IEEE EUROMICRO Conference (EUROMICRO'00)
Maastricht, September 2000

ACTion: Combining Logic Synthesis and Technology Mapping for MUX based FPGAs

Wolfgang Günther Rolf Drechsler

Institute of Computer Science
Chair of Computer Architecture (Prof. Dr. Bernd Becker)
Albert-Ludwigs-University
79110 Freiburg im Breisgau, Germany
{guenther,drechsle}@informatik.uni-freiburg.de

Abstract

Technology mapping for Multiplexor (MUX) based Field Programmable Gate Arrays (FPGAs) has widely been considered. Here, a new algorithm is proposed that applies techniques from logic synthesis during mapping. By this, the target technology is considered in the minimization process. Binary Decision Diagrams (BDDs) are used as an underlying data structure due to the close relation between BDDs and MUX netlists. The algorithm uses local don't cares obtained by a greedy algorithm. The mapping is sped up by computing signatures. A trade-off quality versus runtime can be specified by the user by setting different parameters. Experimental results comparing the approach to the best known results show improvements of more than 30% for area and 40% for delay for many instances.

1 Introduction

Field Programmable Gate Arrays (FPGAs) have been widely used in implementations of *Application Specific Integrated Circuits* (ASICs) due to several reasons, like short turnaround time and time-to-market aspects. FPGAs are mainly based on a basic cell that consists e.g. of a *Look-Up Table* (LUT) or a *Multiplexor* (MUX) structure. Several successful approaches to technology mapping have been proposed in the last few years [15, 13, 3, 19, 25].

Technology mapping is usually applied after a technology independent logic optimization phase. During technology mapping the optimized network is transformed to a functional equivalent network respecting the underlying circuit architecture. But since this structure is not considered during logic synthesis the resulting mapped circuits are often far from being optimal.

Several approaches to combine logic synthesis and technology mapping have been already proposed using *Pass Transistor Logic* (PTL) as a target technology. In [5] a decomposed BDD representation and variable reordering is used. In [6], a BDD based synthesis flow for PTL is presented. Basic operations to cells are applied. Also local don't cares are used to minimize the BDDs of cells.

In this paper we present a new approach to combine technology mapping and logic synthesis. As a target architecture we use FPGAs based on Actel-1 cells [1]. The algorithm starts with a pre-optimized netlist, obtained by a standard tool, like SIS [23]. However, instead of only mapping the circuit to the target technology, also further global optimizations with respect to technology are performed. After each modification, the mapper is run again and the mapped netlists are compared (instead of using e.g. the literal count as an estimation for the resulting circuit size). It has been observed in [7, 11] that this combination tends to be time consuming. However, in most cases only small parts of the netlist are affected by one modification and therefore the mapper only needs to be re-run on the modified parts of the netlist, allowing a fast evaluation. Furthermore, we use an efficient representation of the subcircuits, i.e. we use *Binary Decision Diagrams* (BDDs) [4]: BDDs for cells of the netlist are merged and the resulting BDDs are mapped. To minimize the size of the BDDs, different variable ordering techniques are applied. We also use sophisticated techniques for local don't care assignment.

The algorithm has been implemented as the program ACTion. We give comparisons to many other approaches, including *ite_map* [13, 14, 19] which is integrated in SIS. We show that significant improvements can be obtained using ACTion.

The paper is structured as follows: The detailed problem formulation is given in Section 2. In Section 3 the new approach to combine technology mapping with logic synthesis is described. Experimental results are given in Section 4. Finally, the results are summarized.

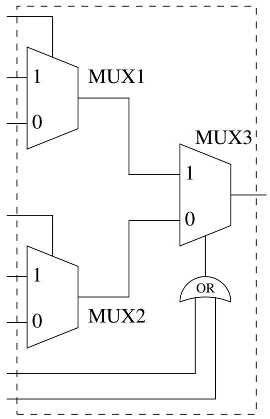


Figure 1. Architecture of Actel-1

```
map(MUX netlist)
    set_of_gates_to_realize := set of output gates;
    while (set_of_gates_to_realize is not empty)
        g := first element of set_of_gates_to_realize;
        do
            for (all fanin gates h)
                merge g and h, if the resulting cell
                is realizable with one Actel-1 cell;
        while (improvement);
        insert Actel-1 cell to mapped netlist;
        insert fanin gates to set_of_gates_to_realize;
```

Figure 2. Sketch of mapping algorithm

2 Problem Formulation

During technology mapping a given netlist is transformed to a functional equivalent representation with respect to a target technology. One of the main problems results from the fact that this target technology is usually not considered during logic synthesis.

The approach in this paper focuses on one specific cell type, i.e. the Actel-1 cell. The basic cell is given in Figure 1. Several technology mappers have been proposed in the past for this architecture (see e.g. [13, 18]). Note that our approach can easily be extended to other cell types. However, a MUX based structure is advantageous because of the similarity to BDDs.

For the rest of the paper we mainly focus on area minimization, i.e. we try to find a representation that minimizes the number of cells needed to represent the circuit. Even though delay is not the main optimization objective, it turns out that the resulting circuits often have very good delay. Choosing an other evaluation function, also other criteria can be incorporated, like power dissipation.

3 Combining Logic Synthesis and Technology Mapping

In this section the overall flow of the algorithm is described. To give a better understanding of the interaction between logic synthesis and technology mapping, the main ideas of the mapping algorithm are outlined in Section 3.1. Then the tool implemented as the program ACTion is described in Section 3.2.

3.1 Actel-1 Mapper

A relatively simple greedy strategy is used to map MUX based netlists to Actel-1 cells: Starting from the primary outputs of the circuit, a gate is chosen and assigned to a new Actel-1 cell. Then the fanin gates are added to the cell one after another until no further gate fits into the cell. Now all gates in the fanin which could not be added to the cell become "output" cells since they have to be represented by separate cells. A sketch of the mapping algorithm is given in Figure 2.

To decide whether a gate can be added to an Actel-1 cell, in a preprocessing step the set of functions which can be realized with one cell is computed. Permutation invariant signatures (see e.g. [17]) are used to identify most of the computed results. We used the following signatures for a *function* f:

- The number of minterms of f.
- The Boolean values $f(0,\ldots,0)$ and $f(1,\ldots,1)$.
- The number of symmetry groups of f and their size.

For each *variable* x_i, the following signatures were used:

- The number of minterms of $f_{x_i=1}$.
- The number of minterms of $\exists x_i\ f$.
- The size of the symmetry group of x_i.

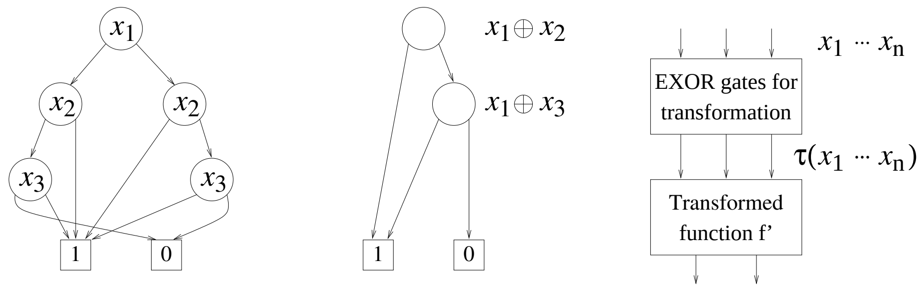
If the signatures do not allow to uniquely determine the variable correspondences, a complete enumeration of all variable correspondences has to be carried out. For Actel-1 cells, at most $8! = 40320$ cases have to be considered. In the experiments, it turned out that only very few different correspondences had to be tested to decide whether a function is in the table of computed results.

3.2 Overall Algorithm

Before the preprocessing and main optimization phase of the overall algorithm are discussed, the data structure used is briefly described.

3.2.1 Representation of the Netlist

To represent a netlist, a data structure is used that can represent cells with arbitrary functionality. The function of



(a) (b) (c)

Figure 3. Linear transformations for BDD minimization

each cell is described by a BDD. Additionally, each cell is marked by a label which denotes either a basic function ("MUX", "OR") or a complex function (in this case, the label is simply "BDD"). By this it is possible to decompose the function into blocks which can be represented by Actel-1 cells. Also merging of cells is possible. Note that this representation combines both structural and functional properties.

3.2.2 Preprocessing

Library-invariant optimization tools are used to find a good starting point. For this, three different starting points are created and then the best one is selected:

1. The first starting point is obtained using SIS script *rugged*. For the cells of the resulting netlist, local BDDs are constructed. These BDDs are then mapped to a MUX network by substituting each BDD node by a MUX cell.

2. The MUX network is used to construct a global BDD (if possible within given memory bounds; otherwise the MUX network is the only starting point). To minimize the BDD size, different variable reordering techniques are applied: for small functions (14 variables or less), an exact algorithm is used [8]. For larger functions, the following reordering techniques are applied (in that order): converging group sifting [21], converging sifting [22], a simple genetic algorithm [24], and finally converging group sifting again. Then the BDD is mapped to a MUX network.

3. If the global BDD could be constructed in the previous step, a window optimization algorithm using linear transformations [16, 10] is applied. Linear transformations replace input variables of the BDD by the EXOR of several variables, i.e. instead of assigning variables to each node of the BDD, the parity of a set of variables is assigned to each node. Figure 3 (b) shows an example of a linearly transformed BDD for the function given by its BDD in Figure 3 (a). To heuristically find a good linear transformation, the window optimization algorithm computes the optimal linear transformation for small windows of size 3. This results in a circuit like the one shown in Figure 3 (c). The circuit is mapped to MUX cells by replacing each EXOR gate by two MUX cells.

The starting point used for the following steps is the one that results in the least number of Actel-1 cells (using the mapper of Section 3.1). Compared to the following step, the preprocessing consumes only a small fraction of the runtime.

3.2.3 Local Optimization

Starting from a netlist obtained in the preprocessing step, local synthesis operations are applied. These modifications may result in cells consisting of BDDs with many nodes. The operations we used are the following:

1. Change polarity of one cell [20].

2. Merge one cell with all its fanin cells.
   In a first step, each cell is merged with its fanin cells one after another. To avoid local minima, in a second step also larger regions of fanin cells are merged, unless the BDDs for the cells become too large.

3. Merge one cell with all its fanout cells.

4. Merge two cells with similar inputs.
   As the resulting cell has two outputs, the shared BDD representation for the cell is mapped to MUX cells. By this, many small cells are created, which enables further merging steps.

After each modification the resulting number of Actel-1 cells is computed using the mapper of the previous section. (If the netlist contains BDD cells consisting of more than one MUX, these BDDs are replaced by a MUX netlist before mapping.) If the resulting number of Actel-1 cells is larger than the size before, then the last modification is undone and the previous netlist is restored. Therefore, the optimization is no longer technology independent. The opti-

```
ACTion (netlist) {

    /* preprocessing */
    use SIS script rugged to get a first starting point net;
    build BDD for netlist;
    if (successful) {
        net2 := BDD mapped to MUX netlist;
        replace net by net2 if better (in terms of Actel-1 cells);
        build linearly transformed BDD;
        net3 := transformed BDD mapped to MUX netlist;
        replace net by net3 if it is better (in terms of Actel-1 cells);
    }

    /* local optimization phase */
    repeat 3 times {
        repeat {
            foreach operation o of (change_polarity, merge_one_fanin,
                        merge_set_of_fanins, merge_fanout, merge_similar) {
                for (all gates of net) {
                    net' := perform local synthesis operation of type o;
                    if (net' is better in terms of Actel-1 cells) {
                        net := net';
                    }
                }
            }
        } until (no improvement or time exceeded)
        net := net mapped to a MUX netlist;
    }
}
```

Figure 4. Sketch of algorithm

mization steps are iterated until no further improvement can be obtained (or a user defined time limit is reached).

Then, the netlist is mapped to a multiplexor netlist, leading to tiny cells consisting of only one MUX gate. This is also done in each mapping step, but in this case the small cells are part of the new netlist, whereas the netlist remains unchanged for the mapping step.

If the size of the intermediate BDDs grows too much, the merging process can be stopped, as it is very unlikely (yet desirable) that the size can be reduced by further merging steps.

The local optimization cycle is carried out three times. A sketch of the overall algorithm is given in Figure 4. All in all, the flow of the algorithm is very simple and the good quality (see below) mainly results from the technology dependent optimizations during *local optimization* and the good BDD minimization techniques, which will be described in more detail in the following.

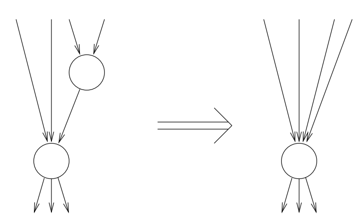


Figure 5. Merging two cells

3.3 Merging Cells

Merging two cells is the main operation of local optimization. Basically, this is done by building the BDD for the merged cell, i.e. a variable is substituted by another BDD (see Figure 5).

As the size of the resulting BDD directly influences the mapping result, a minimal representation is very desirable. Therefore, we apply different methods for variable reordering. If the number of variables is small enough, an exact

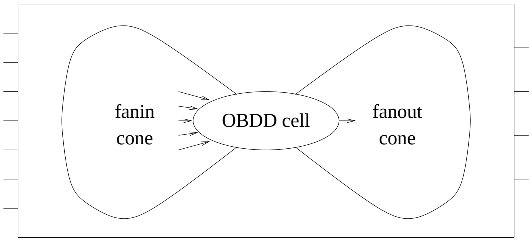


Figure 6. Region surrounding a cell to compute local don't cares

algorithm [8] is used. Otherwise, converging sifting [21] is called. By this procedure, in most cases a good variable ordering is found.

To further minimize the BDD sizes, we make use of local don't cares. The don't care set could be determined by constructing the characteristic function of the whole netlist. However, its BDD size is often too large to fit into the main memory. Therefore, the don't care set is computed only for a local region surrounding the cell (see Figure 6). The region is chosen in a way that the BDD sizes remain acceptable: It consists of a fanin cone of the cell of maximum depth 4 and a fanout cone of the cell of maximum depth 4 (unless the gate is primary output). The BDD is constructed for the region unless its size exceeds a given limit, i.e. four times the size of the netlist. (These numbers can be changed by the user according to the trade-off quality versus runtime.)

More formally, given the characteristic functions for the fanin cone, fanout cone and the whole region, a characteristic function χ_{cell} for the cell is constructed. This function contains the "care" information of the cell. Next, the characteristic function has to be converted into a don't care function. But as the cell is single-output, this can be done using universal quantification over the output variable: Let $x \in \mathbf{B}^n$ denote some assignment to the primary inputs. Then

$$\begin{aligned} dc(x) = 1 \quad &\Leftrightarrow \quad \chi_{cell}(x,0) = 1 \wedge \chi_{cell}(x,1) = 1 \\ &\Leftrightarrow \quad \forall o \in \mathbf{B} : \chi_{cell}(x,o) = 1, \end{aligned}$$

i.e. $dc = \forall o.\ \chi_{cell}$.

With the don't care information given as a Boolean function, standard minimization algorithms known from formal verification can be applied. We used a safe minimization algorithm [12] which guarantees that the size of the BDD does not increase by the minimization process. Note that it does not prevent the final result from becoming worse, due to the mapping step. However, in most cases some improvements can be obtained.

Using these sophisticated minimization techniques, the BDD sizes can often be reduced significantly.

4 Experimental Results

In this section we describe our experimental results. All experiments are carried out on a *SUN Ultra 1* workstation. The CUDD package [24] is used as underlying BDD package. A memory limit of 100 MB and a runtime limit of 15,000 CPU seconds is set in all experiments. The high runtime limit is used because the quality of the results was our main interest.

In a first series of experiments the ACTion algorithm is compared to several previously presented approaches. Results are given in Table 1. The name of the benchmark is given in the first column. The following columns report the number of Actel-1 cells needed for each circuit for the approaches *misII* [23], *Amap* [13], *Prosperine* [9], *mis-pga* (old, without last gasp, with last gasp) [18], respectively[1]. In the last column we give the results of the algorithm ACTion. The best result in each row is given in bold. As can be seen ACTion clearly outperforms all other approaches. Compared to the other mappers improvements of more than 50% can be observed (see *alu2*). The average improvement to *mis-pga* with last gasp is 20%.

In a second series of experiments we compare ACTion to the *ite_map* algorithm integrated in SIS (see Table 2). For *ite_map*, we first run script *rugged* and then apply *ite_map* (with two iterations). Both area and delay (using the unit delay model) values are given. The starting point of ACTion is given in column *start* (see Section 3.2.2): SIS means that the initial netlist was obtained by SIS, while for BDD it was obtained by building the global BDD. LT refers to linearly transformed BDDs. The runtime of ACTion in CPU seconds is given in the last column. Again, the best results are given in bold.

An improvement in size can be observed for most examples. It is important to notice that in both cases where *ite_map* outperforms ACTion the results differ by less than 10%. But in some cases the improvement of ACTion over *ite_map* is much larger (see e.g. *t481*), i.e. more than a factor of 30. This clearly demonstrates the robustness of our algorithm resulting from the different starting points.

Furthermore, using ACTion also the delay of the circuits is on average 26% smaller than using *ite_map*. In most cases where the area is smaller, the delay is also much smaller (see e.g *alu4* or *f51m*).

For larger instances, the algorithm tends to need much runtime. Therefore, we use an upper limit of 15,000 CPU seconds for the runtime. If the limit is exceeded, the best result found so far is reported. By this approach the user can trade off the runtime to spend and the quality of the result. Figure 7 shows an analysis of this trade-off for *c3540*. Starting with 534 cells, the area is much larger than the result of *ite_map* (462 cells). After less than 500 seconds, ACTion has also reached that value, but then further reduces

[1] The numbers are taken from [18].

Table 1. Comparison to previously presented algorithms

circuit	misII	Amap	Proserpine	mis-pga			ACTion
				old	new-NLG	new-LG	
5xp1	51	42	53	45	39	35	**30**
9sym	99	106	n/a	119	17	17	**14**
alu2	193	188	n/a	208	175	175	**83**
apex7	113	104	121	96	92	91	**72**
b9	64	81	n/a	65	63	60	**49**
bw	81	83	67	65	54	54	**52**
clip	57	60	73	51	43	43	**27**
e64	95	105	n/a	**94**	**94**	**94**	95
f51m	52	56	63	48	39	39	**23**
misex2	46	47	45	41	38	38	**35**
rd84	62	62	70	61	36	36	**28**
sao2	52	56	n/a	62	49	49	**43**
vg2	47	44	46	36	31	30	**29**
z4ml	20	20	n/a	16	18	14	**9**
total			—	1007	788	775	**589**
sub-total			538	443	372	366	**296**

the function to 414 cells. Also other parameters of the algorithm can be modified in order to trade off area versus runtime. Such parameters are e.g. the limitation of maximal intermediate BDD sizes or the size of the region the local don't care set is computed from.

In a third series of experiments, we compare our results to the results of the Actel Designer using *ACTmap VHDL Synthesis* [2]. Results are given in Table 3. It can be seen that ACTion always produces results which have smaller or equal size. In some cases, the reduction is even more than a factor of two (see e.g. *cordic* or *term1*).

Finally, we compare the "best ever" values that we found in literature to the results obtained by ACTion using a fixed parameter set and a runtime limit of 15,000 CPU seconds and to the "best ever" values we obtained for ACTion by varying the parameters. The numbers are given in Table 4. As can be seen, in many cases even ACTion with fixed parameters can further improve the best known values. For some benchmarks, like *alu2* or *rd84*, even large reductions can be observed. Using different parameter settings, these results can be further improved in many cases. However, these improvements are rather small and therefore underline the robustness of our approach.

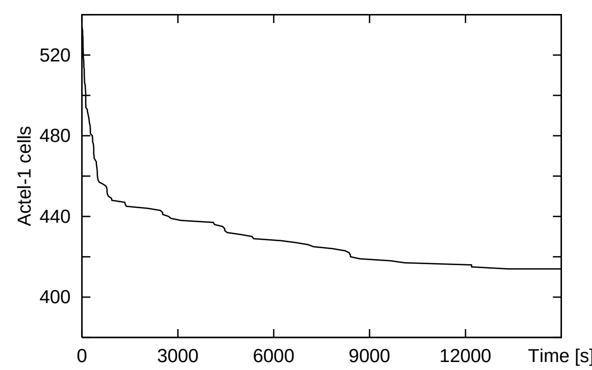


Figure 7. Trade-off time versus quality for *c3540*

5 Conclusions

We presented a new approach to combine technology mapping and logic synthesis. The networks are optimized such that they fit well on multiplexor based FPGAs. To minimize the underlying BDD data structure, we use variable reordering techniques and local don't care minimization. The user can specify parameters like a runtime limit or the maximal size of intermediate BDDs in order to trade off quality of the results versus runtime. The algorithm has been implemented as the program ACTion. A comparison to previously published approaches clearly demonstrates the efficiency of the approach.

References

[1] Actel. ACT™1 series FPGAs. Also available at http://www.actel.com/docs/databook97/section01/97s01d07.pdf, 1997.

Table 2. Comparison to *ite_map*

circuit	in	out	ite_map		ACTion				gain	
			area	delay	start	area	delay	CPU time	area	delay
5xp1	7	10	43	15	LT	**30**	**5**	63.0	-30.2 %	-66.7 %
9sym	9	1	84	9	LT	**14**	**6**	29.9	-83.3 %	-33.3 %
alu2	10	6	145	22	BDD	**83**	**7**	802.0	-42.8 %	-68.2 %
alu4	14	8	261	34	LT	**164**	**8**	3587.8	-37.2 %	-76.5 %
apex2	39	3	**105**	**13**	SIS	113	18	11053.8	7.6 %	38.5 %
apex3	54	50	654	27	BDD	**427**	**7**	15000.0	-34.7 %	-74.1 %
apex7	49	37	84	9	SIS	**72**	**6**	329.7	-14.3 %	-33.3 %
b9	41	21	53	8	SIS	**49**	**5**	127.0	-7.5 %	-37.5 %
bw	5	28	64	18	SIS	**52**	**13**	1286.3	-18.8 %	-27.8 %
c1355	41	32	166	**11**	SIS	**129**	**11**	15000.0	-22.3 %	0.0 %
c1908	33	25	162	**20**	SIS	**145**	**20**	14278.0	-10.5 %	0.0 %
c2670	233	140	210	**11**	SIS	**167**	15	6207.0	-20.5 %	36.4 %
c3540	50	22	462	25	SIS	**414**	**24**	15000.0	-10.4 %	-4.0 %
c432	36	7	92	31	SIS	**69**	**18**	8160.6	-25.0 %	-41.9 %
c5315	178	123	555	20	SIS	**457**	**16**	15000.0	-17.7 %	-20.0 %
c6288	32	32	1187	76	SIS	**969**	**59**	15000.0	-18.4 %	-22.4 %
c7552	207	108	651	**22**	SIS	**568**	23	15000.0	-12.7 %	4.5 %
clip	9	5	52	13	LT	**27**	**6**	75.4	-48.1 %	-53.8 %
cordic	23	2	24	8	BDD	**19**	**7**	26.9	-20.8 %	-12.5 %
dalu	75	16	407	19	LT	**279**	**13**	15000.0	-31.4 %	-31.6 %
des	256	245	1387	18	LT	**1024**	**11**	15000.0	-26.2 %	-38.9 %
duke2	22	29	190	17	BDD	**171**	**8**	4315.6	-10.0 %	-52.9 %
e64	65	65	**89**	**26**	SIS	95	32	1815.8	6.7 %	23.1 %
ex4	128	28	218	**9**	LT	**217**	**9**	4473.5	-0.5 %	0.0 %
f51m	8	8	34	16	LT	**23**	**5**	26.0	-32.4 %	-68.8 %
k2	45	45	433	**16**	SIS	**427**	22	15000.0	-1.4 %	37.5 %
misex2	25	18	41	8	SIS	**35**	**5**	102.5	-14.6 %	-37.5 %
rd84	8	4	57	11	LT	**28**	**6**	44.3	-50.9 %	-45.5 %
rot	135	107	257	**18**	SIS	**241**	**18**	15000.0	-6.2 %	0.0 %
sao2	10	4	53	17	LT	**43**	**7**	133.2	-18.9 %	-58.8 %
spla	16	46	262	**12**	SIS	**255**	16	15000.0	-2.7 %	33.3 %
t481	16	1	357	28	LT	**9**	**3**	12.4	-97.5 %	-89.3 %
vg2	25	8	33	10	SIS	**29**	**8**	58.9	-12.1 %	-20.0 %
z4ml	7	4	12	8	BDD	**9**	**4**	5.2	-25.0 %	-50.0 %
total			8884	625		**6853**	**441**			

[2] Actel. Actel DeskTOP CD. See http://www.actel.com/products/systems/desktop.html, 1999.

[3] S.D. Brown, R.J. Francis, J. Rose, and Z.G. Vranesic. *Field-Programmable Gate Arrays*. Kluwer Academic Publisher, 1992.

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

[5] P. Buch, A. Narayan, A.R. Newton, and A.L. Sangiovanni-Vincentelli. Logic synthesis for large pass transistor circuits. In *Int'l Conf. on CAD*, pages 663–670, 1997.

[6] R. Chaudhry, T.-H. Liu, A. Aziz, and J.L. Burns. Area-oriented synthesis for pass-transistor logic. In *Int'l Conf. on Comp. Design*, pages 160–167, 1998.

[7] G. De Micheli. *Synthesis and Optimization of Digital Circuits*. McGraw-Hill, Inc., 1994.

[8] R. Drechsler, N. Drechsler, and W. Günther. Fast exact minimization of BDDs. *IEEE Trans. on CAD*, 19(3):384–389, 2000.

[9] S. Ercolani and G.D. Micheli. Technology mapping for electrically programmable gate arrays. In *Design Automation Conf.*, pages 234–239, 1991.

Table 3. Comparison to Actel Designer

circuit	in	out	Designer	ACTion	gain
alu4	14	8	267	**164**	-38.6 %
apex6	135	99	359	**266**	-25.9 %
c8	28	18	47	**32**	-31.9 %
cc	21	20	28	**21**	-25.0 %
cm150a	21	1	11	**9**	-18.2 %
cm151a	12	2	13	**5**	-61.5 %
cm162a	14	5	19	**16**	-15.8 %
cm85a	11	3	**15**	**15**	0.0 %
cordic	23	2	40	**19**	-52.5 %
cu	14	11	25	**18**	-28.0 %
f51m	8	8	35	**23**	-34.3 %
i5	133	66	**66**	**66**	0.0 %
rot	135	107	279	**241**	-13.6 %
term1	34	10	88	**40**	-54.5 %
too_large	38	3	232	**111**	-52.2 %
x2	10	7	19	**18**	-5.3 %
x3	135	99	378	**260**	-31.2 %
z4ml	7	4	**9**	**9**	0.0 %

Table 4. ACTion vs. "best ever" results

circuit	literature best ever	ACTion fixed param	ACTion best ever
5xp1	35	30	**26**
9sym	**14**	**14**	**14**
alu2	111	83	**79**
apex7	80	72	**68**
b9	54	49	**47**
bw	54	52	**46**
c432	86	69	**67**
clip	36	27	**25**
cordic	21	19	**18**
e64	**89**	95	90
f51m	23	23	**21**
misex2	36	35	**34**
rd84	37	**28**	**28**
sao2	50	43	**39**
t481	10	**9**	**9**
vg2	30	**29**	**29**
z4ml	12	**9**	**9**
total	778	686	**649**

[10] W. Günther and R. Drechsler. BDD minimization by linear transformations. In *Advanced Computer Systems*, pages 525–532, 1998.

[11] G. Hachtel and F. Somenzi. *Logic Synthesis and Verification Algorithms*. Kluwer Academic Publisher, 1996.

[12] 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.

[13] K. Karplus. Amap: a technology mapper for selector-based field-programmable gate arrays. In *Design Automation Conf.*, pages 244–247, 1991.

[14] K. Karplus. ITEM: an if-then-else minimizer for logic synthesis. Technical report, University of California, Santa Cruz, 1992.

[15] K. Keutzer. Dagon: Technology binding and local optimization by dag matching. In *Design Automation Conf.*, pages 341–347, 1987.

[16] C. Meinel, F. Somenzi, and T. Theobald. Linear sifting of decision diagrams. In *Design Automation Conf.*, pages 202–207, 1997.

[17] J. Mohnke, P. Molitor, and S. Malik. Limits of using signatures for permutation independent Boolean comparison. In *ASP Design Automation Conf.*, pages 459–464, 1995.

[18] R. Murgai, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. An improved synthesis algorithm for multiplexor-based PGA's. In *Design Automation Conf.*, pages 380–386, 1992.

[19] R. Murgai, R.K. Brayton, and A.L. Sangiovanni-Vincentelli. *Logic Synthesis for Field-Programmable Gate Arrays*. Kluwer Academic Publisher, 1995.

[20] R. Murgai, Y. Nishizaki, N. Shenoy, R.K. Brayton, and A. Sangiovanni-Vincentelli. Logic synthesis for programmable gate arrays. In *Design Automation Conf.*, pages 620–625, 1990.

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

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

[23] E. Sentovich, K. Singh, L. Lavagno, Ch. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. Brayton, and A. Sangiovanni-Vincentelli. SIS: A system for sequential circuit synthesis. Technical report, University of Berkeley, 1992.

[24] F. Somenzi. *CUDD: CU Decision Diagram Package Release 2.3.0*. University of Colorado at Boulder, 1998.

[25] C. Yang, M. Ciesielski, and V. Singhal. BDS: a BDD-based logic optimization system. In *Design Automation Conf.*, 2000.