

Assignment

Logic Synthesis and Optimization

Lecturer:

Prof. Zhufei Chu, <mailto:chuzhufei@nbu.edu.cn>

Drafted by:

Prof. Zhufei Chu

Jingren Wang, <mailto:jingrenwangcyber@gmail.com>



Note

- This assignment left a few lines of blank after each problem, but it might not be enough space for the answer and it is recommended to write on a separate sheet.
- You could reference open source projects to finish but answer should be written with clearly illustration.
- Example cases can be found at EPFL combinational benchmark [Amarù et al. \(2015\)](#).
- Referenced thesis in the assignment may contain SOTA research, but do not spend too much time on recommended thesis unless you finish the assignment first.
- For those who have no background in logic synthesis, finish reading chapter 6 of [Wang et al. \(2009\)](#) will be a good start before starting this assignment.

1 Introduction

Have a clear overview of the whole design flow helps in comprehending what happens under the hood and the reason why each step is needed.

1.1 Concepts

Some basic concepts or abbreviations you should know even after the introduction. Always check the concepts of the core idea in this course, it helps you understand in general.

Problem 1

- (a) What are the following terms abbreviated for:
EDA
VLSI
FPGA
- (b) Draw a whole design-flow flow chart, try to draw it as detailed as possible.
Check Figure 1.1 in Yosys manual [Wolf \(2024\)](#), compare with what you have drawn.
- (c) Can you map the idea to the language compiler?

1.2 Explore

Explore the active research, have your own way of finding basic information will help you solve problems by yourself, always try to find solutions by yourself, note that there is a very high possibility that there are people already asked or illustrated what you want to know on the internet.

Problem 2

- (a) Do you know any of the active open source project in EDA area?
- (b) Do you know where and how to efficiently find the related research reference? Try to search for key words Logic Synthesis, Synthesis, EDA, etc., see which conferences does the research always occur?

2 Representations of Logic Functions

2.1 Truth table

Truth table(TT) is essential in logic synthesis, pay good attention even if you think this is simple and easy to understand. Exact simulation is heavily used in logic synthesis tool ABC and mockturtle, you will see what happens behind the simulation method after solve following questions.

2.1.1 Direct view

This section gives a preview of TT, every answer should draw by hand, literal by literal, step by step.

Problem 3: TT related

- (a) Write down a full truth table for a 4 variable boolean function(complete boolean function), how many different combination of the elementary variables could you list here? And what about n variables? How many input combination can you get?
- (b) What you write above is the all possibilities of the elementary variables combination, from a function perspective, how many different functions could you produce under these combination? And why?
- (c) Can you show an example of an incomplete boolean function in TT? If so, can you try to simplify and show a better representation of this specific function in TT? If so, can you related the simplification to the don't care concept?

2.1.2 Coding Practice

After finishing writing, look what you have written, how do you write it? Is there any pattern on the full list? Write down the pattern you see, and if you are going to implement this TT in C/C++, what would you do? Can you find a way to optimize your implementation so that the TT could take as less memory as possible?

2.2 SOP/DNF & POS/CNF

These concepts are easy but vital in later boolean optimization. It is better to compare and remember, since these two concepts are contrary.

Problem 4: Expression

- (a) Write down the De Morgan's Law.
- (b) What is the concept of literal/clause under the boolean context? Give an example.
- (c) What are SOP/DNF & POS/CNF abbreviated for?
- (d) As we can see above, TT is canonical, now, try to give a boolean function f , prove that SOP representation is not canonical.
- (e) Give an example and illustrate the SOP in cubes context.

2.3 CNF & SAT solver

2.3.1 SAT

SAT solver could be integrated into logic synthesis tool to power the optimization. Know the basic concepts under the hood will help with a good use.

Problem 5: SAT

- (a) Explain why use CNF in SAT? Give an problem example that SAT can solve.
- (b) SAT is optimization problem or decision problem? Can you illustrate the difference between optimization problem and decision problem?
 - This question belongs to complexity classes, if you are not familiar, check 4.2.2.1 in [Wang et al. \(2009\)](#).

2.3.2 Davis-Putnam-Logemann-Loveland(DPLL)

DPLL is one of the typical Branch-and-bound algorithm, you should have good comprehension of what happens under the hood.

2.3.3 Explore

Problem 6: Self-study

- (a) Check DIMACS format, know what does each line mean.
- (b) Try to find well-known SAT solver, list them below.
- (c) Try to use one of them to solve the problem you wrote above in the previous subsection.

2.3.4 Coding Practice

Implemented in C/C++

Try to build a SAT solver yourself, the most simple version can be built by implementing exact simulation, which enumerates all the possible combination of the variables you listed above in the TT section.

1. You need a parser to DIMACS format.
2. Implement your solver.
3. Compare the time cost with any of the solver you find on internet.
4. Try to boost the efficiency of the solver using the method you have learnt during the class.

2.4 Binary Decision Diagram(BDD)

Binary Decision Diagram(BDD) is another clear expression of the boolean function, it is a powerful expression when modified to reduced and ordered.

Problem 7: BDD & ROBDD

- (a) What is BDD? Draw a BDD based on the TT you draw in the TT section.
- (b) Write down the definition(math formula) of Shanno expression, compare the definition and the BDD you draw, can you explain the definition based on the BDD?
- (c) So based on the previous question, if you are using simple MUXes to implement this whole BDD, can you show one upper bound of how many MUXes on the longest path?
 - This question helps understand Theorem 1, Bounds on Essential Critical Paths, THEORY OF EQUIOPTIMIZABLE ARRIVAL PATTERNS in [Amarú et al. \(2017\)](#).
- (d) Is BDD canonical? If not, give an example.
- (e) When reduce a BDD, list all the three rules.
- (f) Is ROBDD canonical?

2.5 And-Inverter-Graph(AIG)

And-Inverter Graph(AIG) is one of the core structures in ABC and mocturtle, SOTA optimization method such as rewrite, refactor, balance and resubstitution are all implemented based on AIG.

Problem 8: AIG format and representation

- (a) Search for the standard AIG format.
Check Prof. Armin Biere's report [Biere \(2007\)](#) on AIG format, there's also a new version [Biere et al. \(2011\)](#), but with only added up features, so pay attention to the old one.
- (b) What is structure hashing?
- (c) After applying structure hashing, do you think there are still two nodes with same fanin? If not, do you think there are still two node with same functionality? If so, why?
- (d) Is AIG canonical? Why? If not, give an example.

2.6 Majority-Inverter-Graph(MIG)

Majority graph is a novel representation, provides much more compact form than AIG, a direct overview of the representation of a case *f51m* from MCNC benchmark could be seen here [MIG \(2014\)](#). The application was originally for emerging technology, we won't discuss here, however, in logic optimization perspective, it could also be used as a tie breaker.

Problem 9: MIG representation

- (a) Give definition of n -input(n is odd) majority function.
- (b) As for an 3-input Majority gate support by a , b and c , see what would happen if the last variable c is equal to 0 or 1, what does the majority gate substitute to?
- (c) Based on the previous question, give the set relationship of the following homogeneous GIG(graph-inverter graph):
MIG, AOIG(AND/OR/INV) and AIG(AND/INV).

2.7 XOR-And-Grpah(XAG)

2.8 XOR-Majority-Graph(XMG)

2.9 Basic Concept

2.9.1 Transitive Fanin(TFI) Cone & Transitive Fanout(TFO) Cone

This is a basic concept but barely used directly, try to relate this to support variables from functional perspective.

- Also, try to think does it correct to define all the TFI leaves as support variables.

2.9.2 Nodes

There's not too much about nodes here instead of different types of representation or expression of the network holds different types of nodes. However, some basic concept such as **dag-node**, **tree-node**, will help you understand the concepts in cut(specifically factored cut) and Factored Form Literal Count(FFLC) under AIG context(You don't need to know about this now, we will discuss about it later.).

Problem 10: Nodes-type

- (a) Search for the definition of **dag-node** and **tree-node**. Draw a simple AIG and give example of both type of nodes.
 - This helps understanding the concept of factored cuts in [Mishchenko et al. \(2006\)](#) and FFLC in [Tempia Calvino et al. \(2023\)](#).

2.9.3 Cut

Cut method is used regularly in most of the optimization in logic synthesis since global optimization is hard and local based method is needed. Have a clear understanding of the following concepts help comprehend most of the SOTA research.

Problem 11: Cut in general

- (a) Give the definition of K -feasible cut and give an example on an AIG graph with $K = 4$.
- (b) How many 4 -feasible cut can you find on the AIG you provided?
- (c) If you want to find a better candidate that could replace this cut, in a functional perspective, how many candidates could you find for a 4 variable function?
- (d) If the amount of function is too large, can you suppress it?
 - This question is related to function classes, if you are interested, read [Huang et al. \(2013\)](#).
- (e) Can you filter out some of the cuts you provided from the previous question due to a threshold/criteria/cost function? Give an example.
 - This question helps understand the concept of **priority cut**, if you are interested, check [Mishchenko et al. \(2007\)](#).
 - There's another cut called reconvergence-driven cut, it **tries** to maximize the number of condition such that fanouts from the same nodes meets again. Try to give a definition of reconvergence-driven cut, and compare your definition with the one in [Mishchenko and Brayton \(2006\)](#). In addition, think why this is needed? And do you think this cut will always contain a reconvergence condition or just increase the possibility of containing one?
- (f) There are actually two ways to generate cuts, one is **bottom-up**, which is the most common one we see, another one is **top-down**(This two abstract definition is from [Testa \(2020\)](#).), which is the one we mentioned in node concept, called factored cuts. Try to think about what is the difference and the pros and cons of each method.

2.9.4 Maximum Fanout Free Cone(MFFC)

MFFC is essential, every time you have the chance to remove the pivot node, it allows you to remove the MFFC of the pivot.

Problem 12: MFFC

- (a) Give the definition of MFFC by selecting nodes in an example aig graph. You could try to use command `print_mffc` in ABC to check if you have selected the correct region.
 - Based on the definition of the MFFC, can you provide a definition on Maximum Fanout Free Window(MFFW)? Check [Zhu et al. \(2023\)](#) and see if you are correct.

2.10 Window

Window is essential in don't care based optimization.

Problem 13: Window

- a Try to draw one window in an AIG graph and give sedu-code on an algorithm that you would organize to automatically select the window in an AIG.

References

- (2014). Majority-inverter graph (mig). <https://www.epfl.ch/labs/lsi/page-102566-en-html/mig/>. Accessed: 2024-10-22.
- Amarú, L., Soeken, M., Vuillod, P., Luo, J., Mishchenko, A., Gaillardon, P.-E., Olson, J., Brayton, R., and De Micheli, G. (2017). Enabling exact delay synthesis. In *Proceedings of the 36th International Conference on Computer-Aided Design, ICCAD '17*, page 352–359. IEEE Press.
- Amarù, L., Gaillardon, P.-E., and De Micheli, G. (2015). The epfl combinational benchmark suite.
- Biere, A. (2007). The aiger and-inverter graph (aig) format version 20071012.
- Biere, A., Heljanko, K., and Wieringa, S. (2011). AIGER 1.9 and beyond. Technical Report 11/2, Institute for Formal Models and Verification, Johannes Kepler University, Altenbergerstr. 69, 4040 Linz, Austria.
- Huang, Z., Wang, L., Nasikovskiy, Y., and Mishchenko, A. (2013). Fast boolean matching based on npn classification. In *2013 International Conference on Field-Programmable Technology (FPT)*, pages 310–313.
- Mishchenko, A. and Brayton, R. K. (2006). Scalable logic synthesis using a simple circuit structure.
- Mishchenko, A., Chatterjee, S., and Brayton, R. (2006). Improvements to technology mapping for lut-based fpgas. In *Proceedings of the 2006 ACM/SIGDA 14th International Symposium on Field Programmable Gate Arrays, FPGA '06*, page 41–49, New York, NY, USA. Association for Computing Machinery.
- Mishchenko, A., Cho, S., Chatterjee, S., and Brayton, R. (2007). Combinational and sequential mapping with priority cuts. In *Proceedings of the 2007 IEEE/ACM International Conference on Computer-Aided Design, ICCAD '07*, page 354–361. IEEE Press.
- Tempia Calvino, A., Mishchenko, A., Schmit, H., Mahintorabi, E., Xu, X., and De Micheli, G. (2023). Improving standard-cell design flow using factored form optimization.
- Testa, E. (2020). *Data Structures and Algorithms for Logic Synthesis in Advanced Technologies*. PhD thesis, EPFL, Lausanne.
- Wang, L.-T., Chang, Y.-W., and Cheng, K.-T. T. (2009). *Electronic Design Automation: Synthesis, Verification, and Test*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA.
- Wolf, C. (2024). Yosys open synthesis suite.
- Zhu, X., Tang, R., Chen, L., Li, X., Huang, X., Yuan, M., Sheng, W., and Xu, J. (2023). A database dependent framework for k-input maximum fanout-free window rewriting. In *DAC*, pages 1–6.