Problem A: Reinforcement Logic Optimization for a General Cost Function

Chung-Han Chou, Kwangsoo-Han, Chih-Jen (Jacky) Hsu, Chi-An (Rocky) Wu, and Kuan-Hua Tu
Cadence Design Systems, Inc.

- 2024.03.04 First Version

1. Introduction

Logic synthesis/optimization is a key step in digital design flow. Traditionally, in this stage, the optimization metrics PPA (power, performance, area) might be majorly determined. However, as the technology node shrinks and the design process becomes extremely complicated, the logic optimization tools might be invoked into some iterative optimization flow or called by some local re-synthesis for variant purposes. In the meantime, the optimization metric would not be limited to PPA. It may also optimize for critical modules, placeability, routability, verification, testability, engineering change order, system redundancy, and more.

In this contest, we propose a reinforcement logic optimization problem. We will provide a cost function estimator (a black-box executable file). Contestants are tasked with developing a program that interacts with the estimator and learns to perform optimizations to minimize the cost. The expected optimization framework is shown in Figure 1. The inputs of the framework are the given netlist and cell library, and the output is the optimized netlist. In the framework, the developed algorithms and the cost function estimator continuously interact with each other. The algorithms perform optimizations based on the cost received from the estimator and output the optimized netlist to the estimator. The estimator receives the optimized netlist and outputs the cost to the algorithms again. The objective is for the algorithms to learn to minimize the cost.

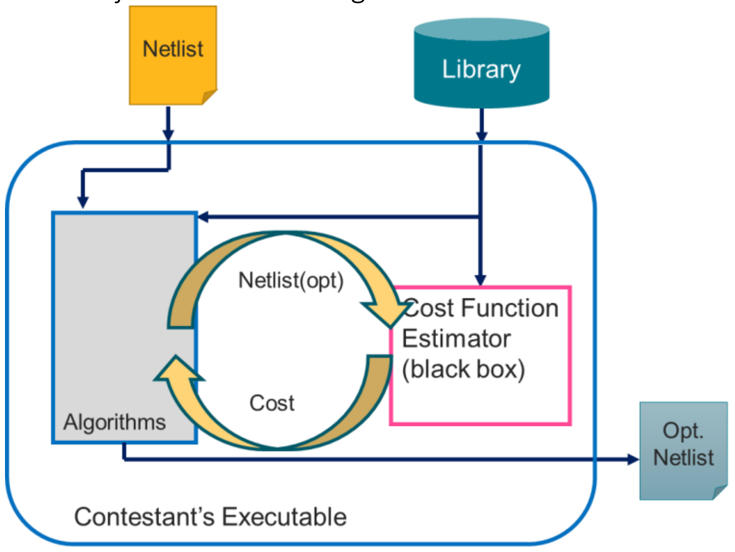

Figure 1. The expected optimization framework.

2. Background

2.1 Logic optimization in local re-synthesis

Local re-synthesis is widely used in modern design flow not only for PPA, but also for other reasons such as resolving congestion, supporting functional ECO, etc. In such complex scenarios, the objective is to correctly model the criticality and budget of resources (timing, power, P&R resource, …) as non-linear functions in local re-synthesis[1][2][3][4].

In addition, there is a set of complicated constraints different from one block to the other. Let us take ECO as an example. When the spare cells for each gate type are limited, the number of usable cells becomes a limitation or part of the constraint. Similarly, considering routing resources during local resynthesis is also important to avoid congestion that can end up with a large amount of DRC violations. Considering all the conditions, an optimization strategy for general cost function may greatly reduce human effort and the time for design closure.

2.2 Reinforcement learning and logic optimization

Traditional machine learning techniques, such as supervised learning, typically rely on large amounts of hand-labeled training data, which can be challenging to obtain, especially for optimization problems. In contrast, reinforcement learning (RL) enables an AI-driven system to learn through trial and error, receiving feedback from its actions. This makes RL well-suited for problems that require searching and exploring a vast solution space within a complex environment[5][6][7].

For the logic optimization problem with a general cost function, contestants are expected to develop an intelligent methodology to explore the solution space, rather than relying on a cost-function-based algorithm for optimization and legalization of the design.

3. Problem Formulation and Input Output Format

This section defines the input, output, and requirements of this contest.

The submitted executable file by the contestants should take three input files and output the optimized netlist. The first input is a flattened Verilog netlist to be optimized. The second input is a cell library which describes the spec. of each library cell. The third input is a cost function estimator (an executable file). Contestants need to write a program to generate a functionally equivalent netlist using the cells in the cell library and minimize the cost reported by the estimator.

3.1 Program requirement

The developed program must be run on a Linux system. The time limit for each case is 8 hours. Parallel computation with multiple threads or processes is not allowed. The program should accept four groups of arguments, "-netlist <netlist_path/name.v>", "-library <lib_path/name.lib>", "-cost_function <cost_function_path/name>" and "-output <output_path/name.v>", for specifying the input and output files, respectively. For example, the command line for executing the program can be "`./cada0000_alpha -`

`cost_function cost_function_1 -library low_vt.lib -netlist design.v -output design_optimized.v`”

3.2 Input file format

3.2.1 Netlist

- The input netlist is a flattened netlist in Verilog format without hierarchy (one top module only).
- The netlist is composed of:
    1. primitive gates (`and, or, nand, nor, not, buf, xor, xnor`)
    2. wires
    3. constant values (1’b1, 1’b0)

    Note that all primitive gates are assumed to have only 2 inputs and 1 output except for `buf` and `not` gates, which have only 1 input and 1 output.
- All primary inputs and primary outputs are scalars (i.e., one-bit signals)
    - input in1; // an input scalar
    - output out1; // an output scalar

3.2.2 Cell library

- For each primitive gate, at least one cell exists in the cell library.
- For each 2-input cell (all primitive gates except `buf` and `not`), the names of the input pins are A and B, and the name of the output pin is Y.
- For `not` and `buf`, the names of the input pin and the output pin are A and Y, respectively.
- The cell library is given in a json-liked format.
    - The first section is `information` with three fields. The first field is `cell_num`, which states the number of cells in the cell library. The second field is `attribute_num`, which states the number of attributes in each cell. The third field is `attributes`, in which there are `#attribute_num` strings denoting the field names of the attributes of each cell. Note that at least 2 attributes, `cell_name` and `cell_type`, are given.
    - The second section is `cells`. There are exactly `#cell_num` cells described in this section. For each cell, exactly `#attribute_num` fields are specified. The values of `cell_name` and `cell_type` are all a string. For the attributes whose names end with “_f”, their values are floating numbers. For those attributes whose names end with “_i”, their values are integers. Furthermore, the value of `cell_type` is one of the primitive gates.
- For the convenience of parsing the file, each bracket and comma is followed by a space, enter, or tab.
- Figure 2 shows an example.

```
 {
    "information" : {
          "cell_num" : "8" ,
          "attribute_num" : "6" ,
          "attributes" : [
                "cell_name" ,
                "cell_type" ,
                "delay_f" ,
                "power_f" ,
                "attribute_1_i" ,
                "attribute_2_f"
          ]
    } ,
    "cells" : [
          {
                "cell_name" : "NAND2X1" ,
                "cell_type" : "nand" ,
                "delay_f" : "45.23" ,
                "power_f" : "910.85" ,
                "attribute_1_i" : "123" ,
                "attribute_2_f" : "45.678"
          } ,
          ...
    ]
 }
```

Figure 2. Example of cell library file.

3.2.3 Cost function estimator

- The cost function estimator is an executable file, which takes 2 input files and generates 1 output file. The input files are a cell library and a netlist. All the gates in the netlist should be specified by the cell library. The output file contains a floating number which denotes the cost of the netlist.
- An example of executing the estimator is as follows: "`./cost_function_1 -library low_vt.lib -netlist design_iter0008.v -output cost_0008.txt`"

3.3 Output file format and requirements

- The output file should be in Verilog format.
- DO NOT change the name of the top module.
- DO NOT change the name and declaration for the primary inputs and the primary outputs.
- Only the cells specified in the cell library are allowed in the output netlist.

4. Evaluation Criteria

- Correctness is necessary. If the generated netlist is not functionally equivalent to the given netlist, the contestants will get zero points for the case.
- ***cost*** for each case: The cost reported by the cost function estimator on the finally outputted netlist.
- ***point*** for each case:

$$\boldsymbol{point} = \frac{\min\left(\boldsymbol{cost}\ of\ the\ case\ from\ all\ teams\right)}{\boldsymbol{cost}\ of\ the\ case} \times 100\%$$

- ***final_score*** is the sum of ***point*** of all cases.
- The team having the largest ***final_score*** wins.

5. Examples

TBD.

6. References

[1] Maestre, S., Gumera, A., Hora, J., & de Guzman, M. J. (2022, July). Improving Digital Design PPA (Performance, Power, Area) using iSpatial Physical Restructuring. In 2022 37th International Technical Conference on Circuits/Systems, Computers and Communications (ITC-CSCC) (pp. 647-651). IEEE.

[2] Pandini, D., Pileggi, L. T., & Strojwas, A. J. (2002, March). Congestion-aware logic synthesis. In Proceedings 2002 Design, Automation and Test in Europe Conference and Exhibition (pp. 664-671). IEEE.

[3] Ratkovic, I., Palomar, O., Stanic, M., Unsal, O., Cristal, A., & Valero, M. (2014, July). Physical vs. physically-aware estimation flow: case study of design space exploration of adders. In 2014 IEEE Computer Society Annual Symposium on VLSI (pp. 118-123). IEEE.

[4] Tatsuoka, M., Watanabe, R., Otsuka, T., Hasegawa, T., Zhu, Q., Okamura, R., ... & Takabatake, T. (2015, June). Physically aware high level synthesis design flow. In Proceedings of the 52nd Annual Design Automation Conference (pp. 1-6).

[5] What is reinforcement learning? https://online.york.ac.uk/what-is-reinforcement-learning/

[6] Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing atari with deep reinforcement learning. arXiv preprint arXiv:1312.5602.

[7] Shan-Hung Wu, "Reinforcement Learning," https://nthu-datalab.github.io/ml/slides/14_Reinforcement_Learning.pdf