DOI: 10.31114/2078-7707-2022-3-212-218

Exact Synthesis of Low Precision Multipliers for Intel FPGAs

Mikhail Shupletsov[1], Vladimir Zhukov[1], Sergey Gribok[2], Ilya Ganusov[2], Mikhail Mestetskiy[1], Mikhail Lopunov[1], Ekaterina Kuprash[1]

[1]Lomonosov Moscow State University, Moscow, Russian Federation, shupletsov@cs.msu.ru

[2]Intel Corporation, San Jose, USA, ilya.ganusov@intel.com

***Abstract* — Building efficient FPGA-based AI inference accelerators sets new requirements for FPGA compilers. AI inference accelerator contains very large amount of very small identical circuits (such as low precision multipliers). So, in order to maximize the overall accelerator performance, it is very important to build those small circuits optimally (in terms of area and delay). SAT-based exact synthesis is known to be an efficient technique to build optimal LUT-based circuits for logic functions with a small number of inputs.**

In this paper we extend exact synthesis methods to support FPGA Adaptive Logic Module (ALM) structure. We present a SAT-based exact synthesis tool that builds optimal Stratix 10 and Agilex ALM-based circuits. We use the tool to build optimal FPGA circuits for low-precision multipliers and demonstrate that the new circuits are 10-50% more area-efficient if compared with circuits generated by Quartus Compiler. Furthermore, we demonstrate that our approach identified several new mappings with improved delay. To the best of our knowledge, this is the first application of exact synthesis realized on commercial FPGA architecture.

***Keywords* — Exact synthesis, Agilex, Stratix 10, multipliers.**

I. INTRODUCTION

In the recent years there is a growing interest to use FPGAs for AI inference. For example, Microsoft has used FPGA to build their Brainwave AI service [1]-[2]. The key calculation that the Brainwave-like inference engines use FPGAs for is low-precision multiplication. Therefore, it is important to be able to implement low-precision multipliers in FPGA in the most optimal way.

Quartus compiler significantly improved quality of multiplier synthesis with the recent introduction of Fractal synthesis [3].

But still a question remains whether low precision multipliers generated by Quartus are optimal.

This paper answers that question by applying SAT-based exact synthesis techniques [4]-[6] to FPGA synthesis. SAT-based exact synthesis solves synthesis problem by reformulating it as SAT problem and then applying efficient SAT/SMT solvers. It can build an optimal circuit and prove that a better circuit does not exist.

The rest of this paper is organized as follows. In section II we provide an overview of exact synthesis technique. In section III we describe an architecture of FPGA ALM block and formulate practical ALM synthesis constraints. In section IV we formally describe ALM exact synthesis problem. In section V we introduce ExactS tool [7] that applies exact synthesis technique to generate an optimal ALM-based circuit for the given problem. In section VI we provide an overview of low-precision multiplier circuits generated with the help of ExactS.

II. BACKGROUND

A. Exact synthesis problem

Let $\mathrm{B} = \{0, 1\}$ and B^k is the k-th cartesian product of B for any $k, k > 1$. In this paper, we are interested in Boolean operators $F : \mathrm{B}^n \longrightarrow B^m$ that map n input truth values to m output truth values. Logic synthesis is the problem of finding an optimal circuit in a given class of circuits and for a given Boolean operator F with respect to selected optimality criterion (e.g., complexity or depth) or their combination. Sometimes logic synthesis is considered as constrained optimization problem, when specific optimality criterion is optimized with several other criteria constrained.

Logic synthesis problem is known to be NP-hard. Consequently, practical logic synthesis problems are usually solved using heuristic algorithms and rarely guarantee optimal solutions.

Exact synthesis aims at finding optimal solution of the given logic synthesis problem. The problem is considered solved, when optimal circuit is found, and we have proof that a circuit with better complexity parameters does not exist. Usually, exact synthesis is possible when a Boolean operator F has a very small number of inputs n and outputs m, or when a considered class of circuits has very strict constraints.

B. Related work on Exact synthesis

Exact synthesis algorithms typically fall into one of the following three categories [8]:

1. algorithms based on functional decomposition [9]-[12];
2. algorithms based on explicit [13]-[16] or implicit enumeration [4]-[5], [17]-[21];
3. hybrid approaches [8], [22]-[23].

In recent years, SAT-based exact logic synthesis became one of the dominant techniques. This approach is based on explicit enumeration and its main idea is to represent initial logic synthesis problem as a decision problem and then encode it as a SAT instance, which is solved by SAT or SMT solver. Popularity of this approach is dictated by both recent advance in SAT algorithms [18] and flexibility of the SAT encodings, which can be used to encode complex circuit types and constraints [24].

Significant part of recent results in exact synthesis is related to complexity classification of Boolean functions with small number of variables (typically, no more than 5) for different classes of circuits. Boolean circuits (some authors call them Boolean chains) [4]-[6], [15], [25] and different types of inverter graphs (subclasses of Boolean circuits) [26]-[30] attracted the most attention.

Other classes, such as exclusive-or sum-of-products forms [31], were also considered. Exact synthesis was also applied to synthesis of specific Boolean operators [32].

Libraries of optimal circuits, which result from this classification, are used in different applications, such as logic synthesis [27] and Boolean matching [33]. These libraries are also used to improve theoretical complexity bounds for specific Boolean functions and operators [34]. Recently, research of other complexity measures began in the scope of exact synthesis. For example, exact synthesis of delay optimal circuits is considered in [35]-[36].

On the final note, exact synthesis research results in a number of different academic tools, such as ABC [37] ('exact', 'twoexact', 'lutexact', and 'majexact' commands), CirKit [38] and Percy [39], and libraries [40].

C. SAT-based exact synthesis

The main idea of the SAT-based exact synthesis described in [6] is to reduce synthesis problem to Boolean satisfiability problem (SAT problem). As NP problem the synthesis problem can be efficiently reduced to NP-complete SAT problem. The statement of the SAT problem is to find variable values for which given Boolean function is true or provide the proof that such set of values does not exist. Further we will give more details about rewriting synthesis problem to the SAT problem.

Consider the logic network synthesis problem for certain m Boolean functions $f_1(x_1, \dots, x_n), \dots, f_m(x_1, \dots, x_n)$ of n variables. Logic network is a directed, acyclic graph that consists of input nodes marked by variable names, output nodes, other nodes marked with symbols of logic gates, and edges which represent connections between nodes.

To reduce the synthesis problem to the SAT problem we need to define some sets of variables and constraints between them. Initially we don't have a logic network, we have only input nodes, output nodes and a set of unconnected logic gates so their inputs and outputs are independent. Also, we haven't defined Boolean functions which are implemented in logic gates yet. The first set of variables V consist of 2^n variables $v_1^{(w)}, \dots, v_{2^n}^{(w)}$ for each input node, output node and inputs and outputs of the logic gates w. These variables represent truth tables of Boolean functions which are implemented at corresponding nodes. The second set of variables P defines Boolean functions for each logic gate in the network. Next, we introduce three kinds of constraints:

1. Input and output constraints. Truth tables for n network inputs and m network outputs should be equal to truth tables of corresponding input variables $x_1, \dots, x_n$ and functions $f_1, \dots, f_m$.
2. Connection constraints. Logic gates and outputs of the logic network should be connected to other gates or to inputs of the network. We also can restrict the set of available connection for each specific node. This type of constraints can be written as follows:

   $$\forall i \left(v_i^{(w)} = v_i^{(u_1)}\right) \vee \dots \vee \forall i \left(v_i^{(w)} = v_i^{(u_k)}\right),$$

   where logic gate input or network output w can be connected to one of nodes $u_1, \dots, u_k$.
3. Functionality constraints. For each logic gate in the network, we describe its functionality. For example, for AND gate with inputs u, v and output w we get the following expression: $\forall i \left(v_i^{(w)} = v_i^{(u)} \,\&\, v_i^{(v)}\right)$.

   For logic gates with undefined Boolean functions, we also use variables from the set P in the constraint.

All constraints that are described above form Boolean formula, which satisfies if and only if there is a logic network with given set of logic gates and connection restrictions that implements system of functions $f_1, \dots, f_m$. Moreover, if we have the solution of defined SAT problem, we can recover this network using values of variables from sets V and P. On the other hand, if SAT solver reports that the SAT problem does not have a solution it is a proof that desired logic network doesn't exist.

III. Stratix10 ALM structure and synthesis constraints

In this paper we address exact synthesis problem for Intel Stratix 10 ALM-based circuits. To the best of our knowledge this is the first application of exact synthesis to commercial FPGA architecture. ALM is a basic Stratix 10 FPGA building block [41]. It could be represented as a Boolean circuit with 9 inputs, 6 outputs, 11 basic logic gates, such as AND, OR, XOR, and MUX, and 4 LUT4-blocks as shown on fig. 1. This ALM structure is derived from Stratix 10 architecture specification [41] and simulation models. There are 8 "primary" inputs (a, b, c0, d0, c1, d1, e, f) and 5 "primary" outputs (lut5out0, lut5out1, lut6out, sumout0, sumout1). All primary ports are connected to the routing fabric. There is also "carry-in" input (cin) and "carry-out" output (cout) not connected to the routing fabric. ALMs are grouped into 10-ALM chains in such a way that cin input of the first ALM in a chain is connected to 0, while cin of any other ALM is connected

to cout of the previous ALM. These chains are called FPGA logic array blocks (LABs). Since we consider only combinational circuits, ALM's registers and related routing is discarded. Furthermore, we add additional constraint, that no more than 4 out of 5 primary outputs can be used simultaneously, because Quartus Compiler appears to not able to route ALMs with all 5 outputs connected. We consider the block to have unit complexity and unit delay between any pair of primary input and output. Delay between cin and cout is very small and considered to be equal to zero in our model.

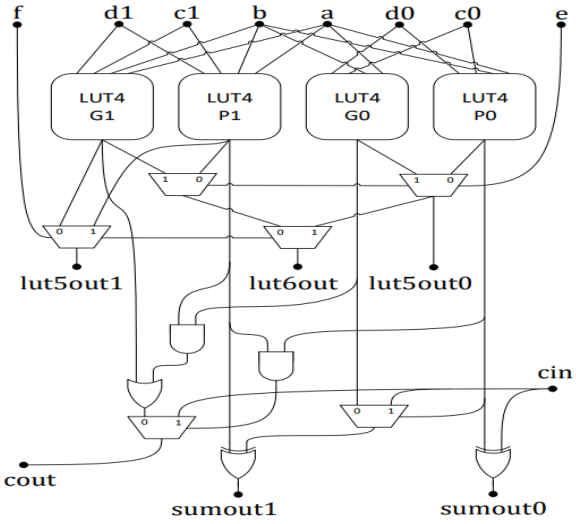

Fig. 1. Internal structure of Stratix10 ALM block

ALM is rather large circuit. For more precise complexity measurements it is convenient to define a smaller building block, which we call "half ALM". Half ALM (which is basically an extension of Stratix 10 arithmetic LCELL primitive) is defined as an ALM where no more than 4 primary inputs (either a, c0, d0, e, or b, c1, d1, f) and two primary outputs (either lut5out0, sumout0, or lut5out1, sumout1) are used, and only two LUT4 blocks (either G0, P0, or G1, P1) are configured. It is possible to implement two half-ALMs within a single ALM. Half-ALM is modelled by imposing additional constraints on the ALM configuration.

IV. Problem Statement

In this section ALM-based exact synthesis problem is formulated. For a given Boolean operator $F(x_1,\ldots,x_n)$: $\mathrm{B}^n \longrightarrow \mathrm{B}^m$ we seek to determine minimal number r, which allows to implement operator F (input and output inversions are allowed) using a chain of r ALMs (the last element of the chain can be half-ALM).

Following constraints are imposed on ALM's inputs and outputs:

1. each ALM input can be connected to one of the primary inputs $x_1,\ldots,x_n$, constants 0 or 1 or one of the ALM's outputs
2. each ALM output can be connected to one of the primary outputs $f_1,\ldots,f_m$ or one of the ALM's inputs.

Considered ALM-based circuit does not contain directed cycles (i.e. we consider purely combinational circuits). Furthermore, carry in port cin of the first ALM and carry out port of the last ALM in the chain should be connected to the constant 0. This additional constraint allows stacking of generated ALM-based circuits, which is needed in order to pack great number of identical circuits to FPGA. It should be noted, that for some Boolean operators value r may be greater than size of the LAB. In this case ALM-based circuit may be composed of several ALM chains, which are placed in nearby LABs.

Using the technique described in section 2 we can reformulate this FPGA synthesis problem as SAT problem. Note that the difference between arbitrary logic network synthesis and FPGA synthesis is that a lot of logic gates are already connected and has predefined Boolean functions. It significantly simplifies synthesis problem so this exact synthesis method can be applied to functions of more than 4 or 5 variables and to bigger networks.

V. Implementation

We implemented FPGA exact synthesis tool that we called ExactS that solves the problem specified in section 4 and is based on the approach described in section 2. The tool is written in Python and is using Z3 SMT Solver [42] under the hood. The tool takes a description of FPGA ALM architecture (also written in Python), and a circuit template in Verilog with custom hints and extensions describing the synthesis problem. The circuit template defines a system of functions to be implemented, as well as the circuit topology and configuration constraints. The output of ExactS is either a circuit, which implements the specified system of functions, or a proof of unsatisfiability (see fig. 2).

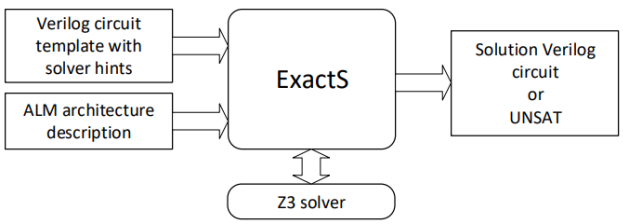

Fig. 2. ExactS flow diagram

VI. Results

We applied ExactS to generate low precision unsigned multipliers for Stratix10 and Agilex Intel FPGA architectures. We chose to focus on multipliers because those circuits are very important for FPGA AI inference accelerators and might have direct impact on accelerator throughput. But the tool setup can be easily modified for other Boolean operators. All the generated circuits were simulated and pushed through Quartus compiler to make sure these are legal circuits. Experiments show that ExactS works well for circuits with up to 6-8 inputs, 6-8 outputs, and complexity up to 3-4 ALMs, but for bigger circuits we face very long runtimes because of exponentially growing complexity of the SAT problem.

Also, we noticed that proofs of unsatisfiability require orders of magnitude more time.

Stratix 10 results are summarized in tab. 1 (Agilex results are very similar). "NxM" stands for unsigned multiplier with N-bit input A and M-bit input B (and N+M-bit output). Circuits generated by ExactS are compared with circuits generated by Quartus Compiler 20.3. The data in the table demonstrates that the complexity of multipliers 2x2, 2x3, 2x4, 2x5, 2x6, 3x3 is 10-50% better if compared with multipliers generated by Quartus. Also, for 3x3, 3x4, 3x5 multipliers we got circuits with better depth. Fig. 3 and fig. 4 demonstrate 2x2 and 3x3 multipliers generated by ExactS.

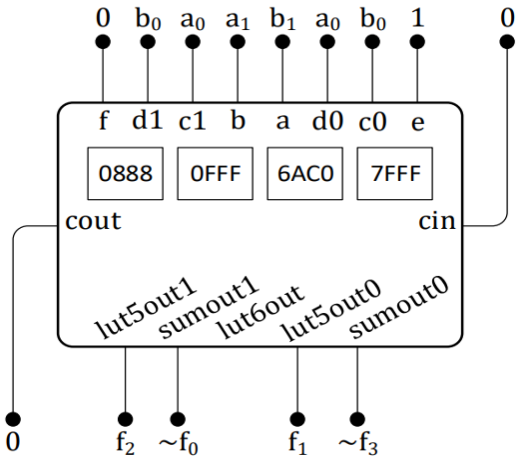

Fig. 3. 2x2 multiplier generated by ExactS

Table 1

Depth and complexity results for Stratix10 multipliers

	Quartus		SAT/SMT		UNSAT		Runtime		Improvement complexity
	ALMs	Depth	ALMs	Depth	ALMs	Depth	SAT	UNSAT	
2x2	2	1	1	1	—		1.5 s.	—	50%
2x3	2.5	1	2	1	1.5	1	1.9 s.	3.3 s.	20%
2x4	3	1	3 2.5	1 2	2.5	1	7 m. 25 s. 8.2 s.	11 m. 19 s.	17%
2x5	3.5	1	3.5 3	1 2	3	1	2 m. 27 s. 45.7 s.	2 h. 49 m.	14%
2x6	4.5	1	4	1	3.5	1	1 m. 59 s.	8 h. 56 m.	11%
3x3	4	2	3.5 3	1 2	3	1	1 m. 12 s. 30 s.	2 h. 29 m.	12.5% 25%
3x4	5	2	7	1	—		15 m. 34 s.	—	—
3x5	6	2	12	1	—		3 h. 57 m.	—	—

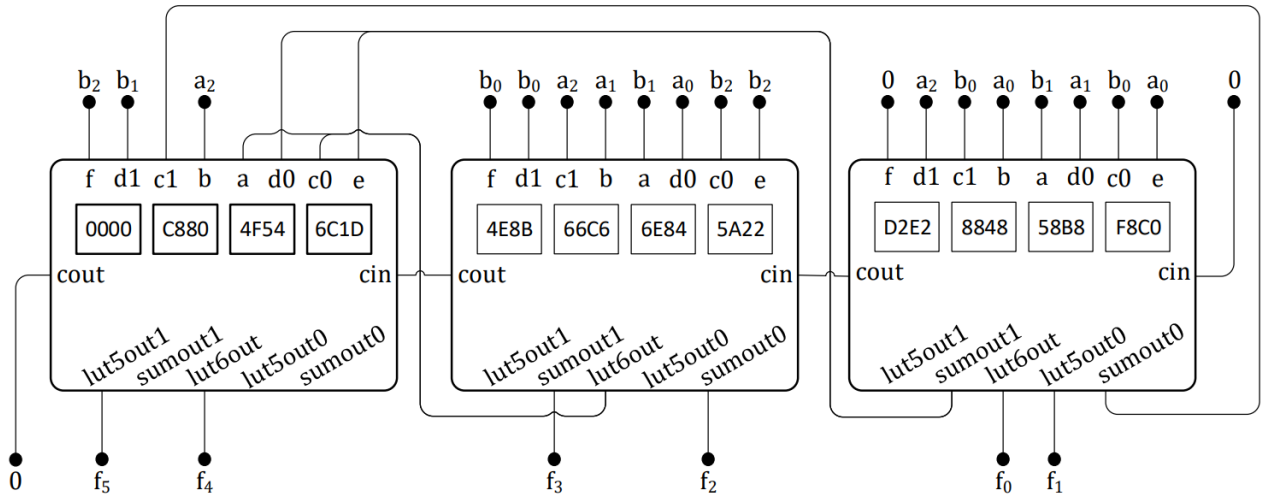

Fig. 4. 3x3 multiplier generated by ExactS

ExactS is also able to prove that a circuit with specific parameters doesn't exist (the UNSAT result). So, for some multipliers we proved that it's impossible to get a circuit with better complexity or depth. Thus, for 2x3, 2x4, 2x5, 3x3 multipliers circuits with the optimal complexity and depth 1 were designed.

The main limitation of the tool is that the runtime of a SAT solver grows exponentially with the number of inputs, and so for bigger multipliers (3x6, 4x4, 4x5 ...) the runs didn't finish (we set the runtime limit to one week).

VII. Future work

In the future, it is planned to improve ExactS tool so that it can be used to generate optimal circuits for other Intel FPGA families (Arria 10, Stratix V, etc.) as well as for FPGAs from other vendors (Xilinx, Lattice, etc.). We are also working on ExactS runtime improvement to be able to deal with bigger circuits.

VIII. Conclusion

In this paper we apply Exact Synthesis technique to synthesize optimal circuits for Intel Stratix10 FPGA. We developed a tool that generates optimal FPGA circuits for logic functions with a small number of variables. We used the tool to build optimal circuits for several low-precision multipliers and proved their optimality. A number of circuits generated by our tool is smaller than the corresponding circuits generated by Quartus Compiler. Our results could be used to improve performance of FPGA-based AI inference accelerators.

Funding

This work was supported by the Intel corporation.

References

[1] Fowers J. et al. A configurable cloud-scale DNN processor for real-time AI // 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA). – IEEE, 2018. – P. 1-14.

[2] Chung E. et al. Serving dnns in real time at datacenter scale with project brainwave // IEEE Micro. – 2018. – V. 38. – №. 2. – P. 8-20.

[3] Langhammer M., Baeckler G., Gribok S. Fractal synthesis: Invited tutorial // Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. – 2019. – P. 202-211.

[4] Kojevnikov A., Kulikov A. S., Yaroslavtsev G. Finding efficient circuits using SAT-solvers // International Conference on Theory and Applications of Satisfiability Testing. – Springer, Berlin, Heidelberg, 2009. – P. 32-44.

[5] Knuth D. E. The art of computer programming, Volume 4, Fascicle 6: Satisfiability. – Addison-Wesley Professional, 2015.

[6] Soeken M. et al. Practical exact synthesis // 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE). – IEEE, 2018. – P. 309-314.

[7] ExactS FPGA Exact synthesis tool. [Online]. Available: https://mks2.cs.msu.ru/root/intel_altera

[8] Ernst E. A. Optimal combinational multi-level logic synthesis. – University of Michigan, 2009.

[9] Karp R. M. et al. A computer program for the synthesis of combinational switching circuits // 2nd Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1961). – IEEE, 1961. – P. 182-194.

[10] Roth J. P., Karp R. M. Minimization over Boolean graphs // IBM journal of Research and Development. – 1962. – V. 6. – №. 2. – P. 227-238.

[11] Schneider P. R., Dietmeyer D. L. An algorithm for synthesis of multiple-output combinational logic // IEEE Transactions on Computers. – 1968. – V. 100. – №. 2. – P. 117-128.

[12] Lawler E. L. An approach to multilevel Boolean minimization // Journal of the ACM (JACM). – 1964. – V. 11. – №. 3. – P. 283-295.

[13] Smith R. A. Minimal three-variable NOR and NAND logic circuits // IEEE Transactions on Electronic Computers. – 1965. – №. 1. – P. 79-81.

[14] Drechsler R., Günther W. Exact circuit synthesis // In Int'l Workshop on Logic Synth. – 1998.

[15] Knuth D. E. The art of computer programming, volume 4A: combinatorial algorithms, part 1. – Pearson Education India, 2011.

[16] Hellerman L. A catalog of three-variable or-invert and and-invert logical circuits // IEEE Transactions on Electronic Computers. – 1963. – №. 3. – P. 198-223.

[17] Muroga S., Ibaraki T. Design of optimal switching networks by integer programming // IEEE Transactions on Computers. – 1972. – V. 100. – №. 6. – P. 573-582.

[18] Eén N. Practical SAT-a tutorial on applied satisfiability solving // Slides of invited talk at FMCAD. – 2007.

[19] Baugh C. R., Ibaraki T., Muroga S. Results in Using Gomory's All-Integer Integer Algorithm to Design Optimum Logic Networks // Operations Research. – 1971. – V. 19. – №. 4. – P. 1090-1096.

[20] Baugh C. R. et al. Optimal networks of NOR-OR gates for functions of three variables // IEEE Transactions on Computers. – 1972. – V. 100. – №. 2. – P. 153-160.

[21] Lai H. C. et al. Minimization of logic networks under a generalized cost function // IEEE Transactions on Computers. – 1976. – V. 100. – №. 9. – P. 893-907.

[22] Davidson E. S. An algorithm for NAND decomposition under network constraints // IEEE Transactions on Computers. – 1969. – V. 100. – №. 12. – P. 1098-1109.

[23] Culliney J. N. et al. Results of the synthesis of optimal networks of AND and OR gates for four-variable switching functions // IEEE Transactions on Computers. – 1979. – V. 28. – №. 01. – P. 76-85.

[24] Testa E. et al. Exact synthesis for logic synthesis applications with complex constraints // Proceedings of the 26th International Workshop on Logic & Synthesis (IWLS). – 2017. – №. CONF.

[25] Haaswijk W. et al. Classifying functions with exact synthesis // 2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL). – IEEE, 2017. – P. 272-277.

[26] Soeken M. et al. Exact synthesis of majority-inverter graphs and its applications // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – 2017. – V. 36. – №. 11. – P. 1842-1855.

[27] Haaswijk W. et al. A novel basis for logic rewriting // 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC). – Ieee, 2017. – P. 151-156.

[28] Haaswijk W. et al. SAT based exact synthesis using DAG topology families // 2018 55Th Acm/Esda/Ieee Design Automation Conference (Dac). – IEEE, 2018. – P. 1-6.

[29] Chu Z. et al. Exact synthesis of boolean functions in majority-of-five forms // 2019 IEEE International Symposium on Circuits and Systems (ISCAS). – IEEE, 2019. – P. 1-5.
[30] Lozhkin S.A., Zizov V.S., Shupletsov M.S., Zhukov V.V., Khzmalian D.E., Belyankov O.O. On complexity of inverter graphs for Boolean functions of small number of variables // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2020. Issue 4. P. 95-102. doi:10.31114/2078-7707-2020-4-95-102 (in Russian)
[31] Riener H. et al. Exact synthesis of ESOP forms // Advanced boolean techniques. – Springer, Cham, 2020. – P. 177-194.
[32] Stoffelen K. Optimizing s-box implementations for several criteria using SAT solvers // International Conference on Fast Software Encryption. – Springer, Berlin, Heidelberg, 2016. – P. 140-160.
[33] Huang Z. et al. Fast Boolean matching based on NPN classification // 2013 International Conference on Field-Programmable Technology (FPT). – IEEE, 2013. – P. 310-313.
[34] Kulikov A. S. Improving circuit size upper bounds using sat-solvers // 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE). – IEEE, 2018. – P. 305-308.
[35] Amarú L. et al. Enabling exact delay synthesis // 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). – IEEE, 2017. – P. 352-359.
[36] Soeken M., De Micheli G., Mishchenko A. Busy man's synthesis: Combinational delay optimization with SAT // Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017. – Ieee, 2017. – P. 830-835.
[37] Brayton R., Mishchenko A. ABC: An academic industrial-strength verification tool // International Conference on Computer Aided Verification. – Springer, Berlin, Heidelberg, 2010. – P. 24-40.
[38] M. Soeken, The CirKit toolkit. [Online]. Available: https://github.com/msoeken/cirkit
[39] W. Haaswijk, The percy exact synthesis library. [Online]. Available: https://github.com/whaaswijk/percy
[40] Lozhkin S.A., Shupletsov M.S., Konovodov V.A., Danilov B.R., Zhukov V.V., Bagrov N.Yu. Distributed system and switching circuits optimization methods for Boolean functions of small number of variables // Problems of Perspective Micro- and Nanoelectronic Systems Development - 2016. Proceedings / edited by A. Stempkovsky, Moscow, IPPM RAS, 2016. Part 1. P. 40-47. (in Russian)
[41] Intel Stratix 10 logic array blocks and adaptive logic modules user guide. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/ug-s10-lab.pdf
[42] The Z3 Theorem Prover. [Online]. Available: https://github.com/Z3Prover/z3

Точный синтез умножителей малой точности для ПЛИС корпорации Intel

М.С. Шуплецов[1], В.В. Жуков[1], С. Грибок[2], И. Ганусов[2], М.А. Местецкий[1], М.А. Лопунов[1], Е.Д. Купраш[1]

[1]Московский Государственный Университет им. М.В. Ломоносова, г. Москва, shupletsov@cs.msu.ru

[2]Intel Corporation, Сан-Хосе, США, ilya.ganusov@intel.com

***Аннотация* — Создание эффективных ускорителей нейронных сетей на базе программируемых логических интегральных схем (ПЛИС) задает новые требования к компиляторам, работающим с ПЛИС. Ускорители нейронных сетей состоят из очень большого числа идентичных схем. Примером таких схем являются умножители малой точности. Именно поэтому, для того чтобы получить максимальную производительность необходимо строить эти небольшие схемы оптимальным образом. В данном случае оптимальность подразумевается с точки зрения размера синтезируемой схемы и ее задержки. Точный синтез, основанный на задаче ВЫПОЛНИМОСТЬ (англ. satisfiability, SAT) — известная и эффективная техника для построения оптимальных схем для функций алгебры логики с маленьким числом входов.**

В данной работе был применен метод точного синтеза к адаптивным логическим модулям (АЛМ) современных ПЛИС корпорации Intel. Для этого был разработан программный комплекс на базе SAT-решателя, который строит оптимальные схемы на основе АЛМ блоков архитектуры Stratix10 и Agilex. Применяя данный программный комплекс для синтеза оптимальных схем умножителей небольшой размерности, было показано, что новые схемы на 10–50% эффективнее с точки зрения размера по сравнению со схемами построенными Quartus Complier. Кроме того, в некоторых случаях удалось получить схемы, которые имеют меньшее значение задержки. В заключение отметим, что, насколько нам известно, в данной работе впервые методы точного синтеза были применены к архитектурам современных коммерческих ПЛИС.

***Ключевые слова* — точный синтез, умножители, Agilex, Stratix 10.**

ЛИТЕРАТУРА

[1] Fowers J. et al. A configurable cloud-scale DNN processor for real-time AI // 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture (ISCA). – IEEE, 2018. – P. 1-14.
[2] Chung E. et al. Serving dnns in real time at datacenter scale with project brainwave // iEEE Micro. – 2018. – V. 38. – №. 2. – P. 8-20.

[3] Langhammer M., Baeckler G., Gribok S. Fractal synthesis: Invited tutorial // Proceedings of the 2019 ACM/SIGDA International Symposium on Field-Programmable Gate Arrays. – 2019. – P. 202-211.

[4] Kojevnikov A., Kulikov A. S., Yaroslavtsev G. Finding efficient circuits using SAT-solvers // International Conference on Theory and Applications of Satisfiability Testing. – Springer, Berlin, Heidelberg, 2009. – P. 32-44.

[5] Knuth D. E. The art of computer programming, Volume 4, Fascicle 6: Satisfiability. – Addison-Wesley Professional, 2015.

[6] Soeken M. et al. Practical exact synthesis // 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE). – IEEE, 2018. – P. 309-314.

[7] ExactS FPGA Exact synthesis tool. [Online]. Available: https://mks2.cs.msu.ru/root/intel_altera

[8] Ernst E. A. Optimal combinational multi-level logic synthesis. – University of Michigan, 2009.

[9] Karp R. M. et al. A computer program for the synthesis of combinational switching circuits // 2nd Annual Symposium on Switching Circuit Theory and Logical Design (SWCT 1961). – IEEE, 1961. – P. 182-194.

[10] Roth J. P., Karp R. M. Minimization over Boolean graphs // IBM journal of Research and Development. – 1962. – V. 6. – №. 2. – P. 227-238.

[11] Schneider P. R., Dietmeyer D. L. An algorithm for synthesis of multiple-output combinational logic // IEEE Transactions on Computers. – 1968. – V. 100. – №. 2. – P. 117-128.

[12] Lawler E. L. An approach to multilevel Boolean minimization // Journal of the ACM (JACM). – 1964. – V. 11. – №. 3. – P. 283-295.

[13] Smith R. A. Minimal three-variable NOR and NAND logic circuits // IEEE Transactions on Electronic Computers. – 1965. – №. 1. – P. 79-81.

[14] Drechsler R., Günther W. Exact circuit synthesis // In Int'l Workshop on Logic Synth. – 1998.

[15] Knuth D. E. The art of computer programming, volume 4A: combinatorial algorithms, part 1. – Pearson Education India, 2011.

[16] Hellerman L. A catalog of three-variable or-invert and and-invert logical circuits // IEEE Transactions on Electronic Computers. – 1963. – №. 3. – P. 198-223.

[17] Muroga S., Ibaraki T. Design of optimal switching networks by integer programming // IEEE Transactions on Computers. – 1972. – V. 100. – №. 6. – P. 573-582.

[18] Eén N. Practical SAT-a tutorial on applied satisfiability solving // Slides of invited talk at FMCAD. – 2007.

[19] Baugh C. R., Ibaraki T., Muroga S. Results in Using Gomory's All-Integer Integer Algorithm to Design Optimum Logic Networks // Operations Research. – 1971. – V. 19. – №. 4. – P. 1090-1096.

[20] Baugh C. R. et al. Optimal networks of NOR-OR gates for functions of three variables // IEEE Transactions on Computers. – 1972. – V. 100. – №. 2. – P. 153-160.

[21] Lai H. C. et al. Minimization of logic networks under a generalized cost function // IEEE Transactions on Computers. – 1976. – V. 100. – №. 9. – P. 893-907.

[22] Davidson E. S. An algorithm for NAND decomposition under network constraints // IEEE Transactions on Computers. – 1969. – V. 100. – №. 12. – P. 1098-1109.

[23] Culliney J. N. et al. Results of the synthesis of optimal networks of AND and OR gates for four-variable switching functions // IEEE Transactions on Computers. – 1979. – V. 28. – №. 01. – P. 76-85.

[24] Testa E. et al. Exact synthesis for logic synthesis applications with complex constraints // Proceedings of the 26th International Workshop on Logic & Synthesis (IWLS). – 2017. – №. CONF.

[25] Haaswijk W. et al. Classifying functions with exact synthesis // 2017 IEEE 47th International Symposium on Multiple-Valued Logic (ISMVL). – IEEE, 2017. – P. 272-277.

[26] Soeken M. et al. Exact synthesis of majority-inverter graphs and its applications // IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems. – 2017. – V. 36. – №. 11. – P. 1842-1855.

[27] Haaswijk W. et al. A novel basis for logic rewriting // 2017 22nd Asia and South Pacific Design Automation Conference (ASP-DAC). – Ieee, 2017. – P. 151-156.

[28] Haaswijk W. et al. SAT based exact synthesis using DAG topology families // 2018 55Th Acm/Esda/Ieee Design Automation Conference (Dac). – IEEE, 2018. – P. 1-6.

[29] Chu Z. et al. Exact synthesis of boolean functions in majority-of-five forms // 2019 IEEE International Symposium on Circuits and Systems (ISCAS). – IEEE, 2019. – P. 1-5.

[30] Ложкин С.А., Зизов В.С., Шуплецов М.С., Жуков В.В., Хзмалян Д.Э., Белянков О.О. О сложности инверсных графов, реализующих булевы функции от малого числа переменных // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2020. Выпуск 4. С. 95-102. doi:10.31114/2078-7707-2020-4-95-102

[31] Riener H. et al. Exact synthesis of ESOP forms // Advanced boolean techniques. – Springer, Cham, 2020. – P. 177-194.

[32] Stoffelen K. Optimizing s-box implementations for several criteria using SAT solvers // International Conference on Fast Software Encryption. – Springer, Berlin, Heidelberg, 2016. – P. 140-160.

[33] Huang Z. et al. Fast Boolean matching based on NPN classification // 2013 International Conference on Field-Programmable Technology (FPT). – IEEE, 2013. – P. 310-313.

[34] Kulikov A. S. Improving circuit size upper bounds using sat-solvers // 2018 Design, Automation & Test in Europe Conference & Exhibition (DATE). – IEEE, 2018. – P. 305-308.

[35] Amarú L. et al. Enabling exact delay synthesis // 2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD). – IEEE, 2017. – P. 352-359.

[36] Soeken M., De Micheli G., Mishchenko A. Busy man's synthesis: Combinational delay optimization with SAT // Design, Automation & Test in Europe Conference & Exhibition (DATE), 2017. – Ieee, 2017. – P. 830-835.

[37] Brayton R., Mishchenko A. ABC: An academic industrial-strength verification tool // International Conference on Computer Aided Verification. – Springer, Berlin, Heidelberg, 2010. – P. 24-40.

[38] M. Soeken, The CirKit toolkit. [Online]. Available: https://github.com/msoeken/cirkit

[39] W. Haaswijk, The percy exact synthesis library. [Online]. Available: https://github.com/whaaswijk/percy

[40] Ложкин С.А., Шуплецов М.С., Коноводов В.А., Данилов Б.Р., Жуков В.В., Багров Н.Ю. Распределенная система и алгоритмы поиска минимальных и близких к ним контактных схем для булевых функций от малого числа переменных // Проблемы разработки перспективных микро- и наноэлектронных систем (МЭС). 2016. № 1. С. 40-47.

[41] Intel Stratix 10 logic array blocks and adaptive logic modules user guide. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/stratix-10/ug-s10-lab.pdf

[42] The Z3 Theorem Prover. [Online]. Available: https://github.com/Z3Prover/z3