The Case for a Balanced Decomposition Process

Petr Fišer, Jan Schmidt
Czech Technical University in Prague
email: fiserp@fel.cvut.cz, schmidt@fel.cvut.cz

***Abstract*—We present experiments with synthesis tools using examples which are currently believed to be very hard, namely the LEKU examples by Cong and Minkovich and parity examples of our construction. In both cases, we found a way to produce reasonable results with existing tools. We identify the abilities that are crucial for achieving such results, and also generalize them to avoid similar cases of poor performance in future tools.**

I. Introduction

Logic synthesis is believed to be a matured process, giving results reasonably close to optimum. Yet, there are still circuits which are very hard for any synthesis process. Cong and Minkovich [1] published a methods for the construction of combinational circuits with known optimal implementation (LEKO) or with known upper bound (LEKU). Here we study the latter ones, as the gap between the upper bound and obtained results are the largest. Our *parity examples* [2] are another case of difficult circuits. Synthesis tools give results an order or two bigger than a known upper bound.

We investigated the reasons of the observed poor performance experimentally. We succeeded in finding tools and procedures that give satisfactory (i.e. not orders of magnitude worse) results, experimented further to obtain clues what makes those tools and procedures successful.

First we describe our experimental methods. Secondly, experiments with both sets of examples are described together with the results obtained. Finally, we interpret the results and give requirements for future tools.

II. Experimental methods

We used the following tools for decomposition: SIS [3], ABC [4], BDS [5], and BiDecomp [6] chiefly because we can understand, control and modify their internals.

The size of the resulting circuit was measured as the number of 4-input LUTs, similarly to [8] and for similar reasons, namely to obtain a unified measure for decomposition algorithms working with different circuit representations.

Decomposition by SIS was done by running the *script.rugged* (a part of the SIS distribution), followed by *tech_decomp -o 2*, to produce a network of two-input gates. Mapping by SIS was performed by running the script proposed in [3] for LUT synthesis, modified for mapping into 4-LUTs. ABC decomposition was performed by running the *resyn2* script; ABC *dsd* was also tested in some experiments. Mapping in ABC was done by the *fpga* command, or by repeatedly running the command sequence *strash, balance, fpga, cleanup*. In the case of BDS running on some examples, we had to apply a heuristic measure preventing the program from generating an infinite decomposition tree.

III. LEKU Examples

These examples were primarily designed to test the technology mapping [1]. Many of mapping tools used in the original experiments, however, use resynthesis at the beginning of the mapping phase.

LEKU-CB examples are obtained using SIS by collapsing and balancing the circuit, while LEKU-CD examples result from collapsing and technology decomposition. For detailed description, please refer to [1].

LEKU-CD and LEKU-CB examples based on the G25 circuit were used. G25 is known to have the upper bound of 70 4-LUTs. The results from selected experiments are presented in Table I. It can be seen that ABC managed to synthesize G25 collapsed by ABC and MVSIS quite well, synthesis of G25 collapsed by SIS yielded much worse results (900 LUTs after resynthesis). However, when BDS was run prior to the ABC LUT mapping, equal results of 113 LUTs were obtained in all cases. Even better results were obtained by running the *dsd* ABC command prior to the LUT mapping. The ABC *dsd* command generated the same result in all cases. We did not succeed in collapsing CD(G25) by MVSIS or SIS.

IV. Parity Examples

Each parity example consists of an arbitrary circuit – the core circuit – and a parity tree summing up all outputs of the core circuit. The description of the entire circuit in SOP form is the example. An upper bound on size can be obtained as the sum of the core circuit size and the parity tree size when synthesized separately.

Admittedly, this is not a very tight upper bound. Yet the examples are easy to construct, and any synthesis process with results outside such a loose upper bound is certainly worth investigation.

A. Synthesis Experiments

We used MCNC benchmarks [11] and IWLS benchmarks [12] as core circuits. *alu1TWRG* is a doubled and regularized version of *alu1*. 'Nastiness' is the ratio of actual result to

Table I
SYNTHESIS RESULTS OBTAINED FOR DIFFERENT FORMS OF AN INPUT

Input	SOP size [terms]	Tools combination	Result size [LUTs]
G25	–	ABC *fpga*	80
		ABC *resyn2* + *fpga*	72
		ABC *dsd* + *fpga*	102
		SIS	136
G25 collapsed by ABC	18905	ABC *fpga*	280
		ABC *resyn2* + *fpga*	200
		ABC *dsd* + *fpga*	102
		BDS + ABC *fpga*	113
		BiDecomp + ABC *fpga*	188
		SIS	failed
G25 collapsed by MVSIS	121500	ABC *fpga*	232
		ABC *resyn2* + *fpga*	170
		ABC *dsd* + *fpga*	115
		BDS + ABC *fpga*	113
		SIS	failed
G25 collapsed by SIS	201816	ABC *fpga*	1140
		ABC *resyn2* + *fpga*	900
		ABC *dsd* + *fpga*	102
		BDS + ABC *fpga*	113
		SIS	failed
G25 collapsed by ABC + Espresso	14619	ABC *fpga*	1725
		ABC *resyn2* + *fpga*	1520
		ABC *dsd* + *fpga*	102
		BDS + ABC *fpga*	113
		SIS	failed
CB(G25)	–	ABC *fpga*	286
		ABC *resyn2* + *fpga*	189
		ABC *dsd* + *fpga*	102
		BDS + ABC *fpga*	245
		BiDecomp + ABC *fpga*	180
		SIS	424
CB(G25) collapsed by ABC	18905	ABC *fpga*	280
		ABC *resyn2* + *fpga*	200
		ABC *dsd* + *fpga*	102
		BDS + ABC *fpga*	113
		BiDecomp + ABC *fpga*	188
		SIS	failed
CD(G25)	–	ABC *fpga*	40k
		ABC *resyn2* + *fpga*	27k
		ABC *dsd* + *fpga*	102
		BDS	failed
		BiDecomp	failed
		SIS	failed
CD(G25) collapsed by ABC	18509	ABC *fpga*	280
		ABC *resyn2* + *fpga*	200
		ABC *dsd* + *fpga*	102
		BDS + ABC *fpga*	113
		BiDecomp + ABC *fpga*	188
		SIS	failed

the upper bound. Nastiness greater than 1 indicates that improvements in the synthesis process are possible.

We observed the performance of decomposition by SIS, ABC, BDS, and BiDecomp combined with both SIS and ABC mapping, using resynthesis in the mapping step in all cases. Because of previous hints at circuit symmetry as one source of difficulty, we also measured the symmetry using ABC's command *print_symm*. In Figures 1 to 3, we can see that BDS and BiDecomp produce better and often acceptable results, however, we can also see examples that are difficult for all tools. The data on the nastier examples are in Table II. Apparently, there is no direct relationship between symmetry and nastiness.

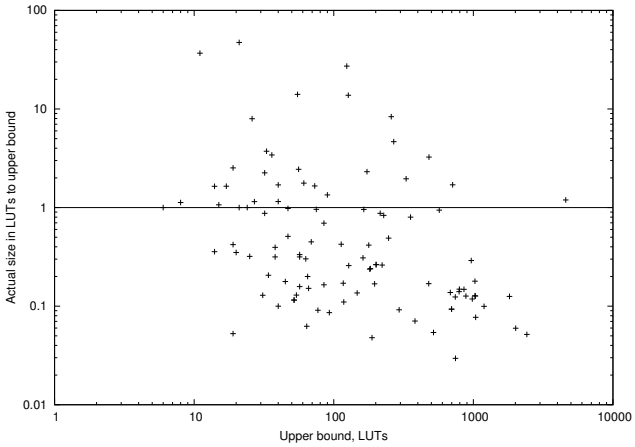

Figure 1. Parity examples decomposed and mapped by ABC

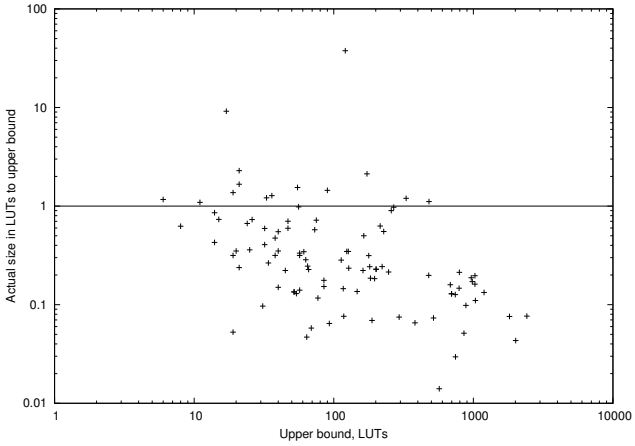

Figure 2. Parity examples decomposed by BDS and mapped by ABC

Table II
THE NASTINESS AND SYMMETRY OF SELECTED PARITY EXAMPLES FOR THE MAIN TOOL SETS

Core Circuit	Upper Bound	SIS	ABC	BDS +ABC	BiDec. +ABC	Sym-metry
alu1	11	38.40	36.80	1.09	1.64	0
alu2	36	6.30	3.40	1.28	2.31	0
alu4	481	3.25	3.25	1.11	2.33	0
ex7	55	21.10	14.05	1.55	5.60	0
misex3c	258	8.23	8.36	0.90	8.23	0
signet	121	–	160.40	37.70	–	0.1%
alu1TWRG	21	–	361.40	1.67	1.76	0
e64		0.03	0.03	0.03	0.03	97%
rd84		0.64	2.54	1.37	1.68	100%

B. Composition of Partial Algorithms

To get a better insight into the role played by decomposition algorithms comprising a single tool, we modified BDS to switch the following algorithms on and off on command [5]:

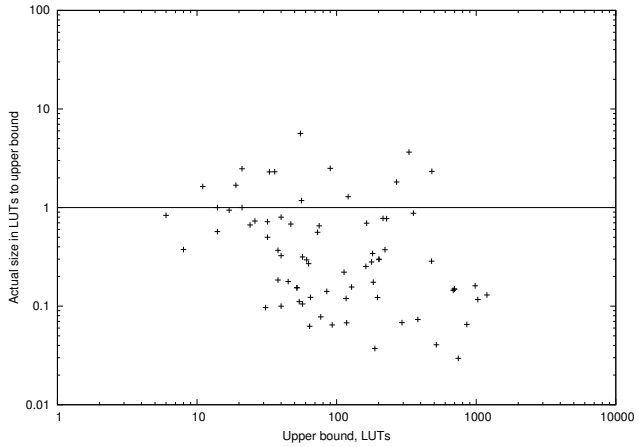

Figure 3. Parity examples decomposed by BiDecomp and mapped by ABC

- The decomposition using generalized dominators (GD), a conjunctive/disjunctive decomposition.
- The branch decomposition (B HardCore, BHC), a MUX evtl. XOR decomposition.
- The XOR decomposition (X HardCore, XHC).

All these algorithms expect the easy cases to have been already sorted out by the simple-dominator decomposition, which therefore cannot be switched off. The relative change

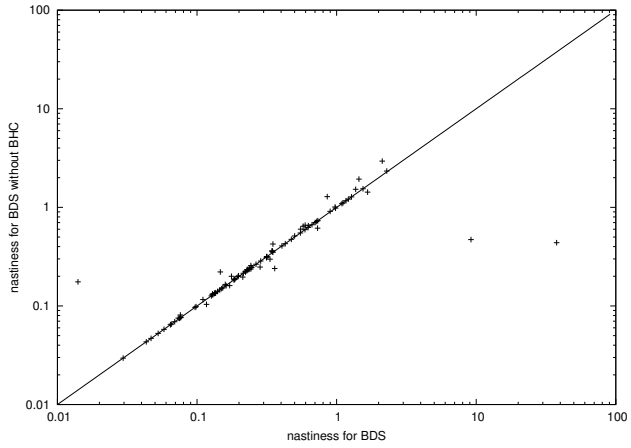

Figure 4. The influence of not using B HardCore (BHC) decomposition in BDS

in nastiness over the entire set of benchmarks are presented in Figures 4 and 5.

For *alu1* and *alu1TWRG*, the ability to rediscover the XOR tree is crucial, and the tree is actually present in the result. For other examples, partial algorithms other than XOR bi-decomposition seem to be principal (cf. BHC in *apex2*). For some examples (*apex2*, *ex7*) the absence of an algorithm (GD) can direct the process to a better result. All

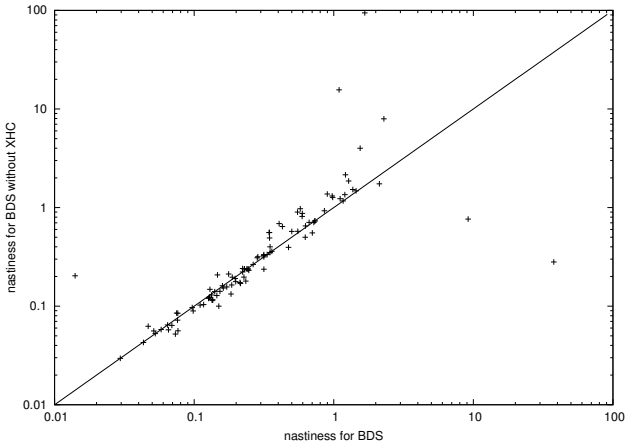

Figure 5. The influence of not using XOR HardCore (XHC) decomposition in BDS

Table III
THE INFLUENCE OF DISABLING PARTIAL ALGORITHMS IN BDS ON NASTINESS

Core Circuit	BDS complete	GD off	BHC off	XHC off
alu1	1.09	1.09	1.09	15.64
alu2	1.28	1.28	1.28	1.86
alu4	1.11	–	1.11	1.23
apex2	2.12	1.24	2.94	1.74
ex7	1.55	0.76	1.55	4.00
misex3c	0.90	1.03	0.91	1.37
signet	37.70	–	39.73	38.33
alu1TWRG	1.67	1.48	1.43	94.30

these influences, however, are not nearly as dramatic as the absence of XOR decomposition in *alu1* or *ex7*.

From all examples tested, *signet* seems to be singular as no tested process is able to obtain a reasonable solution. Besides the tools listed in Table II and described in Section II, ABC scripts based on *fraig* and *resyn* were tried, with no substantial improvement. Also, combination of BDS with ABC resynthesis gave results similar to BDS alone.

V. INTERPRETATION AND DISCUSSION

For all examples (the parity example based on *signet* being a notable exception) we found a way to obtain acceptable solutions using existing tools. The two sets of experiments also discovered two distinct reasons of subpar performance, and therefore distinct requirements to future tools.

A. Abandoning the Original Structure

The results in Section III can be explained as follows. The original G25 circuit description has a 'good' structure, on which the synthesis can build. CD(G25), on the other hand, has a misleading structure produced by *tech_decomp*. The tools, working incrementally from it, could not achieve acceptable results. When a SOP description is used, or BDD is the starting point, there is no structure preserved. The tools were, apparently, able to rediscover an acceptable

circuit structure; they are actually better than concluded in [1]. It is interesting to notice that FRAIG, the central structure of ABC, failed to play the role of structure-neutral representation despite its semi-canonicity.

Abandoning the structure of a circuit has two problems. Firstly, any conversion to a canonical structure has exponential worst case complexity, which is not practical. However, we obtained identical results from SOP inputs varying in size by an order of magnitude. Therefore, a representation which is not entirely canonical but can be generated in acceptable time suffices.

Secondly, an algorithm deciding when to drop the structure is needed. It was suggested [9] to detect outputs with common support but without common substructures in the circuit. For this to work, a structure like FRAIG is beneficiary. Currently, we do not have suitable examples to test this hypothesis.

B. Generating Arbitrary Circuits

Using XOR bi-decomposition, we obtained not only one acceptable solution, but multiple solutions of similar quality. None of them, however, was discovered *without* XOR bi-decomposition. Those solutions therefore form a class, characterized by the use of XOR operators and possibly by some structural properties.

We are not aware of any decomposition algorithm precisely characterized by the class of circuit structures it can produce. It is apparent that the broader the class is, the less probable are cases of poor performance. From this it follows that decomposition should treat AND and XOR operators equally. All 2-argument Boolean operators can be obtained using negation [10]. Let us call such a process *NPN-complete*.

C. Improving Global Control

All decomposition tools actually use multiple decomposition algorithms with top-level global control (e.g., [5]). We disabled or enabled entire partial algorithms in our experiments, which is a quite crude modification of the control. Yet there were measurable benefits. To have more partial algorithms (possibly running in parallel and/or on different representations) tried at each step would make the control less greedy, and, as in any other combinatorial optimization process, would bring a chance for improved results.

VI. Conclusions

The poor performance of existing synthesis tools on CD(G25) and parity examples has distinct reasons. CD(G25) needs the ability to disregard existing structure of the circuit. Parity examples require XOR decomposition and its integration into top-level control.

To overcome the observed limitations, structure-neutral circuit representation and efficient methods for manipulating them are desirable. The tools should be able to produce as broad a class of circuits as possible. Most importantly, the operators derived from AND and from XOR by input and/or output negation, shall be treated equally.

Acknowledgment

This research has been supported by MSMT under research program MSM6840770014. We are also thankful to dr. Minkovich for the LEKU circuits, dr. Steinbach for the BiDecomp executable, and dr. Ciriani for the discussions of their tools.

References

[1] J. Cong and K. Minkovich, "Optimality study of logic synthesis for LUT-based FPGAs", *IEEE Trans. CAD*, vol. 26, pp. 230–239, Feb. 2007. Postprint available free at: http://repositories.cdlib.org/postprints/2376.

[2] P. Fišer and J. Schmidt, "Small But Nasty Logic Synthesis Examples", in *Proc. 8th. Int. Workshop on Boolean Problems*, 2008, Freiberg, p. 183.

[3] E. Sentovitch, K. Singh et al., "SIS: A System for sequential circuit synthesis", Univ. California, Berkeley, Tech. Rep., UCB/ERL M92/41, May 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] C. Yang, M. Cieselski, V. Singhal, "BDS: A BDD-Based Logic Optimization System", in *Proc. 37th DAC'00*, 2000, p. 92.

[6] A. Mishchenko, B. Steinbach, M. Perkowski, "An Algorithm for Bi-Decomposition of Logic Functions", in *Proc. 38th DAC'01*, 2001, p. 103.

[7] A. Bernasconi, V. Ciriani, R. Drechsler, "Logic Minimization and Testability of 2-SPP Networks", *IEEE Trans. CAD*, vol. 27, pp. 1100–1202, July. 2008.

[8] A. Mishchenko, S. Chatterjee, R. Brayton, "DAG-Aware AIG Rewriting – A Fresh Look at Combinational Logic Synthesis", in *Proc. 43rd DAC'06*, 2006.

[9] A review of this contribution.

[10] M. A. Harrison, *Introduction to Switching and Automata Theory*, McGraw-Hill, 1965.

[11] S. Yang, "Logic Synthesis and Optimization Benchmarks User Guide", MCNC, Technical Report 1991-IWLS-UG-Saeyang, January, 1991.

[12] K. McElvain, "IWLS'93 Benchmark Set: Version 4.0", distributed as part of the IWLS'93 benchmark distribution