The Observed Role of Structure in Logic Synthesis Examples

Petr Fišer, Jan Schmidt
Czech Technical University in Prague
Faculty of Information Technologies, Dept. of Digital Design
e-mail: fiserp@fit.cvut.cz, schmidt@fit.cvut.cz

***Abstract*—Logic synthesis Examples with Known Upper bound by Cong and Minkovich are circuits hard to synthesize and map to look-up tables. During experiments with the synthesis of these examples, a way to obtain reasonable results was discovered. The crucial step is to abandon the circuit structure, and to describe the circuit independently of its structure. The feasibility of taking such an approach in general is then discussed.**

I. Introduction

An RTL circuit description is presently the standard starting point for an ASIC and FPGA industrial design. Logic synthesis, as the first step in RTL work flow, is believed to be a matured process, giving results reasonably close to optimum. Yet there are cases where existing synthesis tools perform provably poorly in terms of area and speed. Unless we understand the reasons for such a poor performance, we must expect it to happen in any practical circuit.

One of the most widely known example set are Logic Examples with Known Optimum (LEKO) and with Known Upper Bound (LEKU) by Cong and Minkovich [1]. Circuit synthesized by the tested tools have sometimes more than 500 times the expected size. The authors conjecture that, in the case of LEKU, contemporary logic synthesis tools are incapable of rediscovering the original structure of the circuit.

In this paper we focus entirely on these published examples. Their relevance to practice is discussed in [1]. Our primary aim is to understand the reasons why these examples are so hard to stynthesize properly, therefore we present the most straightforward way to achieve acceptable results; other more elegant and better scalable methods can be devised later.

We have discovered that the crucial step is to abandon the existing structure of the circuit. We have also found that many academic tools can structure the circuit closely enough to the optimum. In this respect, the situation is better than originally believed.

We of course cannot claim that this is the only method applicable. Moreover, we can only guess why an ill-devised structure of the circuit can prevent synthesis tools from decent performance.

To make the paper self-supporting to a degree, we give a brief overview of the examples and their construction. We then describe the synthesis experiments with some of the examples. We feel that the results allow us to formulate plausible conjectures, which are presented next.

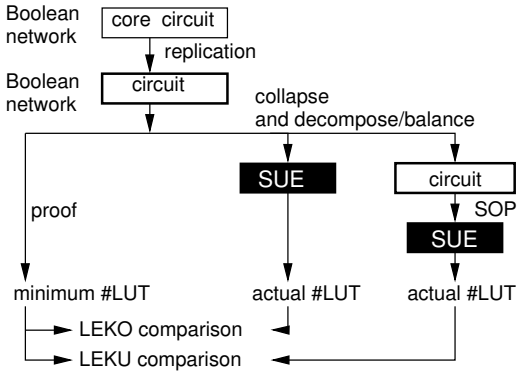

Fig. 1. Circuit construction and data flow of Cong and Minkovich for a SUE (Synthesis under evaluation)

II. Related Work

We were surprised to find no substantial response to [1] from the research community. Although the paper is easily accessible and frequently cited, the only published reactions are early press releases from EDA industry. Hence, we understand our observations to be the first steps towards understanding the problem.

III. Logic Synthesis Examples with a Known Upper Bound

Logic synthesis Examples with Known Optimum (LEKO) are constructed by replicating a relatively small circuit with n inputs and n outputs, given as a Boolean network of two-input nodes. Optimum mapping into look-up tables with 4 inputs (4-LUTs) is known. After the circuit multiplication, there is a path from each input to each output. It has been proven that the optimum mapping of the entire circuit retains the mapping of the core circuit. Hence, the optimum multiplied circuit size is equal to the respective multiple of the original circuit size. The resulting Boolean network can be used to evaluate the performance of LUT mappers against the proven optimum. The network can be also converted into a Sum-of-Product (SOP) description and used to evaluate any synthesis process capable of producing a 4-LUT mapping (Fig. 1). In this case, the proven mapping is only an upper bound of the circuit size, hence this type of evaluation circuits is referred to as Logic synthesis Examples with Known Upper bounds (LEKU). LEKU-CB examples are obtained using ABC [3] by

collapsing and balancing the circuit (by using ABC commands *collapse* and *balance*). On the other hand, LEKU-CD examples result from collapsing and technology decomposition by using SIS [2] commands *collapse* and *tech_decomp*.

IV. Experiments

LEKU-CD and LEKU-CB examples based on the G25 circuit were used, as they suffice to demonstrate the quality of the tested tools. The G25 circuit is known to have the upper bound of 70 4-LUTs. The CB version of the G25 circuit (CB(G25)) has 814 two-input gates, while CD(G25) is substantially larger, having 1167054 two-input gates.

The circuits are primarily meant as *mapping* examples. In a 'textbook' interpretation, mapping operates on a structure (represented by a graph) and produces a structure composed of look-up tables. Such processes are frequently based on FlowMap [8] or bin packing [7] algorithms.

In the original experiments, however, commercial tools were used, where the mapping phase cannot be separated from other logic synthesis steps. Moreover, even ABC [3] and SIS [2] are built this way to a degree; their mapping commands or recommended mapping scripts include substantial amount of resynthesis. Therefore, we understand the examples as general logic synthesis examples.

A. Experimental Setting

We used the following tools for decomposition: SIS [2], ABC [3], BDS [4], and BiDecomp [5]. Complying with respective licenses, we cannot present results obtained from commercial tools.

The principal property of the result we were interested in was whether a particular result is acceptable or shall be considered unacceptable (for example, multiple orders of magnitude larger than necessary), as presented, e.g., in [1]. For such an observation, almost any circuit measure suffices, and it is not very important whether the tool optimizes for speed or area.

The size of the resulting circuit was therefore measured as the number of 4-input LUTs, similarly to [10] and for similar reasons, namely to obtain a unified measure for decomposition algorithms working with different circuit representations.

Mapping by SIS was performed by running the script proposed in [2] for LUT synthesis, modified for mapping into 4-LUTs.

ABC decomposition was performed by running the *resyn2* script. The ABC *dsd* command [6] was also used to perform a disjoint-support decomposition.

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. We estimate that performance up to 10% better could be achieved with ideal cycle prevention.

B. Experiments

The first series of experiments was to merely synthesize the G25 circuit by SIS, ABC, and BDS with ABC mapping in the above described configurations. The size of the circuits obtained varied from 72 (ABC with resynthesis) to 136 (SIS) LUTs. BDS decomposition did not improve the result.

The second series of experiments aimed at the reproduction of the results presented in [1]. Prohibitively large circuits were obtained for the CD(G25) circuit. Repeated application of the FPGA mapping command sequence (see IV-A) did not yield results better than roughly 37,000 LUTs (more than 520-times the optimum). Resynthesis did not significantly help as well. We were unable to process CD(G25) by BDS and BiDecomp, since both applications crashed for this example circuit. The CB(G25) circuit apparently was problematic for all synthesis processes, however, the obtained result were no more than 2.7-times the optimum, thus acceptable.

The third series of experiments was performed on G25 in a SOP form. Various tools were used to obtain the SOP in the PLA format: ABC, MVSIS, SIS. The SOP obtained by ABC had roughly 19,000 terms and 14,000 terms after minimization with Espresso. The other collapsing tools gave SOPs roughly ten times larger than ABC. All these SOPs were synthesized by ABC and the combination of BDS and ABC in the above described configurations. In some cases, BiDecomp with ABC mapping was also tested. In all cases, both BDS with ABC and BiDecomp with ABC were better than ABC alone.

The fourth series also used SOP forms of the circuits, in this case the SOPs of CB(G25) and CD(G25). Again, all the SOPs were synthesized as in the previous series.

C. Results

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.

Surprising results were obtained from pre-processing the collapsed G25 minimized by Espresso. Excessively large results were obtained by ABC (1520 LUTs after resynthesis). However, BDS and ABC *dsd*, again, were able to rediscover the circuit structure. Espresso was unable to minimize the circuits collapsed by MVSIS and SIS, hence those results are not presented.

Notice, that when the CB(G25) circuit was collapsed by ABC, the resulting PLA was equal to the PLA of the collapsed original G25, and thus the synthesis results were equal as well. Very similar results were obtained by collapsing CB(G25) by MVSIS and SIS, hence the results are not presented.

Similarly, when CD(G25) was collapsed by ABC, a PLA of a very similar size to the collapsed original G25 was produced. At the end, synthesis results were equal to the results obtained from the collapsed G25. The ABC *dsd* command generated the

TABLE I
SYNTHESIS RESULTS OBTAINED FOR DIFFERENT FORMS OF AN INPUT

Input	SOP size [terms] [literals]	Tools combination	Result size [LUTs]
G25	–	ABC *fpga*	80
		ABC *resyn2* + *fpga*	72
		ABC *dsd* + *fpga*	102
		SIS	136
G25 collapsed by ABC	18905 218116	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 1755530	ABC *fpga*	232
		ABC *resyn2* + *fpga*	170
		ABC *dsd* + *fpga*	115
		BDS + ABC *fpga*	113
		SIS	failed
G25 collapsed by SIS	201816 3150049	ABC *fpga*	1140
		ABC *resyn2* + *fpga*	900
		ABC *dsd* + *fpga*	102
		BDS + ABC *fpga*	113
		SIS	failed
G25 collapsed by ABC + Espresso	14619 248967	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 218116	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 212206	ABC *fpga*	280
		ABC *resyn2* + *fpga*	200
		ABC *dsd* + *fpga*	102
		BDS + ABC *fpga*	113
		BiDecomp + ABC *fpga*	188
		SIS	failed

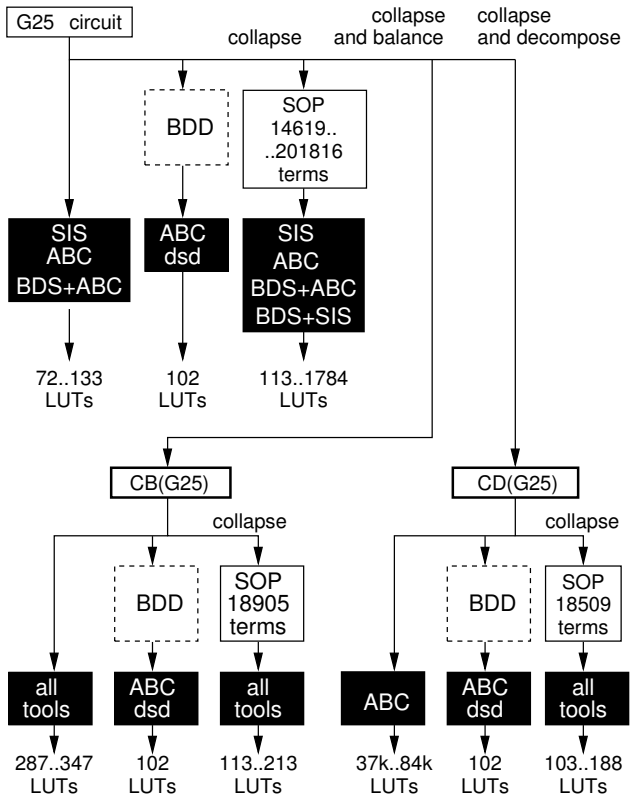

Fig. 2. A summary of experiments with LEKU examples.

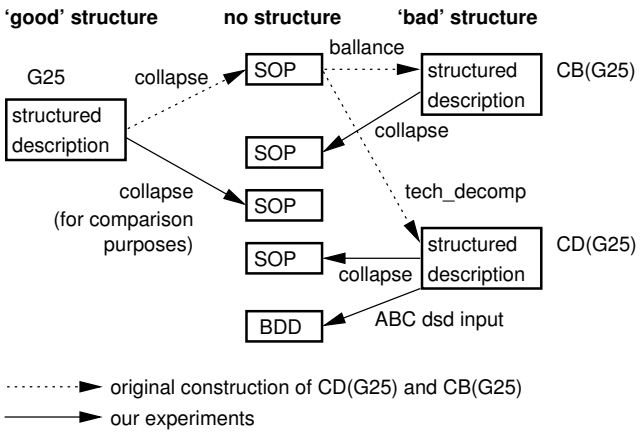

Fig. 3. The structure of inputs to the experiments

same result as in all the previous cases. We did not succeed in collapsing CD(G25) by MVSIS or SIS.

Figure 2 summarizes the data flow and main results from the experiments. Where multiple results were obtained using different tools and/or different versions of input data, a range is indicated in Figure 2. Notice that the number of literals generally follows the number of terms. Thew only exception is G25 minimized by Espresso, which decreased the number of terms but increased the number of literals.

V. INTERPRETATION AND DISCUSSION

First of all, we did find a way to map CD(G25) with a reasonable quality (147%) to LUTs using existing tools.

The original G25 description, CB(G25), CD(G25), and the derived SOPs specify identical sets of Boolean functions; the transformations give them 'good', 'wiped' or 'bad' structure. The designed, 'good' structure is preserved in G25. It is wiped out in the SOP or in a BDD, a non-optimal structure is introduced by *ballance* and a yet worse structure in CD(G25) by *tech_decomp* (Figure 3).

As can be seen from Figure 2, synthesis from the 'good' structure is without problems for most of the tools. This agrees with findings in [1]. The ABC command *fpga* (with or without *resyn2*) produces worse results from a SOP minimized by Espresso; the small increase in literal count can be hardly identified as the cause.

In the case of 'wiped' structure, satisfactory results can be

still obtained from CD(G25) in all experiments. The ABC command *dsd* [6] was able to obtain satisfactory results in all the explored cases; the obtained result was only 1.46-times the optimum. We believe this is due to using a global BDD data structure to represent the decomposed function. This way, the original circuit structure is also eliminated, similarly to the SOP form.

'Bad' structure of the circuit invariantly leads to poor results as in the original experiments by Cong and Minkovich.

Achieving relatively good results from the 'wiped' structure also means that the tools have the ability to rediscover a reasonable structure of the circuit, in disagreement with [1]. The term 'reasonable' here of course relates to the best solution known, which is the upper bound of the LEKU example.

'Being free of circuit structure', when interpreted rigidly, means employing some canonical representation of the circuit. The sum of products is certainly not one. However, it *plays that role*. From SOPs of different sizes (201816 and 18905 terms, respectively), we obtained the same results. The BDD used by ABC *dsd* is canonical with respect to variable ordering, which we believe to carry little information about the former structure of the circuit.

We see that a varying structural description influences the result greatly, while a varying SOP size does not. It is not surprising; the SOP has always been considered to capture the behavior only and the algorithms are constructed that way. On the other hand, there are – and always were – reasons to preserve the structural description, at least as a starting point of the optimization. Perhaps the most practical reason is that a human designer tends to produce structures that make sense.

We have found that discarding an inappropriate structure cures the problems in synthesis of the LEKU examples. Of course, we do not claim this to be a panacea. There are two kinds of problems: in throwing away the structure and in discovering a better one.

A practical synthesis tool shall predict when discarding the structure is necessary or profitable, being able to distinguish an inappropriate circuit structure from a reasonable one. Further, no Boolean function representation is known with worst case size better than exponential in the number of inputs, which means that the construction can fail anyway.

There are indications that the situation in rediscovering a good structure is not hopeless. Many – especially older – synthesis benchmarks are SOP-based, and therefore the algorithms are prepared to work from a neutral description. Potential problems with large circuits seems to lie in parallel with the problem of large circuits representations. There is the possibility, however, that not only the optimal solution, but also all acceptable suboptimum solutions have a structure that cannot result from algorithms used in that particular case.

When we attribute the difficulty of LEKU examples to a misleading structure, we also say the problem can occur in practice. There is no guarantee that, e.g., a high-level synthesis tool would produce a structure as misleading as that one generated by *tech_decomp* from a SOP. From this point of view, G25 *is* a practical circuit.

VI. Conclusions

In contrast to the results presented in [1], we have shown that the Logic Synthesis Examples with Known Upper Bound can be synthesized into a circuit of a reasonable size using existing academic tools. The crux is to abandon the structure contained in the CB(G25) and CD(G25) circuit description. The sum-of-products can be used as a neutral circuit description. The neutrality lies not only in actual loss of structural information, but also in the fact that the tools understand it as a neutral description. To apply such an approach, synthesis tools shall know when abandoning the structure is profitable, must be able to construct some kind of neutral description, and must construct a suboptimum structure from such a description. Synthesis problems similar to the analyzed one can recur in practice whenever a tool or a user produce a circuit with an inappropriate structure.

Acknowledgment

This research has been supported by MSMT under research program MSM6840770014 and GA102/09/1668. We are also thankful to dr. Minkovich for the LEKU circuits, and dr. Steinbach for the BiDecomp executable.

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] E. Sentovitch, K. Singh et al., "SIS: A System for sequential circuit synthesis", Univ. California, Berkeley, Tech. Rep., UCB/ERL M92/41, May 1992.

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

[4] C. Yang, M. Cieselski, V. Singhal, "BDS: A BDD-Based Logic Optimization System", in *Proc. 37th DAC'00*, 2000, p. 92.

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

[6] V. Bertacco and M. Damiani, "Disjunctive decomposition of logic functions", in *Proc. ICCAD'97*, 1997, pp. 78-82.

[7] R. J. Francis, J. Rose, and Z. Vranesic, "Technology mapping of lookup table-based FPGAs for performance," in *IEEE/ACM International Conference on Computer-Aided Design*, 1991, pp. 568–571.

[8] J. Cong and Y. Ding, "FlowMap: An optimal technology mapping algorithm for delay optimization in lookup-table based FPGA designs", *IEEE Trans. CAD*, Vol. 13, Jan 1994, pp. 1–12.

[9] A. Mishchenko, S. Chatterjee, R. Brayton, "Improvements to Technology Mapping for LUT-Based FPGAs", in *Proc. 43rd FPGA'06*, 2006.

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