PLACEMENT AND ROUTING OPTIMIZATION FOR CIRCUITS DERIVED FROM BDDS

Thomas Eschbach[1] *Rolf Drechsler*[2] *Bernd Becker*[1]

[1]Institute for Computer Science
Albert-Ludwigs-University
79110 Freiburg, Germany
{eschbach,becker}@informatik.uni-freiburg.de

[2]Institute for Computer Science
University of Bremen
28359 Bremen, Germany
drechsle@informatik.uni-bremen.de

ABSTRACT

The high complexity of circuits which currently consist of several millions of transistors, can only be managed using a concise design flow. Recently, the One-Pass Synthesis paradigm came up, i.e. to consider the whole design process as one flow instead of isolated steps. In this context, designing circuits based on the mapping of Binary Decision Diagrams (BDDs) shows several advantages.

While various BDD based approaches for logic minimization or design for testability have been proposed, in this paper we show that placement and routing of BDD circuits can be optimized at a high level of abstraction. Based on algorithms for reducing the number of nodes and edge crossings, we demonstrate on multiple benchmarks that significant improvements are possible in reasonable time.

1. INTRODUCTION

Technological advances in the last decades have allowed the production of large chips with millions of gates. Usually, when producing a chip, the design flow is split up into several individually optimized steps. In almost all cases, each one of these steps is supported by a powerful automated design tool, for example:

- high-level and logic synthesis
- mapping
- place and route

Significant effort has been spent trying to improve these individual steps, but in many cases, the optimization criterion of consecutive steps are different and their close interaction has not been considered sufficiently. Thus, the resulting designs were often suboptimal. E.g., a highly optimized technology independent netlist may produce a suboptimal final design when the mapping onto the target architecture is done. In many cases the logic optimization step does not use the underlying basic cell structure on the physical chip optimally. This can be avoided if the logic synthesis process takes the criteria of the final mapping and layout step into account.

Normally, several iterations of the complete design process have to be carried out to get reasonable results. This is a time consuming and expensive process. Therefore it makes sense to consider the final layout in earlier phases. The interaction between the synthesis, placement and routing phases are important since several quality related criteria are directly influenced. E.g. consideration of the interaction can result in smaller delays, less wastage area, lower power consumption, less crosstalk and better testability. As a promising solution to the resulting problems the One-Pass Synthesis methodology has been proposed. There are two main underlying ideas:

- to combine optimization steps that were split before
- to restrict the optimization in one level such that it fits better on the next

For a more detailed introduction see [1].

One very powerful approach in this context is based on circuits derived from a one-to-one mapping of BDDs [2]. The resulting circuits have nice properties regarding testability [3, 4] and power consumption [5]. To avoid crossings in the physical layout, redundant hardware (dummy nodes) can be inserted to obtain non-crossing ordered BDDs (NCOBDDs) as proposed in [6]. The dummy node insertion also allows a fine grained pipelining where every layer corresponds to a stage of a pipeline. Instead of decomposing the circuit in small macrocells, each containing a small number of BDD nodes (for example around twenty in [7]), and then laying out and routing the macrocells, the approach in [6] as well as the approach presented in this paper use the layered rooted tree structure of the BDD to generate a placement and routing for all nodes in one step.

In this paper, starting with a BDD representation whose size is optimized by using the sifting algorithm [8], we present a method for reducing wire crossings during BDD mapping without the addition of dummy nodes. We make use of the layered structure and the absence of cycles implied by the restricted order of BDDs. Based on an optimized BDD variable ordering, i.e. the variable ordering is fixed, we determine for each level an ordering of the nodes corresponding to multiplexer cells. Experiments show that the number of crossings can be considerably reduced.

2. PROBLEM DESCRIPTION

We give some basic properties of Binary Decision Diagrams (BDD), as far as they are important for the purposes of this paper. More details can be found e.g. in [9].

A reduced ordered Binary Decision Diagram (BDD) as introduced in [2] is a directed acyclic graph $G = (V, E)$ in which a Shannon decomposition is carried out in each node v that is not a sink:

$$f_v = \overline{x}_i f_{v_{x_i=0}} + x_i f_{v_{x_i=1}} \quad (1)$$

x_i is called the decision variable in v. In a reduced ordered BDD each path from a source to a sink is consistent with a given ordering of the decision variables. (For an example see Figure 2.)

The size of the BDD is very sensitive to the chosen variable ordering, i.e. the BDD size may vary from linear to exponential in the number of variables for a given function f. In general, improving the variable ordering of BDDs is NP-complete [10]. However, efficient heuristic algorithms for improving the variable ordering are known, in particular the sifting method [8]. Currently, BDDs are commonly used for efficient representation and manipulation of Boolean functions, not only in the VLSI CAD community [11].

In the following, we consider synthesis approaches where circuits are derived from BDDs by a simple one-to-one mapping. This can easily be done by introducing a multiplexer for every node and corresponding wires for every edge of the BDD. A small example for the one-to-one mapping of one BDD node (a) into a multiplexer circuit (b) is given in Figure 1. Please note, that in the following BDD nodes are drawn as circles in contrast to derived multiplexers given as rectangles labeled with the corresponding decision (input) variable.

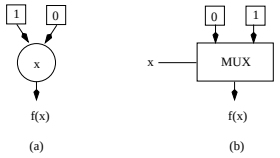

Fig. 1. Example transformation of a BDD (a) into a multiplexer based circuit (b).

Even though BDDs were not designed to produce the physical layout of a circuit, there were already very promising approaches to BDD-based circuit design. (For an overview see [1].) For totally symmetric functions, BDDs have a very regular structure that can be directly transferred into planar layouts. The nodes are locally connected to a maximum of four adjacent nodes. This leads to a layout with less crosstalk and short wires. For non-symmetric functions an approach to transfer the BDD into a lattice structures was proposed in [12]. It produces the same layout except for the fact that additional levels are needed. The main drawback of these methods is that the number of levels for some functions may become exponential in the number of variables. On the other hand, the NCOBDDs [6] only require one layer for every input variable at the cost of additional dummy nodes. Every BDD can be transformed into a NCOBDD by duplicating and reordering the nodes of the BDD to achieve a planar layout. These dummy nodes are then directly mapped onto the circuit using additional area and increase the energy consumption.

Due to this, in the following we consider "traditional BDDs", i.e. reduced ordered BDDs [2]. They provide a good representational compromise between regularity and compactness. To reduce the number of edge crossings of the given BDD in the layout process, a crossing reduction algorithm is applied. For an example see Figure 2.

Then it can be mapped onto a multiplexer based target technology. This step is an important part for the final layout of the circuit since unnecessary edge crossings complicate the process of routing. After this, the algorithm maps the nodes of the BDD to sub-circuits realizing the corresponding functions.

In the following we formulate the edge crossing reduction problem as a graph problem and introduce some common terms from this field: A directed graph $G = (V, E)$ is a *multi-layered* graph with d layers if the node set V is partitioned into disjoint subsets V_1, V_2, ..., V_d, i.e. $V_1 \cup V_2 \cup \ldots \cup V_d = V$ and $(\forall m \neq m')$

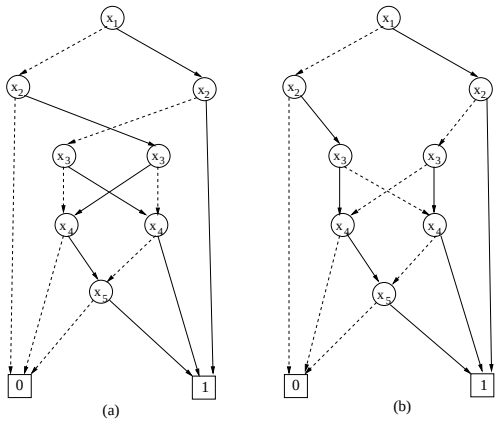

Fig. 2. This example shows a BDD before (a) and after (b) the crossing minimization.

$V_m \cap V_{m'} = \emptyset$, where V_m is called the *m-th layer* of the graph. All edges in E connect nodes in different layers. All nodes of the BDD which are labelled with the same variable can be assigned to the same layer to obtain a multi-layered graph.

A traditional approach to lay out a directed graph was introduced by [13]. It splits up the layout process into four steps:

- Cycle Removal: To obtain an acyclic graph, as few edges as possible are reversed.
- Layer Assignment: A proper layering will be computed by assigning every node to one layer.
- Crossing Reduction: The algorithm computes an ordering for all nodes on each layer which minimizes the total number of edge crossings ("multi-layer straight-line crossing minimization problem").
- X-coordinate assignment of all nodes.

This directed graph layout framework is well suited to lay out the BDD structure. Since BDDs are layered directed acyclic graphs, the goals of the first and the second step are already achieved.

To solve the (exact) multi-layer straight-line crossing minimization problem we have to determine an ordering ord_m for all layers containing all nodes in layer V_m so that the number of crossings is minimized. In the following, a set of orderings ord_m, $m \in \{1, \ldots, d\}$, is called an ordering for the graph. Unfortunately, even minimizing edge crossings in graphs with only two layers is NP-hard [14] and remains NP-hard if the ordering in one of the layers is fixed. Therefore the use of heuristic methods to solve this problem is justified. Many heuristic algorithms are known from literature, e.g. [15, 13, 16]. In this paper we use the popular *averaging* method first introduced by [13] which computes good solutions in a short time.
We then further reduce the number of crossings by post-processing it with the windows optimization heuristic method [17]. It decomposes the graph into smaller subgraphs which contain nodes form several layer and then computes an optimal ordering for each subproblem.

3. PLACEMENT AND CROSSING REDUCTION

In this work we first compute a BDD for a given circuit. The algorithm dynamically reorders the input variables to reduce the num-

ber of BDD nodes using sifting. This step directly influences the number of multiplexers used to implement the circuit, and thus on the properties of the resulting layout.
Then the averaging heuristic method is applied to reduce the number of edge crossings to obtain a good initial ordering which is then post-processed with the windows optimization algorithm. The averaging method helps to cut down overall run times, since a good starting point helps to avoid unnecessary calls of the locally optimal crossing reduction algorithm. Pseudo code for the complete flow is given below:

```
compute_ordering(circuit) {
  BDD = Compute_BDD_using_Sifting(circuit)
  rord = compute random ordering
  ordering = averaging(BDD, rord)
  ordering =windowsopt(BDD, ordering)
  return ordering
}
```

In the remaining part of this section the averaging and the windows optimization algorithm are briefly described.

3.1. Averaging heuristic method

The averaging heuristic method computes the position of node n on one layer with respect to the nodes on the layer above (below) which are directly connected to it. (Through the introduction of dummy nodes that are removed after the ordering has been computed we may assume that there are only edges between adjacent nodes.) It then sorts the nodes with respect to this pre-computed value. To compute a solution for the multi-layer straight-line crossing minimization problem the averaging technique makes use of the so called layer-by-layer sweep:

- Choose an initial ordering.
- Fix the positions of all nodes on the first layer.
- Compute the positions of all nodes on the second layer with respect to the fixed nodes located on the first layer.
- Fix the positions of the nodes on the second layer and compute the position of the nodes located on the third layer and so on.
- Start to sweep back, processing the second last layer considering only the nodes on the last layer until the ordering of the first layer is computed.
- Repeat the procedure until no further reduction in the number of crossings is achieved.

3.2. Windows Optimization

We only give a short description of the windows optimization heuristic method. Further details can be found in [17]. An initial ordering can be improved in the following way. A series of subsets of nodes with constant size, typically spreading over several layers, are extracted and processed with an exact approach with respect to their adjacent nodes. This approach is based on a dynamic programming method which makes use of a lower bound technique to reduce the search space. Only if the local solution induces a crossing reduction for the entire graph, is the new ordering used. The user has a fine grained control on the trade off between run time and solution quality by choosing the size of the window. The algorithm starts with a window width of four nodes per layer and a window depth of two layers. Then the algorithm increases the window depth to three and finally four layers dynamically to further reduce the number of crossings.

4. EXPERIMENTAL RESULTS

All algorithms are implemented in C. The experimental results are based on examples taken from benchmark circuits in [18] and [19]. The experiments were carried out on a 2 GHz personal computer with 1 GB main memory running the Linux OS. The run times are given in CPU seconds.

We utilized the $CUDD$ package [20] to compute a BDD for every circuit. $CUDD$ is a commonly accepted software tool to efficiently minimize and manipulate BDDs. Sifting [8] was used during the construction of the BDDs. The averaging heuristic method from section III.A was used to obtain a good initial embedding of the nodes. As observed in [21, 17], the averaging method computes high quality results in a short period of time compared to many other methods used in this field. Then windows optimization is applied.

In Table 1 the results of the implemented procedures are given. The second (third) column shows the number of multiplexers for each circuit without (with) using sifting. Next, in the fourth column we present the number of edge crossing produced by a random permutation of the multiplexers in each layer. The fifth and sixth columns provide the final results in terms of edge crossing (computed with the averaging heuristic method and after post-processing it with the windows optimization technique). The total run times for the averaging and the windows optimization method are given in the last two columns.

As already reported in [8] sifting reduces the number of nodes significantly. For completeness we give the precise numbers without using sifting in the second column. Compared to the averaging procedure, a random assignment of the nodes produces results which have on average nearly 200 times more crossings. Post-processing the results with the windows optimization technique further reduces the number of crossings on average by seven percent.

Table 2 is given to compare our results with the results utilizing the NCOBDDs published in [6]. In the second column the number of multiplexers required when using the NCOBDD structure is shown. Finally, the third column presents the number of multiplexers that are needed to obtain a circuit derived form the corresponding BDD. The additional dummy nodes in the NCOBDDs are necessary to allow a planar layout (and also pipelining). Of course, our approach does not compute crossing free layouts but it saves up to 90 % of the multiplexers compared to the NCOBDDs given in [6].

5. CONCLUSIONS

We have proposed a new approach to compute placement and routing for a circuit derived from the acyclic graph structure of the corresponding BDDs. This placement can be further improved by reordering the nodes within the layers to reduce the number of wire crossings. An algorithm was implemented that reduces the number of multiplexers and the number of wire crossings used. In this way, layout aspects can be considered at a high level of abstraction, and therefore in an early design phase. Experiments have shown the ef-

Table 1. Benchmark results of circuits

				quality		time/s	
circuit	mplex	sift	rand	av	wo	av	wo
z4ml	33	16	157	3	3	0.27	1
cm138a	26	17	563	10	10	0.23	1
9sym	36	24	1328	8	8	0.49	3
cmb	53	28	2028	11	10	0.58	4
cu	75	31	1851	30	30	0.55	4
decod	41	40	1808	67	63	0.39	6
cm85a	50	36	1548	13	13	0.15	2
x2	51	36	1802	49	42	0.48	5
f51m	55	39	1512	51	48	0.43	3
cm162a	63	41	3477	23	23	0.71	3
pcle	64	41	4874	96	93	0.89	10
pm1	60	43	5587	37	35	0.88	5
i1	90	43	6923	50	50	1	8
cordic	78	49	3833	10	9	1	9
cc	85	60	5417	109	103	0.69	13
lal	128	75	14055	192	183	1	27
unreg	104	81	3379	140	128	0.49	12
count	119	81	43778	229	213	3	83
c8	112	81	18432	296	260	1	42
sct	124	82	9704	139	130	1	16
term1	424	96	18804	73	67	2	31
frg1	280	102	16084	257	244	2	35
pcler8	141	107	27243	336	326	2	30
b9	202	106	26314	181	177	2	36
ttt2	198	121	48158	548	509	3	95
i3	262	132	33421	0	0	3	28
cht	142	124	14889	621	560	0.97	38
comp	174	136	49102	143	135	4	219
alu2	187	168	17448	1578	1523	1	53
i2	1679	205	224872	69	69	17	154
i6	422	214	36755	4090	3806	1	99
apex7	1071	637	195568	1562	1521	7	287
i4	622	337	189885	129	121	8	169
i5	1053	322	193037	1130	1119	7	171
alu4	426	378	133126	5602	5309	4	371
i7	513	396	76898	5810	5273	2	128

ficiency of this approach. Current work focuses on the integration of the new algorithm into a complete layout environment.

Table 2. NCOBDDs vs. BDDs

circuit	multiplexers(NCOBDD)	multiplexers(BDD)
z4ml	77	16
cm138a	48	17
9sym	33	24
alu2	465	168
alu4	3482	378

6. REFERENCES

[1] R. Drechsler and W. Günther, *Towards One-Pass Synthesis*, Kluwer Academic Publishers, 2002.

[2] R.E. Bryant, "Graph - based algorithms for Boolean function manipulation," *TOC*, vol. 35, no. 8, pp. 677–691, 1986.

[3] P. Ashar, S. Devadas, and K. Keutzer, "Path-delay-fault testability properties of multiplexor-based networks," *INTEGRATION, the VLSI Jour.*, vol. 15, no. 1, pp. 1–23, 1993.

[4] B. Becker, "Testing with decision diagrams," *INTEGRATION, the VLSI Jour.*, vol. 26, pp. 5–20, 1998.

[5] P. Lindgren, M. Kerttu, M. Thornton, and R. Drechsler, "Low power optimization technique for BDD mapped circuits," in *Proceedings of Asia and South Pacific Design Automation Conference (ASP-DAC'01)*, 2001, pp. 615–621.

[6] A. Cao and C. K. Koh, "Non-Crossing Ordered BDD for Physical Synthesis of Regular Circuit Structure," in *Int'l Conf. on Comp. Design*, 2003.

[7] L. Macchiarulo, L. Benini, and E. Macii, "On-the-fly layout generation for ptl macrocells," *IEEE Design Automation and Test in Europe*, pp. 546–551, 2001.

[8] R. Rudell, "Dynamic variable ordering for ordered binary decision diagrams," *ICCAD*, pp. 42–47, 1993.

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

[10] B. Bollig and I. Wegener, "Improving the variable ordering of OBDDs is NP-complete," *IEEE Trans. on Comp.*, vol. 45, no. 9, pp. 993–1002, 1996.

[11] U. Wegener, *Branching programs and binary decision diagrams - theory and applications*, SIAM Monographs on Discrete Mathematics and Applications, 2000.

[12] M. Chrzanowska, Z. Wang, and Y. Xu, "A regular representation for mapping to fine-grain, locally-connected fpgas.," *In Proc. Midwest Symp. Circ. Syst*, pp. 2749–2752, 1997.

[13] K. Sugiyama, S.Tagawa, and M.Toda, "Methods for visual understanding of hierarchical system structures," *IEEE Transaction on Systems, Man and Cybernetics*, vol. 11, no. 2, pp. 109–125, 1981.

[14] M. R. Garey and D. S. Johnson, "Crossing number is NP-complete," *SIAM Journal on Algebraic and Discrete Methods*, vol. 4, pp. 312–316, 1983.

[15] P. Eades and D. Kelly, "Heuristics for reducing crossings in 2-layered networks," *Ars Combin.*, vol. 21, no. A, pp. 89–98, 1986.

[16] J. Warfield, "Crossing theory and hierarchy mapping.," *IEEE Transactions on System, Man, and Cybernetics*, vol. SMC-7, no. 7, pp. 505–523, 1977.

[17] T. Eschbach, W. Günther, R. Drechsler, and B. Becker, "Crossing Reduction by Windows Optimization," *Proceedings of the 10th International Symposium on Graph Drawing*, vol. LNCS 2528, pp. 285–294, 2002.

[18] F. Brglez, D. Bryan, and K. Kozminski, "Combinational profiles of sequential benchmark circuits," *Int'l Symp. Circ. and Systems*, pp. 1929–1934, 1989.

[19] K. McElvain, "Benchmark set: Version 4.0," *International Workshop on Logic Synthesis*, 1993.

[20] F. Somenzi, *CUDD: CU Decision Diagram Package Release 2.3.1*, University of Colorado at Boulder, 2001.

[21] M. Jünger and P. Mutzel, "2-layer straightline crosing minimization: Performance of exact and heuristic algorithms," *J. Graph Agorithms Appl.*, vol. 1, no. 1, pp. 1–25, 1997.