New Ways of Generating Large Realistic Benchmarks for Testing Synthesis Tools

Petr Fišer, Jan Schmidt
Faculty of Information Technology, Czech Technical University in Prague
fiserp@fit.cvut.cz, schmidt@fit.cvut.cz

Abstract

In this paper we propose several methods of generating large benchmark circuits for testing logic synthesis tools. The benchmarks are derived from real circuits, so that they are functionally equivalent to their origins. We introduce misleading and/or redundant structures into them, making the benchmark size blow up significantly, with respect to the original circuit. Such benchmarks can be advantageously used for testing logic synthesis tools; the aim is to discover whether particular synthesis processes are sensitive or immune to particular circuit transformations.

1 Introduction

Despite of all the late and recent developments in logic synthesis, current tools are not able to cope with newly emerging designs. Not only their ever-increasing size becomes a problem; there have been discovered small circuits, for which synthesis tools produce extremely bad results [1], with the size orders of magnitude higher than the optimum. Lately we have found a huge class of practical circuits for which synthesis severely fails as well [2], in both academic (SIS [3], ABC [4]) and commercial tools.

For these reasons, benchmarking becomes ever more important. Studying the behavior of synthesis tools on realistic benchmarks with a *known and properly defined origin* could disclose the nature of the problems.

The failure of synthesis tools is most apparent for originally small circuits, whose description was altered to make them large, or to introduce features the synthesis has problems with. In this paper we propose several methods of artificially "enlarging" circuits while preserving their function. Such benchmarks give us the benefit of knowing the upper bound of their complexity, as it is the original circuit size. Some of the methods are adjustable (in terms of specifying the resulting circuit size) to some extent, some are not. In general, any circuit may be processed by any of the proposed "enlarging" methods, sometimes without a guarantee of a circuit enlargement.

After synthesis, the resulting circuit size should not exceed the upper bound, regardless of the circuit alteration. However, in this paper we show that many synthesis tools, even commercial ones, fail to rediscover the original circuit structure. The size of the result is proportional to the size of the source circuit. Thorough evaluation and analysis of the behavior of the synthesis tools is out of the scope of the paper; here we just present methods of generating benchmark circuits.

2 Previous Work

Since there always has been a lack of publicly available practical (industrial) benchmark circuits, there have been many attempts for artificial benchmark generation. These benchmarks are targeted either to test logic synthesis processes in general (e.g., [5], [6], [7]), or to test partitioning and place&route algorithms in particular [8], [9]. In [7], the circuit functionality is considered as well, apart from the Rent's rule only, as in [8], [9]. However, the generated circuits are apparently much more redundant than their industrial origins. A way of generating realistic *clones* of real circuits is proposed in [6]. Here detailed characteristic (signature) is extracted from the *seed circuit,* i.e., a real circuit that serves as a base, from which the clone is created. A circuit with a structure very similar to the seed circuit is produced this way. Again, the function of the generated circuit is not known and not predictable; the generated circuit is random and may degrade to a simple constant in an extreme case.

A generic method of generating "difficult" benchmarks is proposed in [1]. Even though the benchmarks are artificially constructed and their function makes no sense, their optimum or upper bound sizes are known. For details see Subsection 2.1.

The first attempt to generate benchmark circuits *functionally equivalent* to real circuits was proposed in [5]. Here a set of 12 simple network transformation rules was determined. New benchmark circuits are generated by randomly applying these rules to the seed circuit. The authors have proven that any (functionally equivalent) network may be obtained from any network by a sequence of these rules. Perhaps due to chaotic application of the transformation rules, synthesis tools (like SIS [3]) did not have too big problems with the generated circuits presented in experimental results in [5].

2.1 LEKO and LEKU Benchmarks

Up to the knowledge of the authors, circuits substantially difficult for current synthesis processes were introduced in [1] for the first time, originally to test the performance of LUT (look-up table) mappers. These circuits were called LEKO (Logic Examples with Known Optimum) and LEKU (Logic Examples with Known Upper Bound) benchmarks. The LEKO benchmarks are constructed by replicating a relatively small circuit having n inputs and n outputs (n=5, 6), given as a Boolean network of two-input nodes. Optimum mapping into look-up tables with 4 inputs (4-LUTs) is known.

The LEKU benchmarks are constructed by collapsing the LEKO circuits into two-level sum-of-product (SOP) descriptions, followed by technology mapping. The circuit's original structure is thus completely obscured. Consequently, the circuit description (network) size grows up significantly, since the obtained SOP is very large. Technology mapping run upon the SOP just decomposes the huge AND and OR gates, producing a network of numerous 2-input NAND gates.

Since the LEKU circuits are functionally equivalent to the LEKO ones, their expected size is known. However, even better designs could be theoretically obtained by good synthesis and mapping. Therefore, the LEKO size is the upper bound imposed on the size of the LEKU circuits.

The process of the LEKU benchmarks construction is depicted in Figure 1. The 5-input G5 core circuit is used to construct the multiplied circuit G25. Two different decomposition procedures are used: the ABC *balance* command to obtain the LEKU-CB circuit and the SIS *tech_decomp* command to obtain LEKU-CD. For details see [1].

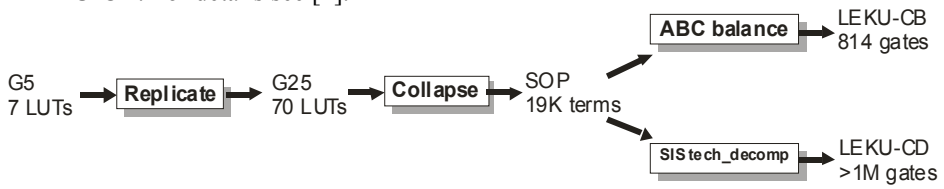

Figure 1: LEKU benchmarks construction

3 The Proposed New Benchmark Generation Methods

3.1 Realistic LEKU Benchmarks

The Cong & Minkovich's LEKU circuits [1] are basically constructed by intentionally introducing a bad structure into an artificially constructed circuit, resulting in a large circuit description. Other, real circuits can be processed in the same way (Figure 2). Collapsing a multi-level network into a two-level circuit completely destroys the circuit original structure, which is then very difficult to be recreated. Processing the circuit by a global BDD [10] does the same job. The size of the circuit usually significantly grows up, as in the LEKU case, although collapsing of some circuits may yield smaller representations. This is documented by Figure 3. Here 250 ISCAS'85 [14], ISCAS'89 [15] and IWLS'93 [16] benchmarks were first mapped into 2-input gates, then collapsed and mapped again (everything was performed by ABC [4]). The ratio between the original circuit size (in terms of 2-input gates) and the size after collapsing is indicated in the y-axis. In summary, 153 of the circuits were enlarged by collapsing.

In our experiments, collapsing of multi-level networks was done by SIS [3] or ABC [4]. The source circuit described in BLIF [11] is converted to a single multiple-output PLA description. Therefore, some logic may be shared between the outputs, in form of group terms. However, even though it may seem to be beneficial for the further synthesis, we have observed that sometimes the term sharing just introduces new misleading structures [12].

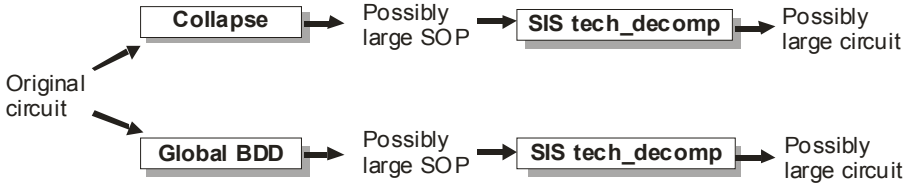

Figure 2: Realistic LEKU benchmarks construction

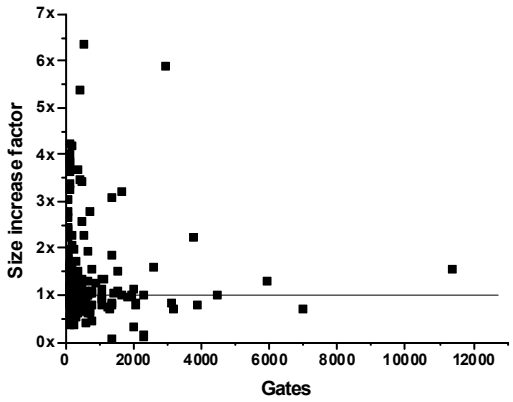

Figure 3: Size increase by collapsing

3.2 Parity Benchmark Circuits

Recently we have encountered a new class of hard-to-synthesize realistic circuits. These circuits are constructed by appending a XOR tree to the circuit's outputs, to obtain one parity bit [2] (Figure 4). Such circuits can be used as parity predictors [13]. The original circuit output values are not important here; only the resulting parity bit is of concern. I.e., single-output functions are produced as a result.

The upper bound of the area is the sum of the original circuit size and the size of the XOR tree. We have found that conventional synthesis tools are not able to minimize the circuit size efficiently, when the circuit is collapsed into a two-level SOP network (in a way described in Subsection 3.1) and resynthesized [12]. This process fully resembles the construction of the artificial LEKU benchmarks. The results of the resynthesis are spun between two extreme cases: at the "good" end, the circuit size is significantly reduced with respect to the upper bound, at the other end the size explodes [2]. The reason for the size explosion is the same as for the LEKU benchmarks – the obtained SOP is too large and the subsequent synthesis is not able to rediscover the original circuit structure. The need for XOR decomposition has been emphasized even more in these experiments. Tools not able to perform the XOR decomposition sometimes produced results 50-times larger than the upper bound.

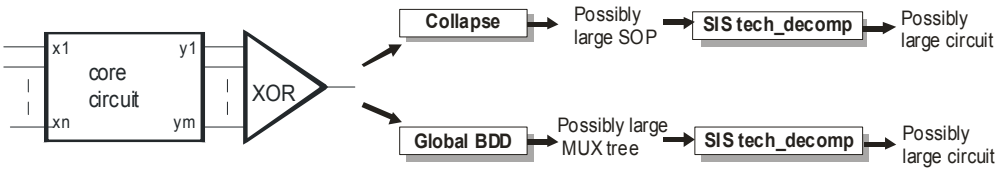

Figure 4: Parity benchmark circuits construction

3.3 Tautology and Near-Tautology Benchmarks

A different kind of artificially complex benchmarks can be created by generating large random SOPs. If the number of product terms (terms of higher dimensions, not only minterms) in the SOP exceeds a particular threshold, the function likely turns into tautology. Functions described by SOPs with the number of terms near this tautology threshold are usually very simple – they are "near-tautologies". Two-level minimization must be run in order to discover the true nature of functions described by these "big" SOPs. However, ABC and commercial tools do not do so for scalability and other reasons. If such a SOP (in form of a PLA or mapped into technology) is submitted to the synthesis, huge circuits are produced.

3.4 Partial Collapsing

The collapsing process in the above-mentioned methods produces results depending solely on the source circuit; the gate count of the result cannot be adjusted. Different collapsing tools e.g., ABC [4] and SIS [3], however, usually produce slightly different results. Results from BDDs processing can be influenced by different variable orderings [10]. Unfortunately, experiments show that the resulting decomposed network is either "too small" (the collapsing process is beneficial for the source circuit), or "too large" (the source circuit is difficult to be collapsed). The same holds for the parity circuits, since collapsing is involved here as well.

A straightforward way to adjust the size of *any* network is *partial collapsing*. Only a part of the circuit is extracted (subcircuit), collapsed, decomposed into 2-input gates, and returned back into the network. The basic algorithm is shown in pseudo-code in Figure 5. The parameters are the required boundaries of the resulting circuit size. First, the initial size of the subcircuit to be extracted is set (say 4 gates). After the circuit part extraction and collapsing the resulting network is checked for validity, in terms of the required size. If its size is too small, the process is repeated with increased size of the circuit part. For details on the part selection process (Extract_Part) see [17].

```
Partial_Collapse( Network N, int min, int max ) {
    size = initial_size;
    do {
        (P, N_R)  = Extract_Part(N, size);
        P' = Collapse&Decompose(W);
        N' = N_R ∪ W';
        if ( |N'| ≥ min && |N'| ≤ max ) { N = N'; break; }
        else if ( |N'| < min ) size++;
    } while (true);
    return N;
}
```

Figure 5: The benchmark generation algorithm

Partial collapsing of the circuit is meant to produce circuits larger than the originals and smaller than the completely collapsed circuits. However, circuits even larger than completely collapsed circuits have been sometimes generated, possibly for several reasons. The number of inputs of the extracted circuit part may be higher than the number of inputs of the entire circuit. Thus, the collapsing procedure could be sometimes more demanding. Next, even though the source circuit function is "simple" and can be described by a few product terms, the extracted part function may be more complex. As an example, the circuit size obtained by running the partial collapsing on the c432 [14] circuit, as a function the collapsed subcircuit size is shown in Figure 6. No size control was applied here (minima and maxima in Figure 5). It can be seen that even though the completely collapsed circuit has approx. 2,000 gates, there is a circuit having more than 10,000 gates. Note that the original c432 has 145 gates, which corresponds to the 0-sized collapsed part in the graph.

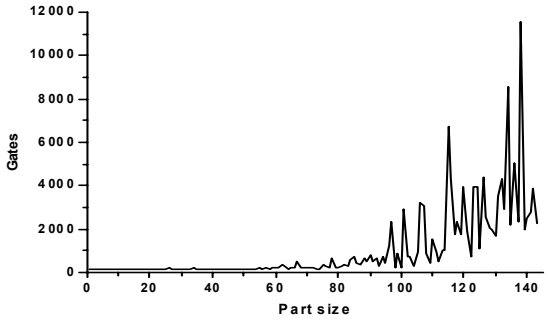

Figure 6: Partially collapsed circuit sizes (c432)

A more striking example is the behavior of the tautology benchmarks (Subsection 3.3) processed by the partial collapsing. Even though the fully collapsed circuit is a constant, partially collapsed circuits can be much larger than the original. This is illustrated by Figure 7a, b (the y-axis is trimmed in latter one, to show the details). When increasing the collapsed subcircuit size up to some limit, the resulting circuit size slowly increases. When enlarging the subcircuit size, a slow decrease of the size is

observed. In the area where the collapsed subcircuit size nears the original circuit size, two extreme cases occur: either the tautology is discovered (hence the resulting circuit size shrinks to 0), or extremely large circuits are produced.

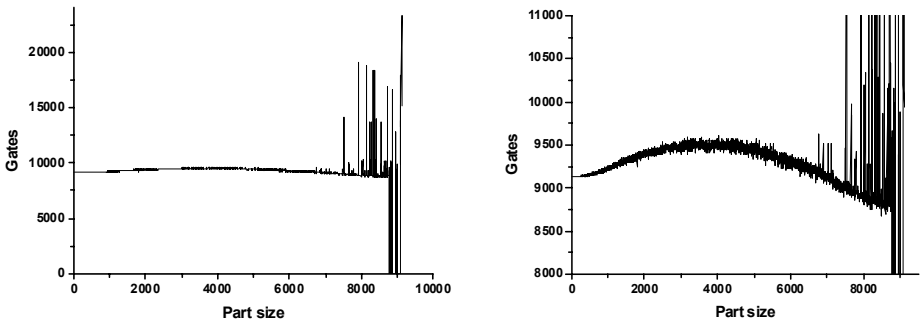

Figure 7a, b: Partially collapsed tautology

3.5 Replicating Shared Logic

Another way to enlarge a network is to duplicate a part of the logic that is originally shared. A branching signal is identified and the transitive fan-in of this signal is duplicated, to a given depth (or up to primary inputs). The branching is then split and each branch is connected to one of the two replicas. An example is shown in Figure 8, for the c17 ISCAS'85 [14] circuit. The net G16, together with its transitive fan-in (gates G16 and G11), is duplicated here. Two parameters drive the generation: the number of duplicated branches and the duplication depth.

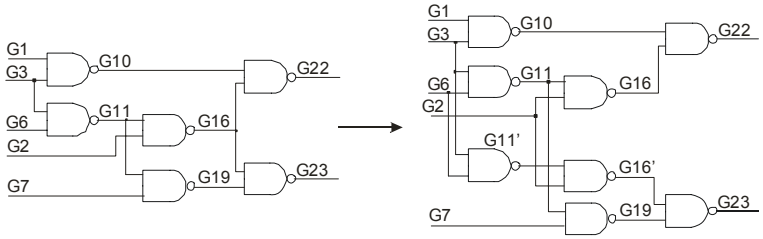

Figure 8: Duplicating shared logic example

4 Experimental Results

In this section we present some representative benchmarks generated by methods described in Section 3, with results obtained by ABC and two commercial synthesis tools.

All the data are summarized in Table 1 (at the end of this document). First, the original (seed) circuit name is shown, with numbers of its inputs ("*i*") and outputs ("*o*"). Then the process by which the circuit was modified is described and the number of 2-input gates obtained after mapping by ABC "*map*" command is given.

The original circuits are indicated by the shadowed rows. Their synthesis results serve us as upper bounds of the complexity; benchmarks with equal names are functionally equivalent. Each circuit was processed by a global BDD and by SIS and ABC collapsing (Subsection 3.1). Next, partial collapsing was applied (Subsection 3.4), and finally duplicities were introduced according to Subsection 3.5. The latter two processes demonstrate possibilities of adjusting the resulting gate counts. In case of the partial collapsing, the "size" parameter indicates the size of the collapsed subcircuit, in gates. The "depth" value corresponds to the replication depth parameter (Subsection 3.5).

We present results of five representative circuits: the c432 [14] (whose size significantly increases by collapsing), c880 [14] (whose size significantly increases by processing by a global BDD and by collapsing as well), s1238 [15] and b4 [16] with parity and one randomly generated big tautologic PLA.

The right-hand part of the table contains the results of four different synthesis processes. In all cases, synthesis and mapping into 4-input LUTs has been performed. First, the circuits were synthesized by ABC [4], by using a sequence of commands suggested by ABC authors: "*choice; fpga; lutpack*". The "*choice*" script performs several different steps of resynthesis, hence a "good" synthesis effort is ensured. Next, results of two commercial tools are presented ("*#1*", "*#2*"). For some circuits commercial tools failed to produce any result at all (the "N/A" entries).

For these examples we can observe how well the partial collapsing and introduction of duplicities adjusts the circuit size. We have processed numerous different circuits and the behavior of these adjustable benchmark generation methods mostly fully resembles the presented representatives. However, there are limitations based on the seed circuit function and structure. For example, the c880 circuit size cannot be much increased by introducing duplicities, because of its low depth (only 14 levels). Partial collapsing also needs not be working universally for any circuit; there needs not exist a subcircuit increasing the network size, when collapsed.

Next, a sorry fact may be observed: the synthesized LUT numbers grow with growing source circuit sizes, for all studied synthesis tools. Only ABC is completely immune to the introduction of duplicities, commercial tools are immune only partially. No tool is able to cope up with sizes of the collapsed circuits. This is illustrated by the two graphs shown in Figure 9. The x-axis describes gate counts of the original circuits (c432, s1238 with parity), the y-axis gives the resulting LUT counts. Data from all the processes Table 1. are present in the graph. Two different behaviors may be observed: the bottom data points represent benchmarks generated by introducing duplicities, while the almost linearly growing dependencies depict the collapsing (and global BDD) processes.

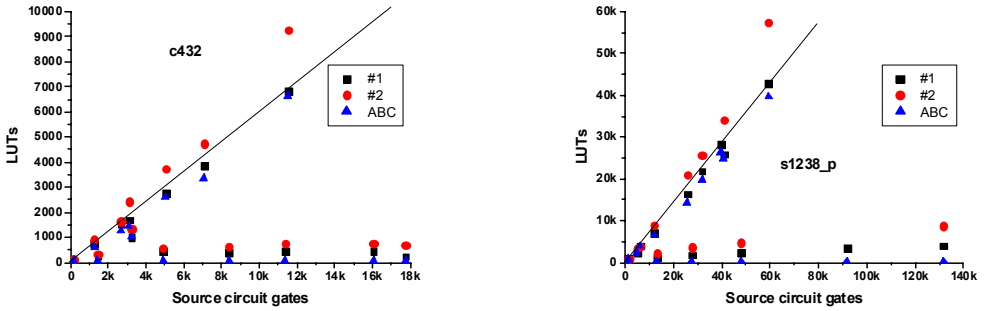

Figure 9: Synthesis results for c432 and s1238_p circuits

We have also tried to map these circuits into 6-LUTs, hoping for better performance of commercial tools when applied to modern FPGAs designs. However, no different behavior was observed. As an example, we show the data and graph for the c432 circuit in Table 2 and Figure 10.

Process	*Gates*	*ABC*	*#1*	*#2*
original	145	73	58	82
global BDD	2,017	559	545	589
ABC collapse	2,658	873	1,138	1,104
SIS collapse	7,075	2,354	2,869	3,289
Partial collapse, size 98	1,247	582	583	615
Partial collapse, size 109	3,077	1,086	1,264	1,682
Partial collapse, size 138	5,026	1,909	2,116	2,514
Partial collapse, size 140	11,531	4,703	5,013	6,205
10k duplicities, depth 1	1,428	73	199	249
10k duplicities, depth 2	4,905	73	378	424
10k duplicities, depth 3	8,389	73	327	468
10k duplicities, depth 4	11,349	73	362	525
10k duplicities, depth 5	16,040	73	397	565
10k duplicities, inf. depth	17,749	73	212	488

Table 2: c432 mapped to 6-LUTs

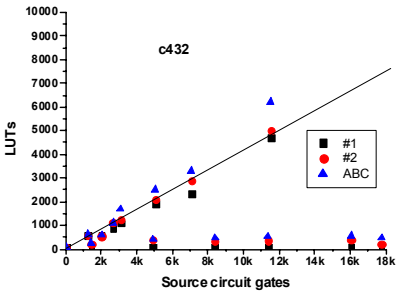

Figure 10: c432 mapped to 6-LUTs

5 Conclusions

We have presented several novel methods of generating artificially large benchmark circuits, which are functionally equivalent to their origins. An upper bound of their complexity is known and it is relatively small. Different adjustable processes of increasing the circuit size have been proposed. This enables us to test efficiency, scalability and capabilities of synthesis tools. Even though the benchmark circuits are generated by artificial and intentional modifications of networks, it cannot be guaranteed that such “bad” networks cannot be produced by, e.g., HDL synthesis.

All the proposed benchmark generation methods may be applied to any seed circuit, however the size increase is not always guaranteed. Two of the methods are adjustable to some extent, in terms of the required number of the produced benchmark circuit gates.

Experiments have shown that both academic and commercial tools are not able to perform satisfactorily. The obtained synthesis result sizes increase with increasing size of the source circuits, even though equal results should be produced. This indicates that gate-level logic synthesis crucially lacks in many aspects and there is still an open field for improvements.

Acknowledgement

This research has been supported by MSMT under research program MSM6840770014 and by the grant of the Czech Grant Agency GA102/09/1668.

References

[1] J. Cong and K. Minkovich: Optimality study of logic synthesis for LUT-based FPGAs, IEEE Trans. on CAD, vol. 26, pp. 230–239, Feb. 2007.

[2] P. Fišer and J. Schmidt, J: Small but Nasty Logic Synthesis Examples, Proc. 8th Int. Workshop on Boolean Problems (IWSBP'08), Freiberg, Germany, 18.-19.9.2008, pp. 183-190.

[3] E.M. Sentovich et al. SIS: A System for Sequential Circuit Synthesis, Electronics Research Laboratory Memorandum No. UCB/ERL M92/41, University of California, Berkeley, CA 94720, 1992.

[4] Berkeley Logic Synthesis and Verification Group: ABC: A System for Sequential Synthesis and Verification. [Online]. Available: http://www.eecs.berkeley.edu/ alanmi/abc/.

[5] K. Iwama and K. Hino: Random generation of test instances for logic optimizers, in Proc. 31st Design Automation Conf. (DAC), 1994, pp. 430–434.

[6] M. D. Hutton, J. P. Grossman, J. S. Rose, and D. G. Corneil: Characterization and parameterized random generation of combinational benchmark circuits, IEEE Trans. Computer-Aided Design, vol. 17, pp., 955–996, Oct. 1998.

[7] D. Stroobandt, P. Verplaetse, and J. Van Campenhout: Generating synthetic benchmark circuits for evaluating CAD tools, IEEE Trans. Computer-Aided Design, vol. 19, pp. 1011–1022, 2000.

[8] J. Darnauer and W. Dai: A method for generating random circuits and its application to routability measurement, in Proc. 4th ACM/SIGDA Int. Symp. FPGA's, Feb. 1996, pp. 66–72.

[9] D. Stroobandt, J. Depreitere, and J. Van Campenhout: Generating new benchmark designs using a multiterminal net model, Integration, the VLSI J., vol. 27, no. 2, pp. 113–129, 1999.

[10] S. B. Akers: Binary decision diagrams, IEEE Trans. on Computers, Vol. C-27. No. 6, June 1978, pp. 509-516.

[11] Berkeley Logic Interchange Format (BLIF), University of California, Brekeley, 2005

[12] P. Fišer, J. Schmidt: The Observed Role of Structure in Logic Synthesis Examples, Proc. 18th of International Workshop on Logic and Synthesis 2009 (IWLS'09), Berkeley, California (USA), 31.7. - 2.8.2009, pp. 210-213

[13] P. Kubalík, P. Fišer and H. Kubátová, Fault Tolerant System DesignMethod Based on Self-Checking Circuits, Proc. 12th International On-Line Testing Symposium 2006 (IOLTS'06), Lake of Como, Italy, July 10-12, 2006, pp. 185-186

[14] F. Brglez and H. Fujiwara: A Neutral Netlist of 10 Combinational Benchmark Circuits and a Target Translator in Fortan, Proc. of International Symposium on Circuits and Systems, pp. 663-698, 1985.

[15] F. Brglez, D. Bryan and K. Kozminski: Combinational Profiles of Sequential Benchmark Circuits, Proc. of International Symposium of Circuits and Systems, pp. 1929-1934, 1989.

[16] K. McElvain: IWLS'93 Benchmark Set: Version 4.O, distributed as part of the IWLS'93 benchmark distribution.

[17] P. Fišer, J. Schmidt: It Is Better to Run Iterative Resynthesis on Parts of the Circuit, Proc. 19th of International Workshop on Logic and Synthesis 2010 (IWLS'10), Irvine, California (USA), 18.-20.6.2010, pp. 17-24

Table 1: Detailed experimental results

Benchmark circuit					*Synthesis into 4-LUTs*		
Bench	*i*	*o*	*Process*	*Gates*	*ABC*	*#1*	*#2*
c432 [14]	36	7	original	145	84	77	118
c432 [14]	36	7	global BDD	2,017	1,031	1,023	1,333
c432 [14]	36	7	ABC collapse	2,658	1,246	1,548	1,648
c432 [14]	36	7	SIS collapse	7,075	3,361	3,872	4,738
c432 [14]	36	7	Partial collapse, size 98	1,247	626	782	916
c432 [14]	36	7	Partial collapse, size 109	3,077	1,445	1,699	2,422
c432 [14]	36	7	Partial collapse, size 138	5,026	2,598	2,761	3,727
c432 [14]	36	7	Partial collapse, size 140	11,531	6,647	6,844	9,255
c432 [14]	36	7	10k duplicities, depth 1	1,428	84	244	333
c432 [14]	36	7	10k duplicities, depth 2	4,905	84	447	586
c432 [14]	36	7	10k duplicities, depth 3	8,389	84	396	637
c432 [14]	36	7	10k duplicities, depth 4	11,349	84	452	739
c432 [14]	36	7	10k duplicities, depth 5	16,040	84	472	771
c432 [14]	36	7	10k duplicities, inf. depth	17,749	84	249	684
c880 [14]	60	26	original	208	113	110	122
c880 [14]	60	26	global BDD	407,098	93,190	174,983	N/A
c880 [14]	60	26	ABC collapse	13,727	7,437	8,109	9,460
c880 [14]	60	26	SIS collapse	30,015	19,787	20,487	28,017
c880 [14]	60	26	Partial collapse, size 129	1,008	485	601	597
c880 [14]	60	26	Partial collapse, size 171	5,034	2950	2,394	3,769
c880 [14]	60	26	Partial collapse, size 201	10,423	6224	5,010	7,887
c880 [14]	60	26	10k duplicities, depth 1	1,258	113	97	262
c880 [14]	60	26	10k duplicities, depth 3	1,828	113	109	317
c880 [14]	60	26	10k duplicities, inf. depth	2,962	113	99	140
s1238_p [15]	32	1	original	493	229	241	263
s1238_p [15]	32	1	global BDD	6,282	3,849	4,055	3,839
s1238_p [15]	32	1	ABC collapse	31,839	19,741	21,875	25,793
s1238_p [15]	32	1	SIS collapse	39,636	26,313	28,254	N/A
s1238_p [15]	32	1	Partial collapse, size 150	1,365	750	792	895
s1238_p [15]	32	1	Partial collapse, size 219	4,876	2,263	2,298	3,586
s1238_p [15]	32	1	Partial collapse, size 309	12,001	6,504	7,120	8,991
s1238_p [15]	32	1	Partial collapse, size 376	25,756	14,288	16,425	20,029
s1238_p [15]	32	1	Partial collapse, size 454	40,845	24,913	25,906	34,180
s1238_p [15]	32	1	Partial collapse, size 477	59,330	39,595	42,756	57,306
s1238_p [15]	32	1	5k duplicities, depth 1	13,171	229	1,125	2,218
s1238_p [15]	32	1	5k duplicities, depth 2	27,480	229	1,828	3,640
s1238_p [15]	32	1	5k duplicities, depth 3	47,898	229	2,356	4,760
s1238_p [15]	32	1	5k duplicities, depth 4	91,889	229	3,582	N/A
s1238_p [15]	32	1	5k duplicities, depth 5	131,173	229	4,014	8,760
s1238_p [15]	32	1	5k duplicities, inf. depth	494,891	229	4,417	N/A
b4_p [16]	33	1	original	267	110	108	116
b4_p [16]	33	1	BDD	16,963	6,347	6,099	4,285
b4_p [16]	33	1	ABC collapse	1,405	730	841	884
b4_p [16]	33	1	SIS collapse	4,087	2,036	2,422	1,627
b4_p [16]	33	1	Partial collapse, size 108	1,097	450	566	677
b4_p [16]	33	1	Partial collapse, size 219	4,664	2,328	2,439	3,543
b4_p [16]	33	1	Partial collapse, size 167	7,960	2,953	3,041	5,550
b4_p [16]	33	1	10k duplicities, depth 1	2,085	110	169	126
b4_p [16]	33	1	10k duplicities, depth 2	9,259	110	195	229
b4_p [16]	33	1	10k duplicities, depth 3	21,029	110	212	170
b4_p [16]	33	1	10k duplicities, depth 4	26,019	110	411	134
b4_p [16]	33	1	10k duplicities, depth 5	39,842	110	635	136
b4_p [16]	33	1	10k duplicities, inf. depth	81,078	110	575	246
tautology	25	1	tautology	9,130	6,798	8,425	8,528
tautology	25	1	Partial collapse, size 7521	14,082	10,542	12,371	14,945
tautology	25	1	Partial collapse, size 7932	19,028	13,839	15,952	20,398
tautology	25	1	Partial collapse, size 9121	24,249	17,364	19,775	27,113
tautology	25	1	500 duplicities, inf. depth	10,397	6,798	8,362	8,521
tautology	25	1	10k duplicities, inf. depth	13,986	6,813	8,441	8,600