Grouping Heuristics for Word-Level Decision Diagrams

Rolf Drechsler Marc Herbstritt Bernd Becker

Institute of Computer Science
Albert-Ludwigs-University
79110 Freiburg im Breisgau, Germany
email: {drechsle,herbstri,becker}@informatik.uni-freiburg.de

Abstract

*Word-Level Decision Diagrams (WLDDs), like EVBDDs, *BMDs, HDDs, K*BMDs, are powerful tools in circuit verification. Especially for arithmetic circuits, like multipliers, for the first time formal verification was possible using WLDDs. Beside a good variable ordering and the decomposition types the size of a WLDD essentially depends on the grouping of the outputs.*

In this paper we study output grouping in more detail. We give examples that show that an exponential reduction or an exponential blow-up can be obtained dependent on grouping. We describe efficient heuristics for output grouping given a circuit description in the form of a netlist. Experimental results are given to demonstrate the efficiency of our approach.

1 Introduction

In the last few years *Decision Diagrams* (DDs) became the state-of-the-art data structure in circuit verification. *Binary DDs* (BDDs) introduced by Bryant in 1986 have in the meantime been frequently used and are also integrated in commercial tools. But recent studies have shown that BDDs fail for some arithmetic functions of large importance, like multipliers [6]. For this, several extensions of the basic concept have been proposed. Among them being *Word-Level DDs* (WLDDs), like EVBDDs [22], *BMDs [7], HDDs [11], K*BMDs [15], that allow to represent functions defined over Boolean variables, but with an integer domain. Furthermore, some of these DD-types also allow to represent functions using different decomposition types [11, 15]. In the meantime WLDDs have been integrated in verifications tools [1, 9] and are also used for symbolic model checking [12, 8]. In [21] HDDs have been applied to verification of circuits at the register transfer level.

As is well known all these DD-types are canonical data structures only, if a fixed variable ordering is used. But finding a good order even for BDDs is an NP-complete problem [4]. Furthermore, as outlined above good decomposition types have to be determined. Recently, dynamic minimization algorithms have been proposed in [20] that allow to reduce the WLDD size up to 99% analogously to the sifting approach for BDDs [24]. An other optimization criteria that has to be considered for WLDDs is the grouping of the output variables, i.e. which outputs should be considered together. Notice that e.g. *BMDs can not represent multipliers efficiently, if all outputs are considered separately, while the representation becomes small if the outputs are takes as one word.

In this paper we study in more detail the effect of output grouping. First, examples are given to show that grouping can give exponential reductions, but also exponential blow-ups. We present heuristics how to group the outputs of a given Boolean network. The algorithm analyses the circuit and computes the support of each output. Dependent on this the outputs are grouped. The heuristic is very fast and the runtime is polynomial in the circuit size. Experimental results are given for different WLDD-types. It turns out that our heuristic gives significant reductions of DD-sizes in most cases.

The paper is structured as follows: In Section 2 WLDDs are introduced. The effect of output grouping is described in more detail in Section 3. In Section 4 the heuristic is described and its complexity is analyzed. Experiments are given in Section 5. Finally, the results are summarized.

2 Word-Level Decision Diagrams

In this section notations and definitions are given that are important for understanding the paper. We give a brief overview on DDs. (For more details see [14, 20].)

All DDs are graph-based representations, where at each (non-terminal) node labeled with a variable x a decomposition of the function represented by this node into two subfunctions (the *low*-function and the *high*-function) is performed. In the following, we assume that the underlying graph is *ordered* and *reduced*, i.e. variables occur in the same order on all paths in the DD and functions represented by nodes of the graph are unique.

For bit-level DDs the following three decomposition have been considered [16]:

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

graph type	edge function
MTBDD, BMD, HDD	$f_e = f$
EVBDD	$f_e = a + f$
*BMD, p*BMD	$f_e = m \cdot f$
K*BMD, pK*BMD	$f_e = a + m \cdot f$

Table 1: Functions represented by edges.

Function f is represented at node v, while f_{low} (f_{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 at a node a decomposition of type S (D) is carried out this node is called a S-node (a D-node).

In this paper, we consider the same three decompositions for word-level functions, i.e. functions of the form $f : \mathbf{B}^n \to \mathbf{Z}$:

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

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 $\mathbf{Z}$ of integers.

Decomposition types are associated to the n Boolean variables $x_1, x_2, \ldots, x_n$ with the help of 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 ($i = 1, \ldots, n$).

Based on the notations and definitions above we now introduce the functions represented by an edge in a DD. The edge function f_e is obtained from the function of the node through multiplication or addition of integer values. For MTBDD [10, 2], BMD [7], HDD [11], EVBDD [22], *BMD [7], p*BMD [19], K*BMD [15], and pK*BMD [20] the corresponding functions f_e are given in Table 1 (a, m are integer numbers).

Edge-values are obtained from the node representation during the graph reduction phase [20]. Note that all remaining DDs from Table 1 are obtained by further restricting the K*BMD reduction rules and DTL.

Of course, the size of the representation for a given function f depends on the chosen variable ordering and DTL. There are examples which, for a fixed function, provide exponential gaps between representations with differing variable order and DTL [5, 3]. On the other hand, finding good variable orderings and DTLs turns out to be difficult [4] and therefore a lot of heuristics have been developed to find at least reasonable orderings and DTLs, the most successful being *Dynamic Variable Ordering* (DVO) [18, 24] and DVO with DTL change [13, 20].

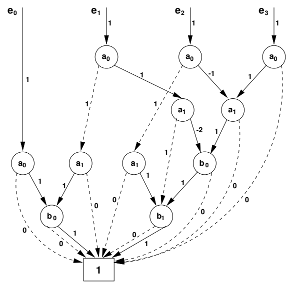

Figure 1: *BMD for multiplication without grouping

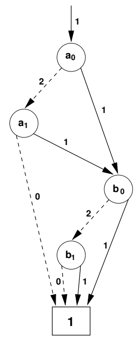

Figure 2: *BMD for multiplication with word encoding

3 Variable Grouping

Since WLDDs can represent functions of type $f : \mathbf{B}^n \to \mathbf{Z}$ also several outputs of a binary circuit can be grouped together by using a weighted sum of the outputs. By this, the DD size can be reduced or increased (dependent on the function). This will be illustrated by the following two examples:

Example 1 *First consider the multiplier function represented by a *BMD. If all outputs of the binary circuit are grouped together, the resulting graph has linear size, while the bit-wise representation has exponential size [7]. For $n = 2$ the corresponding *BMDs are given in Figure 1 and 2, respectively.*

Thus, grouping the variables can result in an exponentially smaller representation, but also the opposite might hold:

Example 2 *Consider MTBDDs for the functions $x_i : \mathbf{B}^n \to \mathbf{B}$ representing the ith variable. Obviously the MTBDD for each x_i consists of only one node. If MTBDDs for all variables are considered in parallel n nodes are needed. But the weighted sum representation*

$$sum := \sum_{i=0}^{n-1} x_i 2^i$$

```
for (i = 0; i < NPO; i++)
  for (j = 0; j < NPO; j++)
    for (l = 0; l < NPI; l++)
      if (i-th and j-th primary output
           is dependent on input l)
        corr[i][j] = corr[i][j] + 1 ;
```

Figure 3: OCM computation

has an exponential number of nodes, if represented by an MTBDD.

In the following we present a heuristic that starting from a netlist of gates determines a good grouping for WLDDs.

4 Topology-based Heuristic

Before giving a detailed description we explain the main idea: The heuristic in the following groups the outputs together that depend on the same variables. It starts with the largest output, i.e. the one with the largest support. Then all outputs that depend on the same variables (or on a subset) are grouped together until no further grouping is possible. Then the next largest output is considered until no output is left.

In the following the heuristic is described in detail. For each step the complexity is analyzed. (We assume that the number of inputs per gate is bound by a constant.)

Initialization First, for each gate an array of length *Number of Primary Inputs* (NPIs) is assigned. Then the support is computed by traversing the netlist in topological order. This can be done in time $O(NPI \cdot |G|)$, where G denotes the netlist.

Evaluation of Support Matrix Then the *Output Correspondence Matrix* (OCM) `corr` is computed. The matrix has $NPO \times NPO$ entries, where NPO denotes the *Number of Primary Outputs*. First, all entries are set to 0 in time $O(NPO^2)$. Then it is computed which outputs are defined over the same support. This is done by the algorithm in Figure 3. As can easily be seen the runtime of the algorithm is given by $O(NPO^2 \cdot NPI)$.

Variable Grouping Dependent on the OCM the variables are grouped. To do so the largest element of the diagonal is found. This determines the first element of the first group. Then it is checked which outputs are defined over the same (or a smaller) set of variables. For the output selection we propose two different methods:

Heuristic 1 (H1): Include an output, if its support is contained in the largest one.

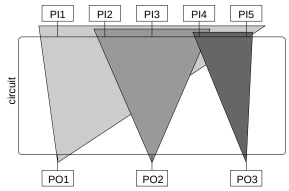

Figure 4: Example for describing the heuristics H1 and H2.

Heuristic 2 (H2): Include an output, if its support is contained in the support of all already selected.

These outputs are added to this group and removed from the matrix. This algorithm is iterated until no outputs are left. The complexity for both heuristics is given by $O(NPO^2)$. H2 is more conservative than H1. The advantages and disadvantages will be discussed in the next section.

Thus, the overall runtime is bound by $O(max(NPI \cdot |G|, NPO^2 \cdot NPI)$, and by this can be computed efficiently also for large circuits.

To simplify the understanding of the heuristics we give the following example:

Example 3 *In Figure 4 an abstract circuit is given showing the support for each PO. PO1 has the largest support, PO3 the smallest support and the support of PO2 is in between. Thus, both heuristics start grouping with PO1. In the next step, H1 as well as H2 incorporates PO2 into the group of PO1, since the support of PO2 is a subset of the support of PO1. The difference between H1 and H2 occurs when considering PO3. H1 checks whether the support of PO3 is a subset of the head of the leading group, namely PO1. Since this is the case H1 will result in one group consisting of PO1, PO2 and PO3. Whereas H2 behaves different: it checks whether the support of PO3 is a subset of all the supports of the POs which are actually in the group. Thus, since the support of PO3 is a subset of the support of PO1 but not a subset of the support of PO2 H2 will create a new group consisting only of PO2. Therefore, H2 will result in two groups: one group consisting of PO1 and PO2 and another group consisting of PO3.*

5 Experimental Results

In this section we give experimental results for benchmark circuits from LGSynth91 [25]. All experiments have been carried out on a *SPARC Ultra 1* with a node limit of 10^6 nodes.

Before the number of nodes are compared, we first give in Table 2 for all benchmarks some informations about the grouping. Column *name* gives the name of the benchmark, and in column *NPO* the number of outputs is given. H1 and H2 denote the number of groups resulting from heuristic 1 and 2, respectively.

name	NPO	H1	H2
alu2	6	1	3
alu4	8	1	3
apex6	99	76	93
apex7	37	24	30
x1	35	20	31
x3	99	76	93
addm4	8	1	1
max46	1	1	1
max128	24	1	1
max512	6	1	1
max1024	6	1	1
rd53	3	1	1
rd73	3	1	1
rd84	4	1	1
adder2	2	1	1
adder4	4	1	1
adder8	8	1	1
adder16	16	1	1
mult2	4	1	1
mult3	6	1	1
mult4	8	1	1
mult5	10	1	1

Table 2: Output grouping by H1 and H2

As can be seen, for most examples H2 has a larger number of groups and for this is more "bit-oriented", while H1 tries to bring many outputs together.

The effect of this can be seen in Table 3. Again, the first column denotes the name of the benchmark. In column *no grouping* results for *BMDs and S-K*BMDs, i.e. K*BMDs with only nodes of type S, are given, if the outputs are considered bit by bit. The results for heuristics H1 and H2 are given in columns *H1* and *H2*, respectively. For all examples the variable ordering has been determined using interleaving [17][1]. It turns out that heuristics H1 and H2 obtain improvements in size in most cases. For some examples the reduction is up to a factor of 3. Studying the results in more detail we see that H1 works very good for Davio based WLDD, like *BMDs, but in some cases gives bad results for Shannon decompositions. H2 is more conservative and gives not as much improvement for *BMDs, but it still results in small WLDDs for Shannon decomposition. All in all, H1 should be applied in Davio dominated WLDDs, while H2 is used for S-based WLDDs

Thus, we can observe, that output grouping should also consider the decomposition types in the WLDD. H1 works better for *BMDs, while H1 should be preferred when using Shannon decomposition.

[1] We also performed the same experiments using a fanin based order similar to [23], the results are nearly the same. For this, we restrict ourselves to interleaving here.

6 Conclusions

In this paper we presented heuristics for output grouping of WLDDs. It has been shown that large reductions are possible, i.e. up to an exponential factor. It turned out that grouping is also dependent on the chosen decomposition type of the variables.

It is focus of current work to incorporate output grouping in the variable ordering and decomposition type choice process, and to also consider "mixed" DTLs.

Acknowledgment

The authors like to thank Martin Keim for many valuable discussions.

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] 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.

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

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

[6] R.E. Bryant. On the complexity of VLSI implementations and graph representations of Boolean functions with application to integer multiplication. *IEEE Trans. on Comp.*, 40:205–213, 1991.

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

[8] 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.

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

[10] 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.

[11] 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.

[12] 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.

name	no grouping		H1		H2	
	*BMD	S-K*BMD	*BMD	S-K*BMD	*BMD	S-K*BMD
alu2	358	223	288	345	160	219
alu4	2807	1250	1567	2178	546	1256
apex6	1170	1625	1015	6173	1082	1690
apex7	927	5343	805	39656	860	4950
x1	1729	1581	1779	1925	1722	1581
x3	1383	1771	1248	9204	1313	1893
addm4	296	189	170	110	170	110
max46	208	85	208	85	208	85
max128	379	129	119	45	119	45
max512	481	237	314	131	314	131
max1024	1058	339	674	180	674	180
rd53	26	16	11	10	11	10
rd73	52	30	27	19	27	19
rd84	71	41	33	26	33	26
adder2	9	6	6	5	6	5
adder4	46	39	43	44	43	44
adder8	320	278	233	240	233	240
adder16	6610	5894	3961	3760	3961	3760
mult2	11	13	6	8	6	8
mult3	48	48	9	32	9	32
mult4	215	177	12	65	12	65
mult5	857	607	14	130	14	130

Table 3: Output grouping by H1 and H2

[13] R. Drechsler and B. Becker. Dynamic minimization of OKFDDs. In *Int'l Conf. on Comp. Design*, pages 602–607, 1995.

[14] R. Drechsler and B. Becker. *Binary Decision Diagrams - Theory and Implementation.* Kluwer Academic Publishers, 1998.

[15] R. Drechsler, B. Becker, and S. Ruppertz. K*BMDs: A new data structure for verification. In *European Design & Test Conf.*, pages 2–8, 1996.

[16] 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.

[17] 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.

[18] 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.

[19] S. Höreth. Implementation of a multiple-domain decision diagram package. In *CHARME*, Chapman & Hall, pages 185–202, 1997.

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

[21] G. Kamhi, O. Weissberg, and L. Fix. *Automatic Datapath Extraction for Efficient Usage of HDD*, volume 1254 of *LNCS.* Computer Aided Verification, 1997.

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

[23] 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.

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

[25] S. Yang. Logic synthesis and optimization benchmarks user guide. Technical Report 1/95, Microelectronic Center of North Carolina, Jan. 1991.