Power and Timing Optimization Using Multibit Flip-Flop

Sheng-Wei Yang, Jhih-Wei Hsu, Yu-Hsuan Cheng, Cindy Chin-Fang Shen

Synopsys, Inc.

0. Revision History

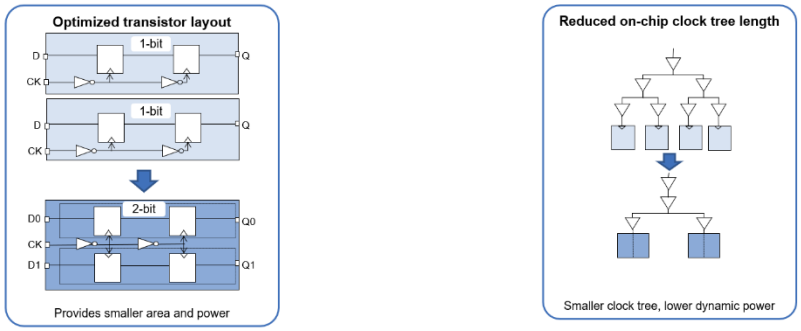
2025-07-08	-	Fixed missing attribute in appended program format rule
2025-06-24	-	Appended program format rule
2025-05-29	-	Revised submit format and file content
2025-05-08	-	Revised testcase access procedure
2025-04-18	-	Added SDC as file input
2025-02-28	-	Added all registers as file input
2025-02-21	-	Second revision
2025-02-14	-	First revision
2025-02-07	-	Initial version

1. Introduction

With advances in modern technology nodes, minimizing power and area has become one of the most crucial challenges in the semiconductor industry. A widely adopted approach to address this challenge is to optimize the number and types of flip-flops, which are fundamental components in modern digital designs.

Substituting multiple single-bit flip-flops with a single multibit flip-flop is referred to as "multibit flip-flop banking." Since a multibit flip-flop requires less space than the individual flip-flops it replaces, more area can be saved. Fig. 1(a) illustrates this concept. Furthermore, each single-bit flip-flop has separate power, ground, and clock pin connections. Replacing them with a single multibit flip-flop helps streamline and reduce power, ground, and clock net routing complexity. Figure 1(b) provides an example. However, optimizing timing, power, and area has become increasingly complex for modern designs. For certain timing-critical nets, the initial approach of multibit flip-flop banking may degrade the timing, hindering the overall optimization goal. Therefore, in some cases, it becomes necessary to split a multibit flip-flop into several single-bit flip-flops to optimize timing-critical nets better. This technique is commonly known as "multibit flip-flop debanking."

This contest is an extension of the 2024 ICCAD CAD Contest Problem B. The contest’s objective is to simulate multibit flip-flop banking and debanking decisions in virtual designs. Contestants must optimize timing, power, and area simultaneously while ensuring your program runs efficiently (i.e., with a short enough runtime to avoid penalties).



(a) (b)

Figure 1(a): A basic transistor layout featuring two single-bit flip-flops, along with one multi-bit flip-flop.

Figure 1(b): A diagram showing how the dynamic power consumption of a design is reduced by shortening the on-chip clock tree length, achieved by replacing four one-bit flip-flops with two two-bit flip-flops.

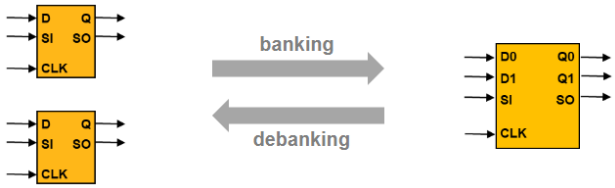


Figure 2: A simplified diagram of how two single-bit flip-flops can be banked into one two-bit flip-flop, and debanked vice versa. In this diagram, the upper left D pin of the single-bit flip-flop is mapped to the D0 pin of the two-bit flip-flop; the lower left D pin of the single-bit flip-flop is mapped to the D1 pin of the two-bit flip-flop.

2. Contest Objective

The goal is to develop a banking and debanking algorithm that produces a placement result optimized for timing, power, and area. No overlap of cells, whether combinational or sequential, is allowed in the placement result. The quality of the result is estimated by the cost function, which is the weighted summation of these three cost metrics: timing, power, and area, as follows:

$$\alpha \cdot TNS + \sum_{\forall i \in FF} \left(\beta \cdot Power(i) + \gamma \cdot Area(i)\right)$$

where i represents each flip-flop instance in the design, TNS denotes the total negative slack of all flip-flops, $Power(i)$ refers to the power consumption of the flip-flop, and $Area(i)$ represents the area cost of the flip-flop. α, β and γ indicate the weights for timing, power, and area, respectively. The values will be provided for each test case.

3. Problem Formulation and Input/Output Format

Given:

1. Weight factors for cost metrics
2. Timing slack and delay information
3. Power information
4. Area information
5. Cell placement result with flip-flop cells

Output:

1. A list of pin mapping between each input flip-flop pins and the output flip-flop pins
2. A flip-flop placement solution with its Verilog of netlist connections.

Note that cell overlaps are not allowed. Additionally, each flip-flop must be placed on the defined sites of the placement rows, while ensuring that the maximum placement utilization ratio is satisfied.

3.1 Format of Input Data

This section illustrates syntaxes for data input. Most of the syntaxes are floating point numbers. Contestants are expected to handle the data range within the limits of DBL_MAX.

There are two parts of files for each testcase:

- The first part is a file including weight factors for cost metrics, timing slack and delay information, power information, area information, and a list of all flip-flops. The syntaxes are described in Sections 3.1.1 to 3.1.4, and the list of all flip-flops is followed in 3.1.5.
- The second part is design files containing cell placement results with flip-flops cells in LEF and DEF formats described in Section 3.1.6.

3.1.1 Weight factors for cost metrics

α, β, and γ values are given out as Alpha, Beta, and Gamma, respectively.

Syntax

Alpha <alphaValue>

Beta <betaValue>

Gamma <gammaValue>

Example

Alpha 1

Beta 5

Gamma 5

3.1.2 Timing slack and delay information

The design's timing slack and delay information will be reported using Fusion Compiler's report_qor command. The keyword TNS stands for the total negative slack value calculated across the current design. The larger the value means the worse the total negative slack is for this design.

Syntax

TNS <Total Negative Slack>

Example

TNS 0.13

3.1.3 Power information

The power information of the design will be reported using Fusion Compiler's report_power command. The keyword. TPO stands for total power value calculated across the current design. The larger the value means the worse the total power consumption it requires to run this design.

Syntax

TPO <Total Power>

Example

TPO 2.76

3.1.4 Area information

The design's area information will be reported using Fusion Compiler's report_qor command. The keyword Area value refers to the cell area in the design. A larger value indicates a larger total cell area required to tape out the design.

Syntax

Area <Cell Area>

Example

Area 19.58

3.1.5 Flip-Flop Library Cells

A list of all register cell names for this design will be listed out following previous syntaxes.

3.1.6 Cell placement result with flip-flops cells

The cell placement results, including flip-flop cells, will be provided in LEF/DEF format. A Synopsys Design Constraint (SDC) file will be included to represent timing information. For the detail format of LEF DEF, please refer to LEF/DEF Language Reference[5]. For the detail format of Synopsys Design Constraints (SDC), please refer to Synopsys Timing Constraints Manager [6], or some other tutorial documents regarding the usage of SDC. [7] [8]

3.2 Format of Output Data

The output contains a cell placement result with flip-flop cells in the same format as the input data. Additionally, contestants need to provide a list detailing the changed cells and their corresponding pin mappings. Only the resultant cells and their mappings should be output, excluding intermittent cells or any combinational gates. The number of cell instances may vary based on the banking and debanking methods used to generate the output. All cell instances must be placed on-site, with the lower-left corner of each cell aligning with the lower-left corner of a placement site, ensuring that no cell overlaps with another.

For each test case, two parts of the output file are expected:

- The first part is a file listing the pin mappings between each input flip-flop's pins and the output flip-flop's pins using the syntaxes described in section 3.2.1.
- The second part is the design files containing the flip-flop placement solution described in section 3.2.2.

3.2.1 A list of pin mapping between each input flip-flop pins and the output flip-flop pins

This file contains all the information regarding the mapping from the original input to the output. The keyword CellInst refers to the total number of flip-flop cell instances in the output design. For each flip-flop cell instance in the input design with a valid data connection, we expect a 1-to-1 mapping of flip-flop pins between the input and output designs.

Syntax

CellInst <InstCount>

<originalCellPinFullName> map <resultCellPinFullNameName>

Example

CellInst 2

C1/D map C6/D

C1/Q map C6/Q

C1/CLK map C6/CLK

C2/D map C5/D1

C2/Q map C5/Q1

C2/CLK map C5/CLK

C3/D map C5/D0

C3/Q map C5/Q0

C3/CLK map C5/CLK

3.2.2 A flip-flop placement solution

A flip-flop placement solution is expected to be presented in the format of LEF DEF and Verilog. For the detail format of LEF DEF, please refer to LEF/DEF Language Reference[5].

4. Example

We take the following circuit as a sample input:

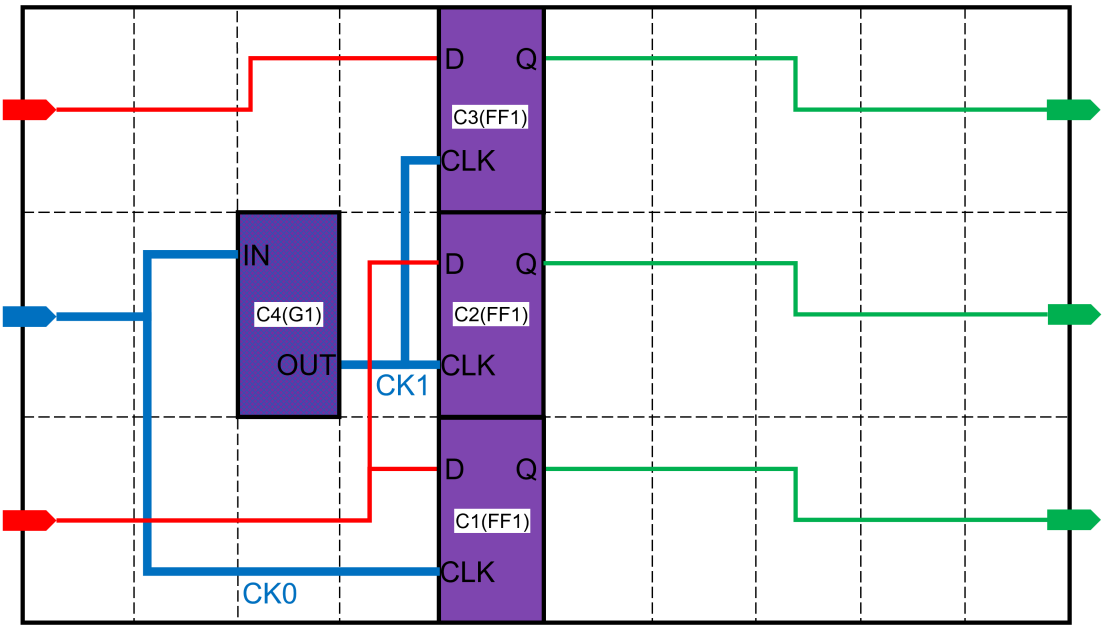


Figure 3: A visualization of the example data input circuit. CK0 and CK1 are two different clock nets.

Input File 1: Syntax Data:

Alpha 1

Beta 5

Gamma 5

TNS 0.13

TPO 2.76

Area 19.58

Input File 2:

Cell placement result with flip-flops cells (in LEF DEF)

A sample output:

Based on the given input, here's an example of output:

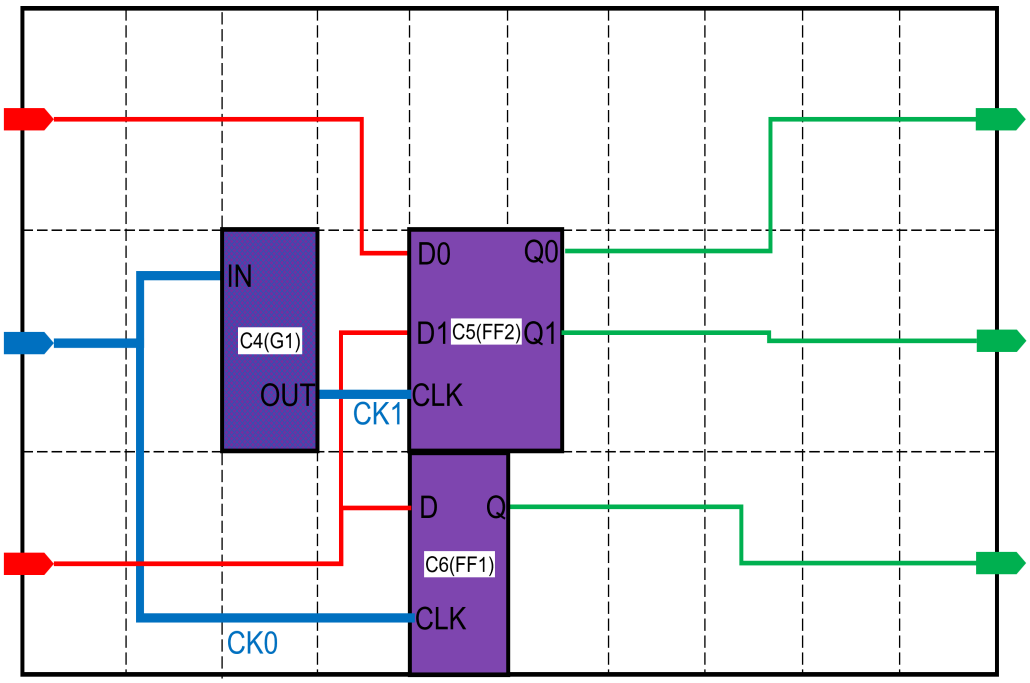


Figure 4: A visualization of the example data output circuit. CK0 and CK1 remain two different clock nets.

Output File 1: Syntax Data

CellInst 2

C1/D map C6/D

C1/Q map C6/Q

C1/CLK map C6/CLK

C2/D map C5/D1

C2/Q map C5/Q1

C2/CLK map C5/CLK

C3/D map C5/D0

C3/Q map C5/Q0

C3/CLK map C5/CLK

Output File 2:

A flip-flop placement solution (outputFileName.def and outputFileName.v)

5. Evaluation

The expected output must satisfy the following constraints. The satisfiability of these constraints would be evaluated via the latest release of Synopsys Fusion Compiler. Failure to meet any of these constraints will result in no score:

A. All instances must be placed on-site and within the die region.
B. No overlap is allowed.
C. Nets connected to the flip-flops must remain functionally equivalent to the data input after banking or debanking. Specifically, all D pin and Q pin connections must remain functionally equivalent, and all CLK pins must be connected to the same clock net. The result should not leave any open or short nets.
D. A list of banking and debanking mapping needs to be provided

- Banking Criteria:
  Contestants are allowed to perform banking by clustering several lower-bit flip-flops to one higher-bit flip-flop. However, all D pin and Q pin connections need to remain functionally equivalent after banking, and all CLK pins need to be connected to the same clock net. If there is any design for testability (DFT) scan chain connection on scan-input (SI) or scan-output (SO) pins of the lower-bit flip-flops, the banking process should honor the scan chain connection.
- Debanking Criteria:
  Contestants are allowed to perform debanking by split a higher-bit flip-flop to several lower-bit flip-flops. However, all D pin and Q pin connections need to remain functionally equivalent after debanking, and all CLK pins need to remain connected to the same clock net. If there is any design for testability (DFT) scan chain connection on scan-input (SI) or scan-output (SO) pins of the higher-bit flip-flops, the debanking process should honor the scan chain connection.

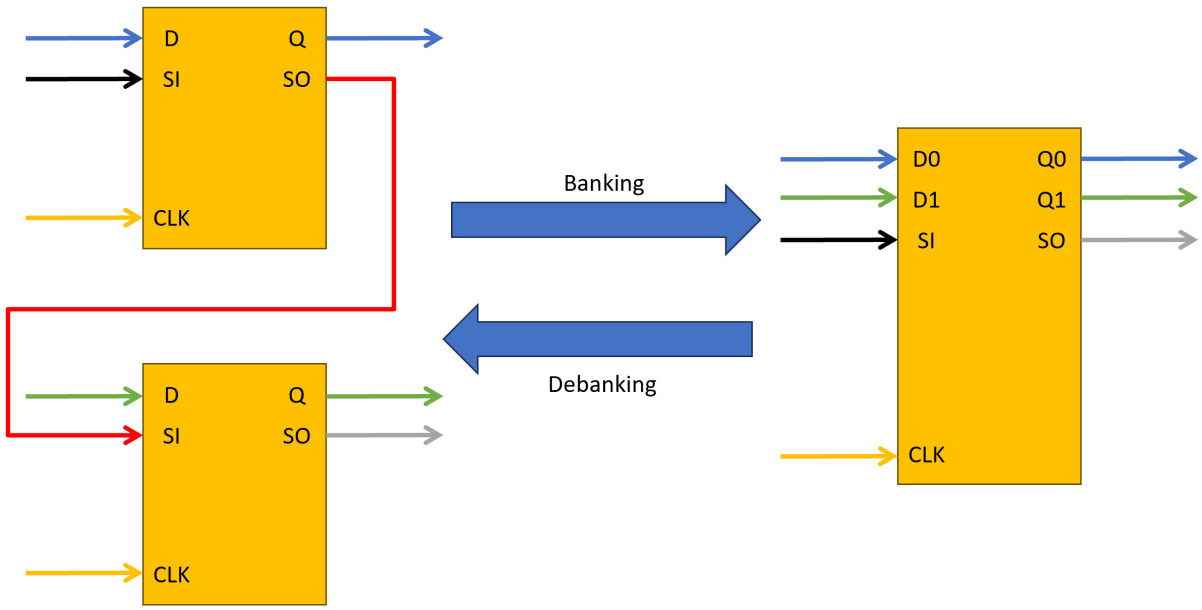


Figure 5: A banking and debanking example involving design for testability (DFT) pins.

Figure 5 provides an example to illustrate the criteria. A scan chain is formed starting from the Scan-Input (SI) pin of the upper-left flip-flop. The Scan-Output (SO) pin of the upper-left flip-flop is then connected to the SI pin of the bottom-left flip-flop, and subsequently to the SO pin of the bottom-left flip-flop. Therefore, when banking the two single-bit flip-flops, the banked multibit flip-flop should map the SI pin of the upper-left flip-flop to the SI pin of the multibit flip-flop, and the SO pin of the bottom-left flip-flop to the SO pin of the multibit flip-flop. When debanking from the multibit flip-flop, a scan chain is formed from the SI to SO pins of the multibit flip-flop. Therefore, when debanking the multibit flip-flop into single-bit flip-flops, the SO pin of the debanked single-bit flip-flop in the upper-left corner should be connected to the SI pin of the debanked single-bit flip-flop in the bottom-left corner.

5.1 Overview of Score Function

The quality of your solution is evaluated by considering both the optimization of timing, power, and area, as well as the efficiency of your code. The formula for calculating the final score is:

$$\boldsymbol{Final\ score} = Initial\ score\ \times (1.0 + runtime\ factor\ bounded)$$

"$Initial\ score$" reflects the optimization of timing, power, and area. It is calculated as:

$$Initial\ score = \alpha \cdot TNS + \sum_{\forall i \in FF} \left(\beta \cdot Power(i) + \gamma \cdot Area(i)\right)$$

TNS indicates how well the timing of your design is optimized. It is reported using the command "***report_qor***" from Fusion Compiler

$Power$ stands for total power consumption of your design. It is reported using the command "***report_power***" from Fusion Compiler.

$Area$ stands for total area of cells. It is reported using the command "***report_qor***" from Fusion Compiler.

"$runtime\ factor\ bounded$" reveals the efficiency of your code. To prevent large changes in the final score, the "$runtime\ factor$" is bounded within a ±10% range as follows:

$$runtime\ factor\ bounded = max\big(-0.1, min(0.1, runtime\ factor)\big)$$

"$runtime\ factor$" adjusts the score based on the performance of your code (i.e., how fast or slow it runs compared to an expected reference time). It is estimated as follows:

$$runtime\ factor = \ 0.02 \times \log_2(\frac{elapse\ time\ of\ test\ binary}{median\ elapse\ time})$$

where

- elapsed time of test binary is the runtime took by your code
- median elapsed time is the reference time based on the median execution time of all contest submissions.

Specifically, if the binary is 2x faster or slower than the expected runtime, the score will increase or decrease by 2% of the initial score. If the binary runs 4x faster or slower, the score will adjust by ±4%.

Key Considerations:

- CPU Cores: You have 8 CPU cores available for your program.
- Time Limit: The evaluation machine is limited to 60 minutes for each test case. The hidden cases will be in the same scale as public cases.

5.2 Program Requirements

The contestant's submitted program should be able to execute as follows

```
./cadb_0000_alpha
    -weight <weightFile>
    -lib <libFile1> <libFile2> ...
    -lef <lefFile1> <lefFile2> ...
    -db <dbFile1> <dbFile2> ...
    -tf <tfFile1> <tfFile2> ...
    -sdc <sdcFile1> <sdcFile2> ...
    -v <verilogFile1> <verilogFile2> ...
    -def <defFile1> <defFile2> ...
    -out <outputName>
```

Notes:

1. Separate multiple files for each parameter with a space.

2. The order of parameters is flexible, but each parameter must be followed by its corresponding files or values.

This program format rule is recommended for alpha testing and will be enforced for beta and final evaluation. Contestants submitting alpha testing, if failed to abide by the requested program format rule, are still eligible for evaluation as long as a **README** file with detailed description on how to run the submission is provided along with the submission. Failure to follow the program format will not be eligible for scoring in beta and final submission.

outputName represents the expected output file name and will be the last parameter when executing the contestant's program. Contestants are expected to output their results as follows:

`outputName.list outputName.v outputName.def`

`outputName.list` represents the output syntax data, including that covered in section 3.2.1.

outputName.def and outputName.v represent the flip-flop placement solution and its netlist connection covered in section 3.2.2. Please follow the naming convention by prefixing your binary as cadb_0000*, where 0000 is the placeholder of contestants serial number. Please replace 0000 with your registered team number.

Please follow the naming convention by prefixing your binary as cadb_0000*, where 0000 is the placeholder of contestants serial number. Please replace 0000 with your registered team number.

The *_alpha stands for the alpha testing stage. For each stage please specify with different postfixes (*_beta, *_final).

Please note that contestants should follow the naming description; otherwise, the score will be annulled in beta and final submission.

Contestants will have access to the machine provided by ICCAD. They must upload their binary along with the necessary modules to run the binary. Contestants should note that the score will be marked as a failure if the public account cannot run its binary for any reason.

6. Access to Testcase

The testcases are generated based on Synopsys technology. Therefore, contestants are required to register at Synopsys SolvNetPlus website to receive the testcases:

Please sign-up to the contest first. Access to the testcase will be distributed after 5/18 and will take up to 7 - 10 work days.

7. References

[1] 2024 ICCAD CAD Contest Problem B: https://www.iccad-contest.org/2024/Problems.html

[2] Ya-Chu Chang, Tung-Wei Lin, Iris Hui-Ru Jiang, and Gi-Joon Nam. 2019. Graceful Register Clustering by Effective Mean Shift Algorithm for Power and Timing Balancing. In Proceedings of the 2019 International Symposium on Physical Design (ISPD '19).

[3] Meng-Yun Liu, Yu-Cheng Lai, Wai-Kei Mak, and Ting-Chi Wang. 2022. Generation of Mixed-Driving Multi-Bit Flip-Flops for Power Optimization. In Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design (ICCAD '22).

[4] Gang Wu, Yue Xu, Dean Wu, Manoj Ragupathy, Yu-yen Mo, and Chris Chu. 2016. Flip-flop clustering by weighted K-means algorithm. In Proceedings of the 53rd Annual Design Automation Conference (DAC '16)

[5] LEF/DEF Language Reference https://www.ispd.cc/contests/18/lefdefref.pdf

[6] Synopsys Timing Constraints Manager https://www.synopsys.com/verification/static-and-formal-verification/timing-constraints-manager.html

[7] Synopsys® Design Constraints File (.sdc) Definition https://www.intel.com/content/www/us/en/programmable/quartushelp/current/index.htm#reference/glossary/def_sdc.htm

[8] Synopsys Design Constraints (SDC) Basics https://www.vlsi-expert.com/2011/02/synopsys-design-constraints-sdc-basics.html

[9] Synopsys' Open-Source Liberty Format to Incorporate On-Chip Variation Extensions https://news.synopsys.com/home?item=123415