New Stories on the Structural Bias in Logic Synthesis

Petr Fišer, Jan Schmidt
Department of Digital Design, Faculty of Information Technology
Czech Technical University in Prague
Thákurova 9, Prague, Czech Republic
fiserp@fit.cvut.cz, schmidt@fit.cvut.cz

***Abstract*—Two-level logic optimization is a well-established process that efficiently minimizes a Sum-of-Products (SOP) form. Espresso is the most known two-level minimizer since its release in 1984. And indeed, it produces optimum or near-optimum results regarding the SOP size. However, there could be an issue with multi-level circuit implementation. When a two-level minimizer processes a completely specified function before multi-level synthesis, it helps since no information is lost. However, as for incompletely specified functions, this needs not be the case since the two-level minimization could introduce a heavy structural bias by assigning the don't cares particular values.**

In this paper, we show that a two-level minimization conducted in the standard way significantly worsens the result of the subsequent multi-level optimization for heavily incompletely specified multiple-output functions, regardless of the optimization process and tool used. In particular, the more incompletely specified the input function is, the more harm the multiple-output minimization does. In conclusion, we propose using a single-output minimization instead. Using this procedure, the obtained circuits are sometimes more than 2-times smaller, while no significant deterioration has been observed for any other functions.

***Index Terms*—logic synthesis, two-level minimization, incompletely specified functions, Espresso**

I. Introduction

When the source to logic synthesis is a tabular description of a function (truth table, set of terms, the Berkeley PLA format), the well-established synthesis process is to run two-level minimization, e.g., Espresso [1] and then proceed with multi-level optimization and technology mapping. In the case of completely specified functions, running the two-level minimization *explicitly* is typically not even needed since multi-level synthesis tools, like SIS [2] and ABC [3], can do the job themselves. However, in the case of incompletely specified functions (e.g., where values of only some minterms are specified), preprocessing the function by a two-level minimizer is often necessary. For example, when reading an incompletely specified PLA, ABC substitutes the don't care values with zeroes, which loses many opportunities for efficient optimization. In general, ABC cannot deal with external don't cares [4].

On the other hand, SIS [2] is able to read incompletely specified PLAs and the don't cares are recognized. In particular, these external don't cares are added to observability don't cares [4] in the `full_simplify` command. However, this procedure is rather time-consuming and hardly applicable to functions with many input variables.

Thus, when one wants to efficiently utilize don't cares in multi-level synthesis by, e.g., ABC, the two-level minimization should be run first, to produce a minimized completely specified function from an incompletely specified one, hoping that the don't cares would be assigned a value efficiently. A multi-level optimization and technology mapping is then run on this completely specified function.

This procedure is believed to yield "reasonable" results. Well, it does, for almost completely specified functions. However, we have found a big issue with heavily incompletely specified functions. We have observed that the *multiple-output* two-level minimization of *incompletely specified* functions introduces an unwanted structural bias [5], significantly affecting the subsequent two-level synthesis and optimization. In particular, we think that generated shared terms are not suitable for multi-level optimization. The observed affection is not negligible – results almost three times as big as those generated by an "unconventional" way are produced. In particular, even though the multiple-output two-level minimization produces results having much fewer terms than the single-output one, the effect of the subsequent multi-level synthesis is the opposite.

In this paper, we document that for incompletely specified multi-output functions, running the Espresso [1] *multiple-output* two-level minimization prior to the multi-level synthesis is not a good option. Running the *single-output* minimization instead helps significantly. This observation holds for all subsequent multi-level synthesis processes we have experimented with. We have experimented with the Boom minimizer [6] as well. No such deterioration has been observed here, but this can be attributed to its inability to perform the multi-output minimization efficiently.

The paper is structured as follows: Section II presents the motivation behind the research. In fact, this paper mostly presents experimental results but with observations important to practice. These are presented in Section III. An attempt to explain the observations is given in Section IV. Finally, the conclusions are drawn in Section V.

The contributions of this paper can be summarized as follows. We show that:

- Doing multi-output minimization of incompletely specified random functions by Espresso is not a good choice

if a multi-level circuit synthesis is supposed to be run afterward.

- There is even no difference in the multi-level synthesis tool and algorithm used when used after the two-level minimization.
- Finally, for incompletely specified functions, we propose conducting a *single-output minimization* prior to the multi-level synthesis instead of the standard multiple-output one.

II. The Story Behind

When designing direct implementations of neural networks, i.e., where the network is implemented as a purely combinational circuit [7], we have run into serious problems. In particular, the neural network we wanted to synthesize into hardware was described as an incompletely specified function having relatively many (100) inputs, relatively many (100) outputs, and tenths of thousands of specified minterms. This function was described in a PLA format [8]. Note that despite the large size of this description, this function is still heavily incompletely specified. In order to efficiently exploit the don't cares, we tried to run Espresso [1] to optimize the PLA. However, it took an extremely long time to run, and after all, it crashed. Thus, we have decided to sacrifice the optimality to some extent and split the function by outputs into two PLAs having 100 inputs and 50 outputs. Thus, the power of the multi-output two-level minimization cannot be fully exploited in this case. This still did not suffice, so we continued with such splitting. After all, when arriving at 25 outputs, the minimization successfully completed. But still, the run times were very high (several days). Thus, we have decided to continue with the splitting, ending up with the single-output minimization. The run times were obviously smaller.

To evaluate how much harm the splitting did, we have compared the results obtained by the application of splittings by different numbers of outputs. One observation was expected: *the more splitting was performed, the more terms and literals the resulting PLAs had*, since the multi-output minimization cannot be fully exploited in the case of splitting. However, the second observation was rather surprising: *the effects of the splitting on the multi-level optimization were exactly the opposite – the more output variables the functions submitted to Espresso had, the more gates had the final networks*. Moreover, this behavior was systematic for all the examples we have tried. After all, we have concluded that when the target technology is not a PLA, *a single-output minimization is the best option in the case of heavily incompletely specified functions*.

To illustrate the aforementioned observations, we have synthesized a neural network from the MNIST benchmark [9]. The neural network had three layers, each having 100 binary inputs and 100 binary outputs, with 60,000 specified minterms. Functions of different layers (L2, L3, and their concatenation L2.L3) were synthesized by Espresso and then by ABC, using a script mapping the circuit into arbitrary 2-level gates:

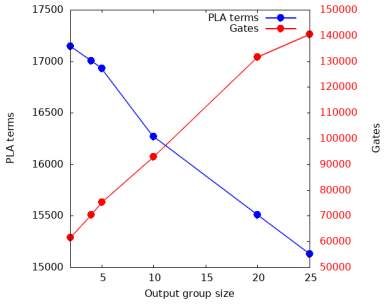

Fig. 1. The influence of group minimization on the number of PLA terms and the resulting number of 2-input gates after synthesis by ABC

```
&st; &synch2; &if -m -a -K 2;
&mfs -W 10; &st; &dch; &if -m -a -K 2;
&mfs -W 10.
```

The script was repeated 20-times.

The results are shown in Table I. The columns represent the number of variables in the minimized functions. I.e., "1" represents a single-output minimization, while "100" represents group minimization of the original, non-split function. The rows indicate the respective functions. In each cell, the first value represents the number of distinct terms of the SOP obtained by Espresso, and the second value represents the number of 2-input gates after ABC synthesis.

For better illustration, we show the results for the most complex functions (the L2.L3) in a graph, see Fig. 1.

It can be clearly seen that when the number of outputs minimized together increases, the number of PLA terms decreases (up to some minor exceptions, which can be attributed to random fluctuations). However, the number of gates after multi-level synthesis *consistently* increases. Therefore, these preliminary results indicate that running a single-output minimization prior to the multi-level optimization is the best option. This phenomenon will be further explored in the next section.

Note that the exercised functions mostly exhibit a random nature, and so they heavily differ from standard benchmarks. However, such functions are becoming ever more important in practice, especially because of the advent of the direct implementation of neural networks [7].

III. When Single-Output Minimization Pays Off

Having the clue that a single-output two-level minimization could yield better results for some "random" incompletely specified functions, we have performed exhaustive experimental work in order to find out *when* does it yield better results, i.e., what properties the function must have. We will restrict ourselves to exploring functions (PLAs) where there is a given number (ratio) of *minterms* specified, as this scenario most closely resembles functions obtained from the direct implementation of neural networks [7].

In particular, we have generated random functions (PLAs, the `fr` type [1]) with a given number of inputs, outputs, and specified minterms. The output values of the minterms were

TABLE I
SPLITTING RESULTS

Layers		Outputs 1	2	4	5	10	20	25	100
L2	PLA terms	11,097	11,003	10,960	10,929	10,802	10,460	10,321	N/A
	2-input gates	33,810	36,394	41,635	43,606	53,974	71,531	73,371	
L3	PLA terms	6,409	6,326	6,202	6,165	5,884	4,317	5,635	N/A
	2-input gates	18,647	20,121	22,861	24,496	29,847	29,181	41,757	
L2.L3	PLA terms	17,309	17,142	17,005	16,932	16,270	15,511	15,129	N/A
	2-input gates	57,708	61,564	70,450	75,290	92,895	131,518	140,360	

also completely specified. Thus, the don't cares were given by the minterms missing in the PLA description. The distribution of ones and zeroes in both the AND plane and the OR plane was uniform, i.e., 50:50. These functions were then processed in two ways:

1) Multi-output minimization by Espresso + synthesis into 2-input gates by ABC, as described in Section II,
2) Single-output minimization by Espresso (using the `-Dso` option) + synthesis into 2-input gates by ABC, as described in Section II,

The results will be shown in the following subsections, particularly Sec. III-A through III-D.

A. The Nice Cases

One would expect that the multi-output two-level minimization will yield better results than the single-output minimization, even after the multi-level synthesis. Actually, we haven't observed such cases for the scenario outlined above. Even for very small numbers of both input and output variables, the single-output minimization produced better or equal results on average.

B. The Expected Cases

As the "expected cases", we denote the behavior observed in Section II. For the first illustration, we will present results obtained from the smallest functions we have tried out, functions with five inputs and five outputs, where we have varied the number of specified minterms from 10 to 32, i.e., up to a completely specified function. We have generated 80 random functions for each number of specified minterms. We have measured the ratio of the numbers of the resulting 2-input gates for the two above-mentioned scenarios. In particular, the ratio was obtained by dividing the number of gates obtained when a multi-output Espresso minimization is run by the number of gates obtained when a single-output Espresso minimization is run. Therefore, values higher than one favor single-output minimization.

The results in the form of a scatter plot are shown in Fig. 2. We can see that the single-output minimization wins on average, regardless of the number of specified minterms. But still, a slight tendency to approach the "1" value with the increase of the specified terms can be observed.

C. The Usual Cases

With the increasing number of variables (both input and output ones), things are becoming "worse". Not only does

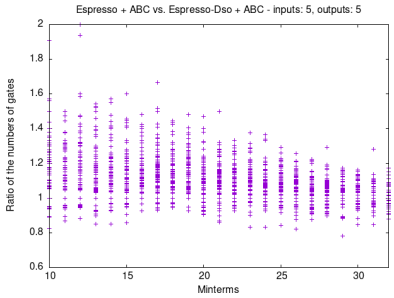

Fig. 2. Comparison of multi-output and single-output minimization by Espresso run prior to the synthesis by ABC, functions with 5 inputs and 5 outputs

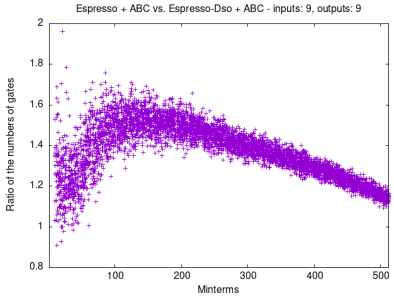

Fig. 3. Comparison of multi-output and single-output minimization by Espresso run prior to the synthesis by ABC, functions with 9 inputs and 9 outputs, 10 random samples for each number of specified minterms

the single-output synthesis almost always produce better results, but there emerges a region at some ratio of specified minterms, where the multi-output minimization performs extremely worse. This is illustrated in Fig. 3, for a function with 9 inputs and 9 outputs. This phenomenon starts to appear at functions with 7 inputs and 7 outputs already and is emphasized when the numbers of variables grow. This is illustrated in Fig. 4.

D. The Strange Cases

After careful exploration, we can see an appearance of some strange behavior in Fig. 4 – there are some outlying cases where the multi-output synthesis fails more than on average. When the number of variables (both the input and output ones) increases even more, strange things start to happen – there emerges a "gap" separating, say, bad and very bad results. This behavior starts even when the number of outputs is increased by one, with respect to the 10x10 example, see Fig. 5.

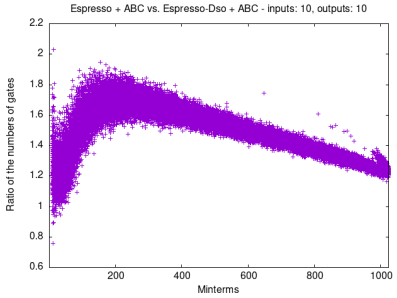

Fig. 4. Comparison of multi-output and single-output minimization by Espresso run prior to the synthesis by ABC, functions with 10 inputs and 10 outputs, 40 random samples for each number of specified minterms

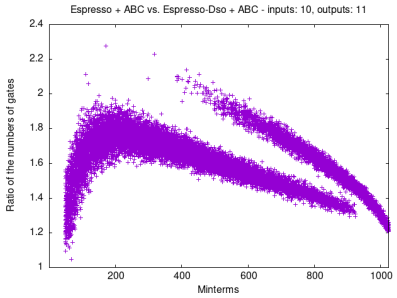

Fig. 5. Comparison of multi-output and single-output minimization by Espresso run prior to the synthesis by ABC, functions with 10 inputs and 11 outputs, 10 random samples for each number of specified minterms

When we increase the number of variables, the gap further increases, see Fig. 6. This happens consistently, see the summary in Sec. III-F. Honestly, we do not have any explanation for this phenomenon.

E. When Things Go Really Bad

When we increase the number of variables further, the above-mentioned phenomenon (the "gap") starts to disappear, and we return to the cases similar as shown in III-C. However, the deterioration is just bigger. This is illustrated in Fig. 7, for 10 inputs and 15 outputs. As a result, one may think that the relatively good cases have disappeared and the bad cases remained – in simple words, the rightmost curve from Fig. 6

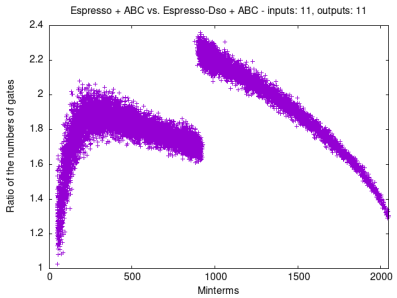

Fig. 6. Comparison of multi-output and single-output minimization by Espresso run prior to the synthesis by ABC, functions with 11 inputs and 11 outputs, 10 random samples for each number of specified minterms

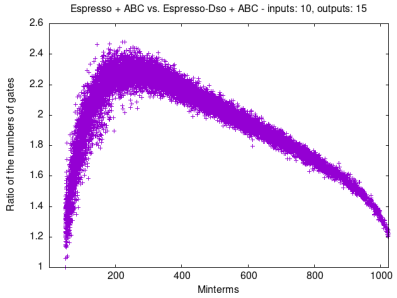

Fig. 7. Comparison of multi-output and single-output minimization by Espresso run prior to the synthesis by ABC, functions with 10 inputs and 15 outputs, 10 random samples for each number of specified minterms

starts to span over the whole range of defined minterms. And most probably, it is so.

F. Summary of the Espresso Results

We have done extensive experimental work to prove that the observed kinds of behavior are not just random outliers. The behavior is consistent over randomly generated functions with different numbers of input and output variables. The summary of all the observations is shown in Table II. We can classify the results obtained from processing functions of different numbers of input (rows) and output variables (columns) by the cases described above. The missing values were not calculated because of excessive computation power demands, but we think the results are representative enough.

IV. Who is the Guilty?

At this point, we have come to the conclusion that the multi-output minimization of heavily incompletely specified functions done by Espresso introduces an unwanted structural bias, and, as a consequence, the multi-level optimization fails. Now we should ask who is guilty of that. In particular, is it an insufficiency of the two-level minimization or of the subsequent multi-level optimization? We will try to answer this question here.

A. The Espresso vs. Boom Case

In an attempt to find out if the multi-output two-level minimization by Espresso is the reason behind all the above-mentioned phenomena, we have tried out an alternative two-level minimizer, Boom [6]. First, we have run Boom with and without the multi-output minimization (i.e., where the implicants reduction phase has been switched off) and compared the results obtained after ABC synthesis. An illustrative example is shown in Fig. 8. As we can observe, there is no significant dependency.

One may attribute this behavior to a smaller efficiency of Boom for such functions, as Boom has been designed to minimize functions with hundreds or thousands of input variables, i.e., to different problem instances. Indeed, this is true, as shown in Fig. 9, where the same functions as above have been processed. As we can see, most of the ratio values are below one, indicating the insufficiency of Boom.

TABLE II
SUMMARY OF THE ESPRESSO RESULTS. B = EXPECTED CASES, C = USUAL CASES, D = THE STRANGE CASES, E = WHEN THINGS GO REALLY BAD

Inputs/Outputs	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16	17	18	19	20
5				B															
6					B														
7						C													
8							C												
9								C											
10	B	B	C	C	C	C	C	C	C	D	D	D	D	E	E	E	E	E	E
11	B	C	C	D	D	D	D	D	D	D									
12	B	D	D	D	D	D	D	D	D	D	D								

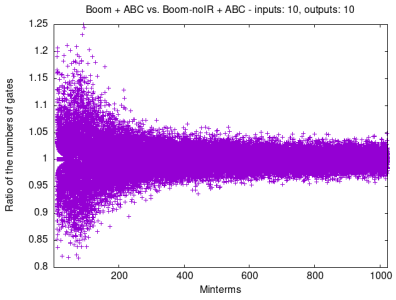

Fig. 8. Comparison of multi-output and single-output minimization by Boom run prior to the synthesis by ABC, functions with 10 inputs and 10 outputs, 40 random samples for each number of specified minterms

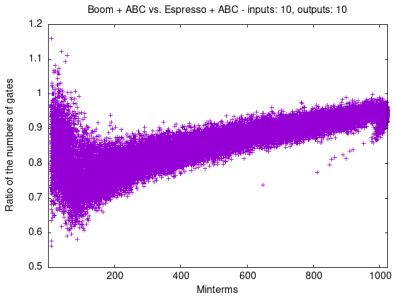

Fig. 9. Comparison of multi-output minimization by Boom and Espresso run prior to the synthesis by ABC, functions with 10 inputs and 10 outputs, 40 random samples for each number of specified minterms

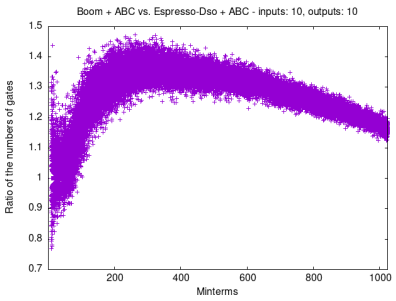

Fig. 10. Comparison of multi-output minimization by Boom and single-output minimization by Espresso run prior to the synthesis by ABC, functions with 10 inputs and 10 outputs, 40 random samples for each number of specified minterms

B. The Espresso vs. Boom Case - The Second Round

In the previous subsection, we have concluded that BOOM is inferior to Espresso in terms of multi-output optimization of the functions targeted in this paper. But still, the results were relatively consistent and unsurprising. Now, we may ask if the single-output Espresso minimization followed by multi-level optimization outperforms the multi-output minimization done by Espresso as well. The ultimate answer is seen in Fig. 10, for the same set of random functions of 10 inputs and 10 outputs. Here we can see the "bump" again. This draws us to a preliminary conclusion: *The single-output minimization done by Espresso, when run prior to the multi-level synthesis, yields the best results*.

C. The Effect of Multi-level Synthesis

To verify that the two-level minimization is responsible for the behavior described above, we have also studied different multi-level synthesis processes that followed the two-level minimization. In particular, we believe that the strongest multi-level synthesis tool able to mitigate the structural bias is BDS [10], which is able to perform a Boolean decomposition efficiently and partially mitigate the effects of structural bias by converting the source file to a ROBDD [11]. Thus, we have run BDS after the Espresso minimization, prior to the ABC synthesis. The comparison of the results obtained by running the Espresso minimization (multiple-output and single-output) followed by processing the result by BDS and then by ABC is shown in Figs. 11 and Figs. 12. We can see from these figures that the deterioration is really caused by the inefficient usage of don't cares in the two-level optimization – the more don't cares there are in the source function, the worse results are obtained.

D. Espresso with Commercial Tools

Finally, one may ask whether Espresso is really the guilty one and whether the poor results are not because of some flaws in ABC. Thus, we have performed similar experiments with commercial tools. The initial incompletely specified PLAs were minimized by Espresso (using both single- and multiple-output minimization), the resulting PLAs were converted to VHDL in a straightforward way (by our in-house tool) and then synthesized by Xilinx Vivado 2023.1 (synthesis into the Artix-7 chip) and Intel Quartus Prime 22.1 (synthesis into the Cyclone V chip).

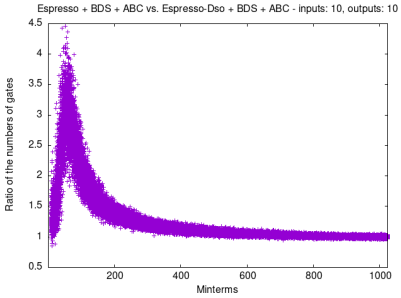

Fig. 11. Comparison of multi-output and single-output minimization by Espresso run prior to the synthesis with BDS followed by ABC, functions with 10 inputs and 10 outputs, 40 random samples for each number of specified minterms

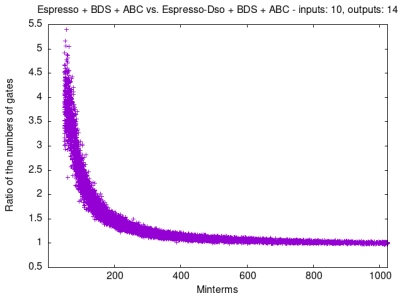

Fig. 12. Comparison of multi-output and single-output minimization by Espresso run prior to the synthesis with BDS followed by ABC, functions with 10 inputs and 14 outputs, 10 random samples for each number of specified minterms

The results are similar to ABC: when multi-output minimization is used, the outcome is almost always worse. This is illustrated by four representatives: two for Quartus (Fig. 13 and 14) and two for Vivado (Fig. 15 and 16). Again, the values higher than one indicate the inferiority of multiple-output synthesis. Therefore, we may conclude that ABC is innocent in this regard, as similar effects can be observed even when using commercial tools.

E. Is Espresso the Guilty?

Until now, we have concluded that the Espresso multi-output two-level minimization, when run prior to *any* multi-level synthesis, is responsible for the lack of quality. Let us now try one last means to take advantage of dont't cares in the function specification – SIS [2]. In particular, the `script_rugged` performing the Boolean optimization. Here, the `full_simplify` command takes don't cares present in the source PLA as external don't cares and appends them to observability don't cares in the node simplification. In this experiment, we have taken the best optimization strategy (single-output two-level minimization by Espresso + ABC) and compared it with running the `script_rugged` (20-times) followed by the ABC optimization and mapping (the same process as in the previous experiments). Example results are shown in Fig. 17. As we can see, no conclusive results can

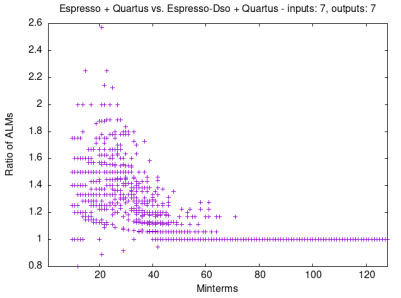

Fig. 13. Comparison of multi-output and single-output minimization by Espresso run prior to the synthesis with Quartus, functions with 7 inputs and 7 outputs, 20 random samples for each number of specified minterms

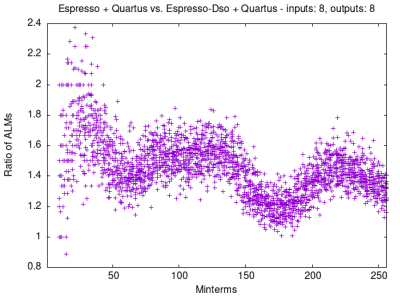

Fig. 14. Comparison of multi-output and single-output minimization by Espresso run prior to the synthesis with Quartus, functions with 8 inputs and 8 outputs, 20 random samples for each number of specified minterms

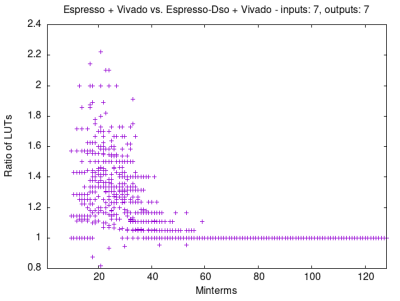

Fig. 15. Comparison of multi-output and single-output minimization by Espresso run prior to the synthesis with Vivado, functions with 7 inputs and 7 outputs, 20 random samples for each number of specified minterms

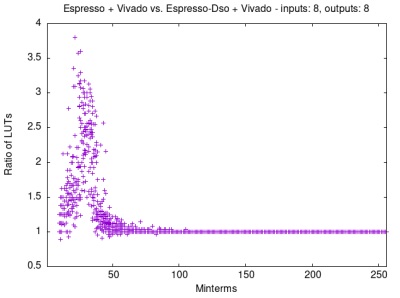

Fig. 16. Comparison of multi-output and single-output minimization by Espresso run prior to the synthesis with Vivado, functions with 8 inputs and 8 outputs, 20 random samples for each number of specified minterms

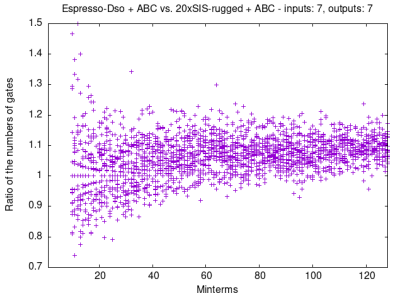

Fig. 17. Comparison of SIS synthesis and a single-output minimization by Espresso run prior to the synthesis with ABC, functions with 7 inputs and 7 outputs, 20 random samples for each number of specified minterms

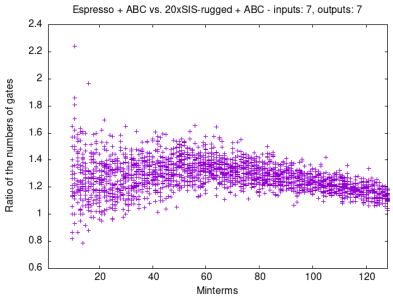

Fig. 18. Comparison of SIS synthesis and a multi-output minimization by Espresso run prior to the synthesis with ABC, functions with 7 inputs and 7 outputs, 20 random samples for each number of specified minterms

be seen – all ratios are close to 1, with some slight fluctuations.

Finally, let's see what will happen if we substitute the single-output Espresso minimization with a multi-level one. The results are shown in Fig. 18. Again, it can be clearly seen that the multi-output minimization causes harm.

Unfortunately, the experiments are presented on small functions only (7 inputs and 7 outputs since the `script_rugged` took too much time to solve bigger instances. However, the results seem to be representative enough – we have tried to synthesize some larger circuits, and the behavior was consistent with the results shown here.

F. The Case of Standard Benchmarks

Randomly generated functions were studied until now. Thus, one may ask if *those* are the reasons for the reported failure of multi-output minimization, or if this holds in general, i.e., for standard benchmarks. However, we are not aware of any benchmarks with circuits of the required property: being heavily incompletely specified functions.

But anyway, we have conducted experiments with the Espresso suite benchmarks [8]. The experiment scenario was the same as in the initial cases: single-output and multiple-output minimization was run first, and then the result was mapped to 2-input gates by ABC. The result was expected: no conclusive result. The average improvement when the single-output minimization was used was 1.7%. There were some outliers where the single-output minimization helped (e.g., `s1269` with 2.4x improvement), but in many cases there was a big deterioration. Note that all these PLAs were completely specified or with a few don't cares only. Thus, these standard benchmark cases fall into the rightmost regions of all the figures above.

V. Conclusion

We have shown experimentally that the structural bias introduced by the multiple-output two-level minimization of incompletely specified functions is the cause of the inefficiency of the subsequent multi-level synthesis, not the two-level minimization itself. In the case of *heavily incompletely specified functions*, the two-level minimization done by Espresso is just not able to exploit the don't cares efficiently when the target implementation is a multi-level network, no matter what multi-level optimization process follows. We have shown that the best option is to use a single-output two-level minimization since the multiple-output two-level minimization only spoils the result. This observation can be generalized for all randomly generated incompletely generated functions, which actually resemble the state-of-the-art applications of logic synthesis – the synthesis of neural networks implemented as combinational circuits.

Acknowledgment

Computational resources were provided by the e-INFRA CZ project (ID:90254), supported by the Ministry of Education, Youth and Sports of the Czech Republic.

References

[1] R. K. Brayton, A. L. Sangiovanni-Vincentelli, C. T. McMullen, and G. D. Hachtel, *Logic Minimization Algorithms for VLSI Synthesis*. Norwell, MA, USA: Kluwer Academic Publishers, 1984.

[2] E. Sentovich, K. Singh, L. Lavagno, C. Moon, R. Murgai, A. Saldanha, H. Savoj, P. Stephan, R. K. Brayton, and A. L. Sangiovanni-Vincentelli, "SIS: a system for sequential circuit synthesis," EECS Department, University of California, Berkeley, Tech. Rep. UCB/ERL M92/41, 1992.

[3] A. Mishchenko *et al.*, "ABC: A system for sequential synthesis and verification," 2012. [Online]. Available: http://www.eecs.berkeley.edu/~alanmi/abc

[4] G. D. Hachtel and F. Somenzi, *Logic Synthesis and Verification Algorithms*. Norwell, MA, USA: Kluwer Academic Publishers, 1996.

[5] P. Fiser, J. Schmidt, and J. Balcarek, "Sources of bias in EDA tools and its influence," in *IEEE Symposium on Design and Diagnostics of Electronic Circuits and Systems*, April 2014, pp. 258–261.

[6] J. Hlavicka and P. Fiser, "BOOM – a heuristic boolean minimizer," in *IEEE/ACM International Conference on Computer Aided Design. ICCAD 2001. IEEE/ACM Digest of Technical Papers (Cat. No.01CH37281)*, 2001, pp. 439–442.

[7] M. Nazemi, G. Pasandi, and M. Pedram, "Energy-efficient, low-latency realization of neural networks through boolean logic minimization," in *24th Asia and South Pacific Design Automation Conference*, Jan 2019, pp. 274–279.

[8] R. Rudell and A. Sangiovanni-Vincentelli, "Multiple-valued minimization for PLA optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 6, no. 5, pp. 727–750, September 1987.

[9] L. Deng, "The MNIST database of handwritten digit images for machine learning research," *IEEE Signal Processing Magazine*, vol. 29, no. 6, pp. 141–142, 2012.

[10] C. Yang and M. Ciesielski, "BDS: a BDD-based logic optimization system," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 21, no. 7, pp. 866–876, Aug. 2002.

[11] R. E. Bryant, "Graph-based algorithms for Boolean function manipulation," *IEEE Transactions on Computers*, vol. 35, no. 8, pp. 677–691, Aug. 1986.