'. The size of C' is at most twice the size of C. V' is given by

$$V' = \{v_{(x_{i+1},low)} \mid v \in V\} \cup \{v_{(x_{i+1},high)} \mid v \in V\},$$

i.e. also the size of V can double at most (see Figure 3).

Now it has to be shown that all nodes of C' can really be represented by V'. Let the decomposition type of x_{i+1} be pD (all other cases are handled analogously). Let $c' \in C'$. Assume that c' depends essentially on x_{i+1} (otherwise the node is not affected by the level exchange). Then $c' = c_{(x_{i+1},e)}$ for some $c \in C, e \in \{high, low\}$,

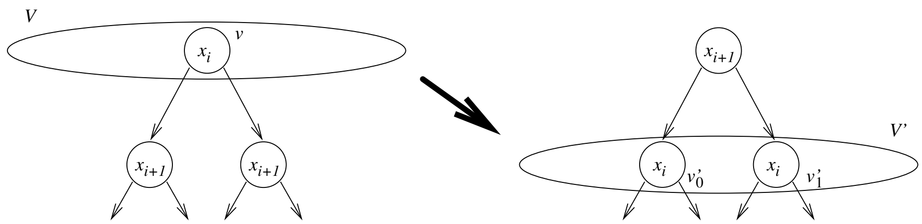

Figure 3. Nodes during level exchanges

i.e. $c' = c_{x_{i+1}=0}$ or $c' = c_{x_{i+1}=1} - c_{x_{i+1}=0}$. Let v be the node representing c, i.e. for some $m, a \in \mathbf{Z} : c = mv + a$. For v, we know that after exchanging the two levels, it holds

$$\begin{aligned} v_{(x_{i+1},low)} &= v_{x_{i+1}=0} \\ v_{(x_{i+1},high)} &= v_{x_{i+1}=1} - v_{x_{i+1}=0}, \end{aligned}$$

and that these functions are represented by some nodes v_0' and v_1', respectively. In case that $c' = c_{x_{i+1}=0}$, then c' can simply be represented by v_0', as $c' = c_{x_{i+1}=0} = mv_{x_{i+1}=0} + a = mv_0' + a$. In the other case, $c' = c_{x_{i+1}=1} - c_{x_{i+1}=0} = (mv_{x_{i+1}=1} + a) - (mv_{x_{i+1}=0} + a) = mv_1'$. Thus, all elements of C' can be represented using nodes of set V' only. □

Remark 1 If neighboring levels do not interact, the resulting level sizes remain unchanged, i.e. $\text{label}'(x_i) = \text{label}(x_i)$ and $\text{label}'(x_{i+1}) = \text{label}(x_{i+1})$. This can be checked efficiently using an interaction matrix [26].

3.2 Lower Bounds for one Variable

In this section we give lower bounds for the WLDD size if a single variable is moved in the variable ordering and the relative order of all other variables remains unchanged. Since these bounds follow from Theorem 1 in the same way as for BDDs [19], no proofs are given in the following.

Let $f : \mathbf{B}^n \to \mathbf{B}^m$ be a Boolean function represented by a WLDD and assume that the initial variable ordering is $(x_1, x_2, \ldots, x_n)$. Let $X_j^i := \{x_i, x_{i+1}, \ldots, x_{j-1}, x_j\}$ denote a set of variables. The definition of $\text{label}(\cdot)$ is extended to a set of variables using

$$\text{label}(X_j^i) := \sum_{k=i}^{j} \text{label}(x_k).$$

Corollary 1 Let $1 \le i < j \le n$. If level i is *moved down* to level j, resulting in the variable ordering

$$(x_1, x_2, \ldots, x_{i-1}, x_{i+1}, \ldots, x_j, x_i, x_{j+1}, \ldots, x_n),$$

for the size of the resulting WLDD $|G'|$ holds

$$|G'| \ge \text{label}(X_{i-1}^1) + 1 + \tfrac{1}{2} \cdot \text{label}(X_j^{i+1}) + \text{label}(X_n^{j+1})$$

and

$$|G'| \ge \text{label}(X_{i-1}^1) + \text{label}(x_i) + \text{label}(X_n^{j+1}).$$

```
SiftingDown(i : level to sift)
    lb = compute_lower_bound();
    best = size_of_BDD();
    while (i < n and lb ≤ best)
        level_exchange(i, i + 1);
        lb = compute_lower_bound();
        if (size_of_BDD() < best)
            best = size_of_BDD();
        i = i + 1;
```

Figure 4. Sketch of the "sifting down"-algorithm

Corollary 2 Let $1 \le j < i \le n$. If level i is *moved up* to level j, resulting in the variable ordering

$$(x_1, x_2, \ldots, x_{j-1}, x_i, x_j, \ldots, x_{i-1}, x_{i+1}, \ldots, x_n),$$

for the size $|G'|$ of the resulting WLDD holds

$$\begin{aligned} |G'| \ge\; & \text{label}(X_{j-1}^1) + (i-j) + \tfrac{1}{2^{i-j}} \cdot \text{label}(x_i) \\ +\; & \text{label}(X_n^{i+1}). \end{aligned}$$

These lower bounds can directly be integrated in the sifting algorithm. In Figure 4, a sketch for the case that a variable is sifted down is given.

4 DTL-Sifting

In order to make use of the full potential of K$*$BMDs, DTL-sifting can be used as DVO method instead of sifting (see Section 2.4). When a variable is moved up, the same lower bound can be used as for sifting. However, moving a variable down cannot be stopped sooner, as it is necessary to have the variable in the lowest level in order to modify the decomposition type. Unfortunately, the lower bound for moving a variable up is not as good as for the other direction. Therefore, DTL-sifting cannot be sped up much using this approach.

One way out of this dilemma is the following: Instead of unconditionally continuing to move a variable x_i downwards, the decomposition type of the variable is directly modified, which requires a synthesis operation to compute the new functions of the outgoing edges: For each node labeled with x_i, the outgoing *high*-edge is replaced by the arithmetic difference ($S \to pD$) or the arithmetic sum ($pD \to S$) of both successors (see Figure 5).

If moving the variable down could be stopped early due to the computed lower bound, the computation of the

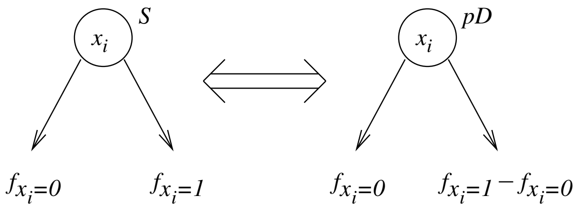

Figure 5. Using a synthesis operation to modify the DTL

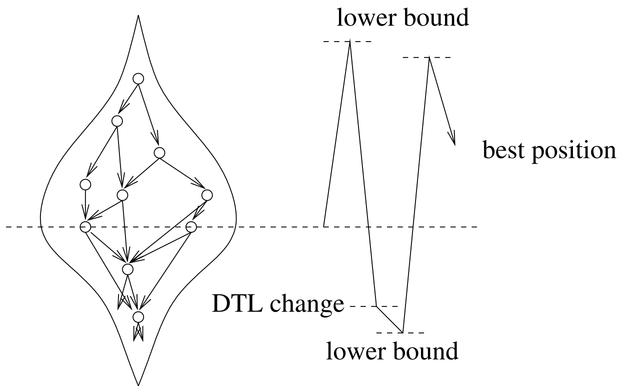

Figure 6. DTL-sifting for one variable using synthesis operations

synthesis operation to modify the decomposition type will probably be cheaper than moving the variable over a long distance in the variable order. On the other side, if the variable had to be shifted to the lower end of the K∗BMD in spite of the computed lower bounds, then in both cases the cost to modify the decomposition type is small.

This leads to the following procedure (see Figure 6):

1. The variable is exchanged with its predecessor variable until it is the topmost variable or the lower bound has been reached.
2. The variable is exchanged with its successor until it is the last variable or the lower bound has been reached.
3. The decomposition type of the variable is modified, i.e. if the decomposition type was S, it is set to pD, and vice versa, using a **synthesis operation**.
4. The variable is exchanged with its successor until it is the last variable or the lower bound has been reached. (This has to be done since the lower bound may have changed because of the previous step.)
5. The variable is exchanged with its predecessor until it is the topmost variable or the lower bound has been reached.
6. The variable is moved back to the closest position which led to the minimal size of the DD. If the decomposition type before has led to a better size, then the type is modified again using a synthesis operation.

5 Experimental Results

In this section we describe experimental results that have been carried out on a *SUN Ultra 1* with 256 MBytes. All runtimes are given in CPU seconds. The algorithm has been integrated in the TUDD package [22].

In a first series of experiments, we compare the original sifting algorithm [27] with the version using the lower bound improvement. The latter one will be referred as lb-sifting. Results are given in the left part of Table 1. In column *circuit* the name of the benchmark circuit is given and column *in* refers to the circuit's number of inputs. First, the K∗BMD was built using decomposition types pD only (column *initial*). The outputs were grouped together in the same order as written in the original description.

The size of the K∗BMD after applying sifting is shown in column *final*. In most cases lb-sifting results in the same sizes. However, in a few cases the optimal position for one variable is not unique, and therefore lb-sifting may result in different sizes. The only difference we observed was *c5315*, where the final size of lb-sifting was one node larger than after sifting.

In columns *sifting* and *lb-sifting*, the runtime is given for a final call of sifting and lb-sifting, respectively. The improvements are given in the next column. As can be seen lb-sifting needs on average 40% less runtime.

In a second series of experiments, we study the improved DTL-sifting. Results are given in the right part of Table 1. In contrast to the previous experiments, we already used DTL-sifting when constructing the K∗BMD. It can be seen that using the improvements discussed in Section 4, runtime can be reduced again by more than 40% on average.

It was also very surprising that runtimes never got larger when synthesis operations were used instead of level exchanges to modify the decomposition type of a variable. This clearly demonstrates the efficiency of the approach.

6 Conclusions

To speed up the time-consuming process of dynamic variable reordering, we proved a lower bound technique which can be used in an automated way during sifting of WLDDs. For DTL-sifting, the lower bounds cannot be used directly. Therefore, we described a method to use synthesis operations to make use of the lower bounds. Experimental results showed a speed-up of more than 40% on average for both methods.

References

[1] L. Arditi. *BMDs can delay the use of theorem proving for verifying arithmetic assembly instructions. In *FMCAD*, pages 34–48, 1996.

[2] R.I. Bahar, E.A. Frohm, C.M. Gaona, G.D. Hachtel, E. Macii, A. Pardo, and F. Somenzi. Algebraic decision diagrams and their application. In *Int'l Conf. on CAD*, pages 188–191, 1993.

[3] C.W. Barrett, D.L. Dill, and J.R. Levitt. A decision procedure for bit-vector arithmetic. In *Design Automation Conf.*, pages 522–527, 1998.

[4] B. Becker, R. Drechsler, and R. Enders. On the computational power of bit-level and word-level decision diagrams. In *ASP Design Automation Conf.*, pages 461–467, 1997.

Table 1. Experimental results

circuit	in	no DTL changes					DTL sifting				
		initial	final	sifting	lb-sifting	difference	initial	final	sifting	lb-sifting	difference
alu4	14	890	617	1.9	1.3	-31.6 %	890	506	2.7	1.7	-37.0 %
apex6	135	998	407	7.9	7.5	-5.1 %	998	445	15.1	11.0	-27.2 %
c432	36	16194	13801	172.9	68.4	-60.4 %	15392	2512	94.9	57.1	-39.8 %
c880	60	57596	35066	297.4	213.9	-28.1 %	49345	37669	570.3	360.4	-36.8 %
c2670	233	4208	3238	74.6	50.4	-32.4 %	9238	3736	244.6	149.0	-39.1 %
c5315	178	24210	23715	272.6	210.8	-22.7 %	22132	21596	588.3	349.6	-40.6 %
dalu	75	1435	909	10.5	7.8	-25.7 %	1533	473	14.5	9.2	-36.6 %
des	256	4667	4188	103.2	87.5	-15.2 %	4404	4054	200.2	120.5	-39.8 %
frg2	143	1022	584	13.4	11.4	-14.9 %	1022	584	28.4	17.7	-37.7 %
i2	201	453	411	19.7	10.5	-46.7 %	453	225	29.7	12.9	-56.6 %
i4	192	560	378	8.4	6.8	-19.0 %	560	509	16.4	10.9	-33.5 %
i8	133	3135	798	30.4	27.5	-9.5 %	3135	649	53.3	35.8	-32.8 %
i9	88	873	569	9.0	6.5	-27.8 %	873	425	21.4	12.7	-40.7 %
i10	257	86902	74349	2378.2	1271.9	-46.5 %	85679	37432	2758.4	1522.0	-44.8 %
k2	45	1452	1003	6.2	5.2	-16.1 %	667	439	5.8	4.0	-31.0 %
rot	135	9992	7684	94.3	66.8	-29.2 %	14932	7487	194.5	124.7	-35.9 %
pair	173	6707	4644	72.2	63.7	-11.8 %	6856	4151	138.7	93.3	-32.7 %
too_large	38	1189	1045	6.3	4.7	-25.4 %	696	558	7.2	4.2	-41.7 %
x3	135	998	407	8.0	7.6	-5.0 %	998	445	15.3	11.1	-27.5 %
sum		223481	173813	3587.1	2130.2	-40.6 %	219803	123895	4999.7	2907.8	-41.8 %

[5] B. Bollig, M. Löbbing, and I. Wegener. On the effect of local changes in the variable ordering of ordered decision diagrams. *Information Processing Letters*, 59:233–239, 1996.

[6] B. Bollig and I. Wegener. Improving the variable ordering of OBDDs is NP-complete. *IEEE Trans. on Comp.*, 45(9):993–1002, 1996.

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

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

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

[10] Y. Chen, E. Clarke, P. Ho, Y. Hoskote, T. Kam, M. Khaira, J. O'Leary, and X. Zhao. Verification of all circuits in a floating-point unit using word-level model checking. In *FMCAD*, pages 389–403, 1996.

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

[12] E. Clarke, M. Fujita, P. McGeer, K.L. McMillan, J. Yang, and X. Zhao. Multi terminal binary decision diagrams: An efficient data structure for matrix representation. In *Int'l Workshop on Logic Synth.*, pages P6a:1–15, 1993.

[13] E.M. Clarke, M. Fujita, and X. Zhao. Hybrid decision diagrams - overcoming the limitations of MTBDDs and BMDs. In *Int'l Conf. on CAD*, pages 159–163, 1995.

[14] E.M. Clarke and X. Zhao. Word level symbolic model checking – a new approach for verifying arithmetic circuits. Technical Report CMU-CS-95-161, 1995.

[15] D. Cyrluk, O. Möller, and H. Rueß. *An Efficient Decision Procedure for the Theory of Fixed-Sized Bitvectors*, volume 1254 of *LNCS*. Computer Aided Verification, 1997.

[16] R. Drechsler. *Formal Verification of Circuits*. Kluwer Academic Publishers, 2000.

[17] R. Drechsler and B. Becker. – Ordered Kronecker functional decision diagrams – a data structure for representation and manipulation of boolean functions. *IEEE Trans. on CAD*, 17(10):965–973, 1998.

[18] R. Drechsler, B. Becker, and S. Ruppertz. The K*BMD: A verification data structure. *IEEE Design & Test of Comp.*, pages 51–59, 1997.

[19] R. Drechsler and W. Günther. Using lower bounds during dynamic BDD minimization. In *Design Automation Conf.*, pages 29–32, 1999.

[20] R. Drechsler, A. Sarabi, M. Theobald, B. Becker, and M.A. Perkowski. Efficient representation and manipulation of switching functions based on ordered Kronecker functional decision diagrams. In *Design Automation Conf.*, pages 415–419, 1994.

[21] M. Fujita, Y. Matsunaga, 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.

[22] S. Höreth. *TUDD: Darmstadt University Decision Diagram Package Release 0.8b*. University of Darmstadt, Germany, 1999.

[23] S. Höreth and R. Drechsler. Dynamic minimization of word-level decision diagrams. In *Design, Automation and Test in Europe*, pages 612–617, 1998.

[24] S. Höreth and R. Drechsler. Formal verification of word-level specifications. In *Design, Automation and Test in Europe*, pages 52–58, 1999.

[25] Y.-T. Lai and S. Sastry. Edge-valued binary decision diagrams for multi-level hierarchical verification. In *Design Automation Conf.*, pages 608–613, 1992.

[26] S. Panda and F. Somenzi. Who are the variables in your neighborhood. In *Int'l Conf. on CAD*, pages 74–77, 1995.

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