Introduction to Exact Logic Circuit Synthesis

Kotaro MATSUOKA

Dec. 7th 2024

自己紹介

- 松岡 航太郎 (@nindanaoto)
- 京都大学大学院情報学研究科通信情報システムコース 佐藤研究室 博士課程 2 年
  - 集積回路の研究室
  - 端っこで準同型暗号を専門に生きている
- セキュリティキャンプ 講師 (2020〜)・開発コースアドバイザ (2022〜)
  - 完全準同型暗号 (TFHE) を教えている (資料:https://nindanaoto.github.io/
- 日本学術振興会特別研究員 DC1
  - 申請書: https://github.com/nindanaoto/DC1proposal
- NHK 学生ロボコン 2019 優勝 (機械研究会)
- 2019 年度未踏スーパクリエータ
  - 応募資料&成果報告資料: https://github.com/virtualsecureplatform/MitouDocument

Belgium Visit

Figure: ムール貝

Figure: Leuven Town Hall

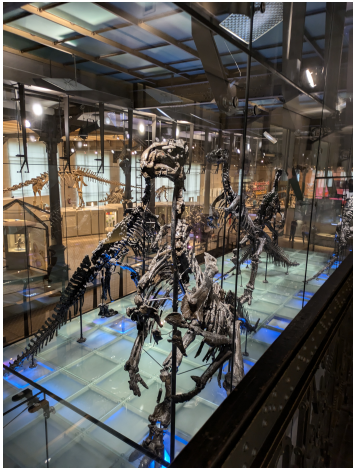

Figure: Iguanodon

What is Logic Synthesis?

- Logic Synthesis is the generation of the logic circuit from a high-level description
  - Hardware Design Language (HDL) is used to describe the circuit in high-level
    - e.g. Verilog, VHDL, Chisel, SpinalHDL, Veryl, etc.
  - Real circuits also require Place and Route (P&R) after this
    - Determine the position of gates in the real hardware
- Modern Logic Synthesis can be divided into three phase
  1. Conversion to normal representations
     - Most of the time this part is trivial
  2. Minimization of normal representation
     - Today's topic is a part of this
  3. Technology Mapping
     - Mapping the mathematical representation to actual physical entities like transistors
- Exact Synthesis will appear in later two phases

Normal Representations

- There are multiple well-known normal representations of Logic Circuit

1. Sum of Product (SOP)
2. Binary Decision Diagram (BDD)
3. And-Inverter-Graph (AIG)

Normal Representations

- There are multiple well-known normal representations of Logic Circuit

1. Sum of Product (SOP)
   - Also known as Disjunctive Normal Form(DNF, 選言標準形)
   - This is the most fundamental representation
   - The function is represented by AND of OR of some literals
     - eg. $a, a \wedge b, (a \wedge b) \vee (a \wedge \neg c \wedge d)$
   - OR of AND version is POS or Conjunctive Normal Form (CNF, 連言標準形)
   - This can be extended to Galois Field Elements
     - Galos Field Sum of Product (GFSOP)
     - Some works use this to synthesize advanced circuits
     - e.g.: Multi-valued quantum, reversible, CNT circuits
   - Pros: Easy to understand as a Boolean Algebra
   - Cons: Only two-layer one-output function is representable
2. Binary Decision Diagram (BDD)
3. And-Inverter-Graph (AIG)

Normal Representations

- There are multiple well-known normal representations of Logic Circuit

1. Sum of Product (SOP)
2. Binary Decision Diagram (BDD)
   - This is one of the most historical logic circuit representations
     - Shanon's first logic circuit representation is very similar to this
     - More precisely, Zero suppressed BDD (ZDD) is the closest one
   - BDD is the tree of MUXs, all primary inputs are connected to only selectors of MUXs
   - We can extend BDD to multi-output by partitioning primary inputs
     - Share some tree for shared inputs
   - Pros: The minimization of BDD is easy to formalize as a mathematical problem
     - Reduced Ordered BDD (ROBDD) is one of the well-known ways
     - ROBDD is also usable as a canonical representation (Function equivalence)
   - Cons: Representation size of some common function can be exponential
     - e.g.: Multiplication
     - If we avoid ordered BDD, multiplication can be smaller
3. And-Inverter-Graph (AIG)

Normal Representations

- There are multiple well-known normal representations of Logic Circuit

1. Sum of Product (SOP)
2. Binary Decision Diagram (BDD)
3. And-Inverter-Graph (AIG)
   - This is currently the most common representation
   - The standardized format is AIGER [1]
   - The graph of AND gates and Inverters
     - This is the complete set for any functions
   - Sometimes, we extend AIG to get a more compact representation
     - eg. Xor-And-Graph (XAG) [2], Majority-Inverter-Graph (MIG) [3]
   - Pros: Almost direct representation of CMOS circuits
   - Cons: Minimization algorithm is highly non-trivial

AIG minimization

- Objective: Minimizing the size of AIG (the number of gates)
- The most fundamental idea is *Divide-and-Conquer*
  - Cutting AIG to small sub-AIG and applying some local optimization
  - Sometimes called as *Peephole Optimization*
- The extraction of sub-AIG is formalized as *cut enumeration*
  - Enumerating subgraphs satisfying some suitable properties (e.g., Limited nodes, inputs, outputs)
- The Local minimization can be classified into three (AFAIK)

1. Converting into Other Formats
   - SOP, BDD, etc.
   - e.g., Quine – McCluskey Method, ESPRESSO [4]
2. Heursistic Method
   - There are some heuristics like eliminating redundant nodes, matching with known minimal circuits, etc.
   - e.g. Transduction [5]
3. Exact Synthesis
   - Today's main topic

SAT-based Exact Synthesis

- Exact Synthesis is the idea to directly treat minimization as NP hard problem
  - Area optimal minimization is known as NP-hard [6]
- SATisfiability (SAT) is a famous NP-complete problem
  - Historically known as the first NP-complete problem (Cook – Levin theorem) [7]
- SAT-based Exact Synthesis encodes the problem into SAT
  - Since SAT is NP-complete but the problem is NP-hard, we iteratively solve SAT
  - Other NP-complete problem like Integer Linear Programming (ILP) [8] is another possible option

SAT

- SAT is the problem of finding the input assignment that makes the output true
  - The circuit is given as CNF (POS)
  - If the maximum number of literals for OR operation is k, called k-SAT
    - The parentheses with OR operations are called *clause*
    - e.g.: $(\neg x_1 \vee x_2) \wedge (\neg x_1 \vee x_3) \wedge (x_1 \vee \neg x_2 \vee \neg x_3)$ is 3-SAT problem
- Offtopic
  - k-SAT $\rightarrow$ 3-SAT: Polynomial time reduction
  - 3-SAT: NP-complete
  - 2-SAT: Solvable in linear time
  - MAX 3-SAT: NP-hard
    - MAX k-SAT is the optimization problem to find inputs that makes maximal clause true.
  - MAX 2-SAT: NP-complete
    - QUBO friendly representation

SAT solver

- Because SAT is a well-known problem, we have a yearly competition
  - The International SAT Competition at SAT Conference `https://satcompetition.github.io/`
- There are several famous SAT solvers
  - MiniSAT: The ancestor of many SAT solvers
    - We rarely use this for practical cases, but it is easy to integrate
  - CaDiCAL [9]: Defact (not the best) standard of SAT solver
    - Supporting Incremental SAT solving
  - CryptoMiniSat [1]: The SAT solver for Crypto, supporting direct treatment of XOR
    - Supporting multi-core but parallelization in SAT will not help in general case
  - Glucose: Used in ABC synthesis tool
    - I don't know much but you can find the name in some ABC commands
  - Kissat [10]: The winner of the 2024 competition.
    - The author of CaDiCAL is developing this

[1] `https://github.com/msoos/cryptominisat`

Boolean Chain

- To encode to SAT, we need to express the circuit as SAT-like formula
- Boolean chain is originally introduced by Knuth [11]
- Boolean chain is a DAG to represent the logic circuit.
  - We can use this to represent any fixed-size boolean circuit

Ex. Full Adder

$$x_4 = x_1 \wedge x_2 \quad (1)$$
$$x_5 = x_1 \oplus x_6 \quad (2)$$
$$x_6 = x_3 \wedge x_5 \quad (3)$$
$$x_7 = x_3 \oplus x_5 \quad (4)$$
$$x_8 = x_4 \vee x_6 \quad (5)$$
$$l(1) = 7 \quad (6)$$
$$l(2) = 8 \quad (7)$$

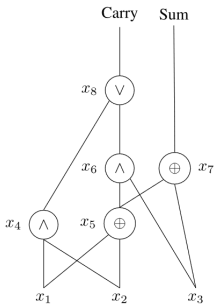

Figure: From Fig. 1 in [12]

Possible Encoding

- There are three main encodings in [12], SSV, MSV, DITT
  - Single Selection Variable(SSV)
  - Multiple Selection Variables (MSV)
  - Distinct Input Truth Tables (DITT)
- The difference between them is how we encode the connection between gates
  - The *selection variables*
- I will explain only MSV encoding
  - Just because I'm using this for my research

Multiple Selection Variables Encoding (Variables)

- Here, I define the variables used in MSV

1. n: The number of primary inputs (inputs to the circuit)
2. r: The number of gates in the circuit
3. $x_{ij}, i \in [0, n+r), j \in [0, 2^n)$: The wire values
   - $i < n$ is primary inputs, and the rest are gates' outputs
   - j means the actual value of primary inputs encodes as binary intger
     - $x_{ij} = (j >> i)\&1$ for $i < n$
4. $s_{ij}, i \in [0, r) j \in [0, n+i)$: The selection variable.
   - If x_j is connected to gate i, s_{ij} is true, false otherwise.
5. $f_{ijk}, i \in [0, r), j, k \in \mathbb{B}$: The truth table of the gates
   - If the gate i is true for inputs j, k, then f_{ijk}, false otherwise
6. $o_{ij}, i \in [0, m), j \in [0, r)$: The selection variables of the primary outputs
   - If the i-th primary output is the output of j-th gate ($x_{(j+n),:}$), o_{ij} is true.

Multiple Selection Variables Encoding (Main Clause)

- The main clause of this encoding for a two-input boolean gate is like this.[2]

$$\bigwedge_{a=0}^{1}\bigwedge_{b=0}^{1}\bigwedge_{c=0}^{1}\left(\bar{s}_{ij}\vee\bar{s}_{ik}\vee(x_{it}\oplus a)\vee(x_{jt}\oplus b)\vee(x_{kt}\oplus c)\vee(f_{ibc}\oplus\bar{a})\right) \tag{8}$$

- If wire j, k are selected for the gate i, $\bar{s}_{ij}\vee\bar{s}_{ik}$ becomes false.
- $(x_{it}\oplus a)\vee(x_{jt}\oplus b)\vee(x_{kt}\oplus c)$ will be 0 iff each value is a, b, c, respectively.
- $(f_{ibc}\oplus\bar{a}$ means the gate output equals to the output wire value.
- We need some other constraints, but I omit that for today
  - Constraints related to o_{ij}.
  - Symmetry Breaking Terms [12]

[2] I guess that a can be removed

Strategy of SAT-based Exact Synthesis

- This encoding gives the assignment for r gates' circuit
  - Not a minimum r, r is given parameter
- Starting from $r = 0$, we increase r until we get the satisfiable assignment
  - If there is no assignment, the SAT solver gives UNSAT as the result
- We can make sure that r for the final answer is the *exactly* minimal one

Known Results: 5-input-1-output functions

- Famous theoretical result by Prof. Knuth [11] using SAT-based Exact Synthesis
- Q: How many two-input gates are enough to implement 5-input-1-output functions?

Known Results: 5-input-1-output functions

- Famous theoretical result by Prof. Knuth [11] using SAT-based Exact Synthesis
- Q: How many two-input gates are enough to implement 5-input-1-output functions?
- A: 12
- Prof. Knuth solved most of the easy cases by heuristics and a few with exact synthesis
  - There is only one 5-input-1-output function that requires 12 gates
- For any of these functions, we can provide the database of minimum circuits[3]

[3] `https://gitlab.com/apgoucher/optimal5`

Known Results: IWLS2023

- In IWLS 2023, eSLIM [13] got second (third[4]) place for competition
  - eSLIM is the SAT-based exact synthesis method
  - eSLIM is developed by TU Wien Univ., Austria
  - This competition is the one for the AIG minimization algorithm

	Team EPFL	Google DeepMind	TU Wien	ours
total nodes	40071	23815	33802	24847
wins	17	63	33	56
uniq. wins	1	36	6	28
score	**7694.87**	**9737.37**	**8487.98**	**9291.43**

Figure: From `https://www.iwls.org/contest/2023/iwls23-contest.pdf`

[4]Prof. Alan's internal baseline was the real second place.

Coclusion

- SAT-based Exact Synthesis is one of the logic circuit synthesis methods
  - Finding exactly the minimum logical function
- Empirically, the number of gates should be around 12 for practical synthesis [14]
- What I missed today:
  - Symmetry breaking terms [12]
    - Terms to reduce the search space without loss of generality
  - Some heuristics
    - Parital DAG [12], Fence [12], [15], Incremental solving [16]
  - Cut enumeration algorithm [17], [18]
    - Required to partition circuits into under 12 gates subcircuits.
- Possible research directions:
  - Speeding up SAT by Quantum computing
  - Applying this to TFHE circuits (Submitted to DAC)
- The advancement of synthesis is the foundation of today's society!
- The best further reading: [12]

Reference I

[1] A. Biere, K. Heljanko, and S. Wieringa, "AIGER 1.9 and beyond," en, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria, Tech. Rep. 11/2, Jul. 2011. [Online]. Available: http://epub.jku.at/obvulioa/5973560 (visited on 12/06/2024).

[2] S. Liu, H. Zhou, Y. Xia, L. Wang, and Z. Chu, "Logic Synthesis for XOR-AND Graphs via Reed-Muller Representations," in *2024 Conference of Science and Technology for Integrated Circuits (CSTIC)*, Mar. 2024, pp. 1–3. DOI: 10.1109/CSTIC61820.2024.10532048. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10532048 (visited on 08/28/2024).

Reference II

[3] M. Soeken, L. G. Amarù, P.-E. Gaillardon, and G. De Micheli, "Exact Synthesis of Majority-Inverter Graphs and Its Applications," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 36, no. 11, pp. 1842–1855, Jan. 2017, Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, ISSN: 1937-4151. DOI: 10.1109/TCAD.2017.2664059. [Online]. Available: https://ieeexplore.ieee.org/document/7842552 (visited on 08/29/2024).

[4] H. Kanakia, M. Nazemi, A. Fayyazi, and M. Pedram, "ESPRESSO-GPU: Blazingly Fast Two-Level Logic Minimization," in *2021 Design, Automation & Test in Europe Conference & Exhibition (DATE)*, ISSN: 1558-1101, Feb. 2021, pp. 1038–1043. DOI: 10.23919/DATE51398.2021.9473961. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/9473961 (visited on 08/21/2024).

Reference III

[5] Y. Miyasaka, "Transduction Method for AIG Minimization," in *2024 29th Asia and South Pacific Design Automation Conference (ASP-DAC)*, ISSN: 2153-697X, Jan. 2024, pp. 398–403. DOI: 10.1109/ASP-DAC58780.2024.10473816. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10473816?casa_token=zPRX1E_ZZsMAAAAA:utrA9Yht3UvNNlGwLTVRo51OzbN7vpcqljCnSeF-Uf0Pp6ILib2fK0_oCZLeoVkB0oxOKsbhGOtb (visited on 11/02/2024).

[6] R. Ilango, B. Loff, and I. C. Oliveira, "NP-hardness of circuit minimization for multi-output functions," in *Proceedings of the 35th Computational Complexity Conference*, ser. CCC '20, Dagstuhl, DEU: Schloss Dagstuhl–Leibniz-Zentrum fuer Informatik, 2020, pp. 1–36, ISBN: 978-3-95977-156-6. DOI: 10.4230/LIPIcs.CCC.2020.22. [Online]. Available: https://doi.org/10.4230/LIPIcs.CCC.2020.22 (visited on 11/01/2024).

Reference IV

[7] S. A. Cook, "The complexity of theorem-proving procedures," in *Proceedings of the third annual ACM symposium on Theory of computing*, ser. STOC '71, New York, NY, USA: Association for Computing Machinery, 1971, pp. 151–158, ISBN: 978-1-4503-7464-4. DOI: 10.1145/800157.805047. [Online]. Available: https://dl.acm.org/doi/10.1145/800157.805047 (visited on 10/28/2024).

[8] A. Kojevnikov, A. S. Kulikov, and G. Yaroslavtsev, "Finding Efficient Circuits Using SAT-Solvers," en, in *Theory and Applications of Satisfiability Testing - SAT 2009*, O. Kullmann, Ed., Berlin, Heidelberg: Springer, 2009, pp. 32–44, ISBN: 978-3-642-02777-2. DOI: 10.1007/978-3-642-02777-2_5.

[9] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt, "CaDiCaL 2.0," en, in *Computer Aided Verification*, A. Gurfinkel and V. Ganesh, Eds., Cham: Springer Nature Switzerland, 2024, pp. 133–152, ISBN: 978-3-031-65627-9. DOI: 10.1007/978-3-031-65627-9_7.

Reference V

[10] A. Biere, T. Faller, K. Fazekas, M. Fleury, N. Froleyks, and F. Pollitt, "CaDiCaL, Gimsatul, IsaSAT and Kissat Entering the SAT Competition 2024," in *Proc. of SAT Competition 2024 – Solver, Benchmark and Proof Checker Descriptions*, M. Heule, M. Iser, M. Järvisalo, and M. Suda, Eds., ser. Department of Computer Science Report Series B, vol. B-2024-1, University of Helsinki, 2024, pp. 8–10.

[11] D. E. Knuth and D. E. Knuth, *Satisfiablility* (The art of computer programming / Donald E. Knuth Volume 4, Fascicle 6), en, Printing with corrections. Boston Columbus Indianapolis: Addison-Wesley, 2018, ISBN: 978-0-13-439760-3.

Reference VI

[12] W. Haaswijk, M. Soeken, A. Mishchenko, and G. De Micheli, "SAT-Based Exact Synthesis: Encodings, Topology Families, and Parallelism," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 39, no. 4, pp. 871–884, Apr. 2020, Conference Name: IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems, ISSN: 1937-4151. DOI: 10.1109/TCAD.2019.2897703. [Online]. Available: https://ieeexplore.ieee.org/document/8634910 (visited on 08/29/2024).

[13] F.-X. Reichl, F. Slivovsky, and S. Szeider, "eSLIM: Circuit Minimization with SAT Based Local Improvement," en, in *DROPS-IDN/v2/document/10.4230/LIPIcs.SAT.2024.23*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2024. DOI: 10.4230/LIPIcs.SAT.2024.23. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.SAT.2024.23 (visited on 08/28/2024).

Reference VII

[14] A. S. Kulikov, D. Pechenev, and N. Slezkin, "SAT-Based Circuit Local Improvement," en, in *DROPS-IDN/v2/document/10.4230/LIPIcs.MFCS.2022.67*, Schloss Dagstuhl – Leibniz-Zentrum für Informatik, 2022. DOI: 10.4230/LIPIcs.MFCS.2022.67. [Online]. Available: https://drops.dagstuhl.de/entities/document/10.4230/LIPIcs.MFCS.2022.67 (visited on 10/17/2024).

[15] L. Shang, S. Lu, S. Jung, and C. Pan, "Novel Fence Generation Methods for Accelerating Reconfigurable Exact Synthesis," in *2023 IEEE 66th International Midwest Symposium on Circuits and Systems (MWSCAS)*, ISSN: 1558-3899, Aug. 2023, pp. 506–510. DOI: 10.1109/MWSCAS57524.2023.10405994. [Online]. Available: https://ieeexplore.ieee.org/abstract/document/10405994 (visited on 10/07/2024).

Reference VIII

[16] S. Zou, J. Zhang, and G. Luo, "Incremental SAT-based Exact Synthesis," in *Proceedings of the Great Lakes Symposium on VLSI 2024*, ser. GLSVLSI '24, New York, NY, USA: Association for Computing Machinery, 2024, pp. 158–163, ISBN: 9798400706059. DOI: 10.1145/3649476.3658739. [Online]. Available: https://dl.acm.org/doi/10.1145/3649476.3658739 (visited on 10/15/2024).

[17] M. Yu, S. Carpov, A. Tempia Calvino, and G. De Micheli, "On the Synthesis of High-performance Homomorphic Boolean Circuits," in *Proceedings of the 12th Workshop on Encrypted Computing & Applied Homomorphic Cryptography*, ser. WAHC '24, New York, NY, USA: Association for Computing Machinery, Nov. 2024, pp. 51–63, ISBN: 9798400712418. DOI: 10.1145/3689945.3694803. [Online]. Available: https://dl.acm.org/doi/10.1145/3689945.3694803 (visited on 11/21/2024).

Reference IX

[18] O. Martinello, F. S. Marques, R. P. Ribas, and A. I. Reis, "KL-Cuts: A new approach for logic synthesis targeting multiple output blocks," in *2010 Design, Automation & Test in Europe Conference & Exhibition (DATE 2010)*, ISSN: 1558-1101, Mar. 2010, pp. 777–782. DOI: 10.1109/DATE.2010.5456946. [Online]. Available: https://ieeexplore.ieee.org/document/5456946 (visited on 09/23/2024).