Generation of Mixed-Driving Multi-Bit Flip-Flops for Power Optimization

Meng-Yun Liu, Yu-Cheng Lai, Wai-Kei Mak, and Ting-Chi Wang
Department of Computer Science, National Tsing Hua University
Hsinchu, Taiwan
newmileou@gapp.nthu.edu.tw,s109062701@m109.nthu.edu.tw,wkmak@cs.nthu.edu.tw,tcwang@cs.nthu.edu.tw

ABSTRACT

Multi-bit flip-flops (MBFFs) are often used to reduce the number of clock sinks, resulting in a low-power design. A traditional MBFF is composed of individual FFs of uniform driving strength. However, if some but not all of the bits of an MBFF violate timing constraints, the MBFF has to be sized up or decomposed into smaller bit-width combinations to satisfy timing, which reduces the power saving. In this paper, we present a new MBFF generation approach considering mixed-driving MBFFs whose certain bits have a higher driving strength than the other bits. To maximize the FF merging rate (and hence to minimize the final amount of clock sinks), our approach will first perform aggressive FF merging subject to timing constraints. Our merging is aggressive in the sense that we are willing to possibly oversize some FFs and allow the presence of empty bits in an MBFF to merge FFs into MBFFs of uniform driving strengths as much as possible. The oversized individual FFs of an MBFF will be later downsized subject to timing constraints by our approach, which results in a mixed-driving MBFF. Our MBFF generation approach has been combined with a commercial place and route tool, and our experimental results show the superiority of our approach over a prior work that considers uniform-driving MBFFs only in terms of the clock sink count, the FF power, the clock buffer count, and the routed clock wirelength.

ACM Reference Format:
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 *IEEE/ACM International Conference on Computer-Aided Design (ICCAD '22), October 30-November 3, 2022, San Diego, CA, USA.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3508352.3549473

1 INTRODUCTION

Due to the increasing demand for longer battery life in portable electronics, reducing power consumption has become one of the most important goals in circuit design, and a clock network accounts for a major portion of power consumption in a chip [11]. Since using multi-bit flip-flops (MBFFs) can reduce the clock network usage, it is often used in low-power circuit design. There are two reasons why MBFFs have better power consumption than traditional single-bit flip-flops (FF). One reason is that the power per bit for an MBFF is smaller than a single-bit FF [3], and the other is that using MBFFs reduces the number of clock sinks, which greatly reduces the wirelength of the clock tree and the clock buffer count, resulting in less clock power consumption.

1.1 Previous Works

In a typical design flow, MBFFs can be generated by merging or clustering multiple single-bit FFs in the logic synthesis, placement, or post-placement stage. Hou *et al.* [6] and Yi *et al.* [22] developed methods for MBFF generation during logic synthesis.

Some studies focus on MBFF generation in the placement stage. Tsai *et al.* [19] developed a FF bonding method inspired by ionic bonding in Chemistry. Lin *et al.* [14] proposed an anchor method to generate MBFFs while considering the impact on clock trees.

There are research efforts that generate MBFFs in the post-placement stage. Wang *et al.* [20], Jiang *et al.* [9], Shyu *et al.* [18], and Liu *et al.* [15] utilized the timing slacks of FF's pins to develop methods for finding timing-feasible regions and generating MBFFs. Wu *et al.* [21] developed a weighted K-means algorithm. Seitanidis *et al.* [16] introduced an Integer Linear Programming (ILP) algorithm, which verifies the timing compatibility and identifies the mergeable FFs; they also provided a method by decomposing an MBFF into multiple single-bit FFs to handle circuits with existing MBFFs and proposed an approach to further reduce the clock sink count by leaving empty bits in MBFFs. Kahng *et al.* [10] proposed an algorithm that considers the aspect ratio of an MBFF and explores the possibility of using multi-row 64-bit MBFFs. Chang *et al.* [2] introduced a method that clusters multiple single-bit FFs by moving them to an appropriate location, which would improve the results of clock tree synthesis (CTS) and worst negative slack (WNS).

1.2 Motivation

Although different MBFF generation strategies have been explored in the previous works, all of them focus on uniform-driving MBFFs (i.e., individual FFs in an MBFF all have the same driving strength). Even if only one of the bits of an MBFF violates timing constraints, every bit in the MBFF has to be sized up to a higher driving strength or decomposed into smaller bit-width MBFF combinations to satisfy timing, which reduces the power saving. Motivated by [5], a better strategy is to use mixed-driving MBFFs whose certain bits have a higher driving strength than the others to fix timing. Such a mixed-driving MBFF also has a smaller area and power than an uniform-

This work was supported in part by the National Science and Technology Council under grants 110-2221-E-007-122 and 111-2221-E-007-119-MY3.

ICCAD '22, October 30-November 3, 2022, San Diego, CA, USA

ACM ISBN 978-1-4503-9217-4/22/10…$15.00
https://doi.org/10.1145/3508352.3549473

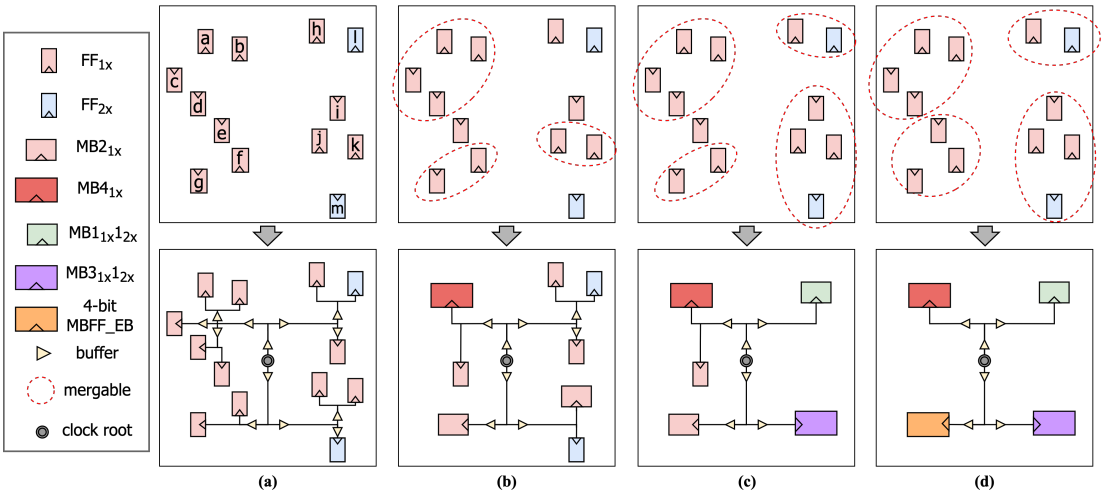

Figure 1: (a) The original clock tree. (b) The clock tree after traditional MBFF generation. (c) The clock tree after mixed-driving MBFF generation . (d) The clock tree after mixed-driving MBFF generation that leaves an empty bit in an 4-bit MBFF.

and high-driving counterpart. For example, Fig. 2(a) shows a 1x-driving 4-bit MBFF $MB4_{1x}$ which is composed of four individual FFs each with 1x driving strength, and if it is in a design and has two bits violating timing, all its bits will be sized up (e.g., to have a 2x driving strength) to fix timing by a traditional approach, forming a 2x-driving MBFF $MB4_{2x}$ as shown in Fig. 2(b). Although $MB4_{2x}$ has a higher driving capability to satisfy timing, it will unnecessarily increase both the area and power; hence it is an *oversized* MBFF. In fact, two bits of $MB4_{2x}$ could be downsized to 1x driving strength while still meeting timing, resulting in a mixed-driving MBFF $MB2_{1x}2_{2x}$, which is shown in Fig. 2(c) and has smaller area and power than $MB4_{2x}$.

Fig. 1 illustrates an example of clock trees constructed under different considerations for MBFF generation. The original clock tree in Fig. 1(a) contains eleven 1x-driving 1-bit FFs (a, b,..., k) and two 2x-driving 1-bit FFs (l, m), with thirteen clock buffers. In Fig. 1(b), the number of buffers is reduced to eight after performing traditional MBFF generation. Since traditional MBFF generation only utilizes uniform-driving MBFFs, the two FFs with different driving strengths in the upper-right corner of Fig. 1(b), i.e., h and l,

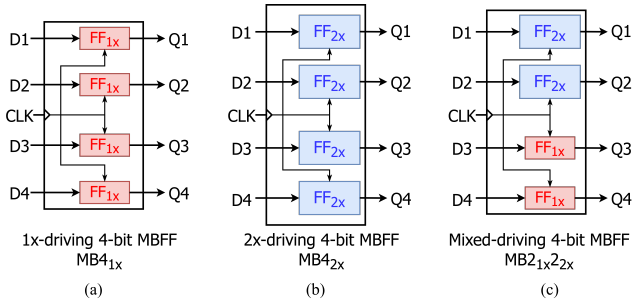

Figure 2: (a) Traditional 1x-driving MBFF. (b) Traditional 2x-driving MBFF. (c) Mixed-driving MBFF.

cannot be merged. Nevertheless, with the consideration of mixed-driving MBFFs, h and l can be merged to generate a 2-bit mixed-driving MBFF, as shown in Fig. 1(c), and the resulting clock tree has less sink and buffer count than the one in Fig. 1(b).

As observed in [16], it is not necessary to generate a complete MBFF where all bits are filled. For example, in Fig. 1(d), there are three FFs in the lower-left corner (i.e., e, f, and g), and they have the chance to be merged. If an MBFF with empty bits is not an option, only f and g are merged into a 2-bit MBFF, as shown in Fig. 1(c). But if the option is available, the three FFs can be merged into a 4-bit MBFF leaving one empty bit, as shown in Fig. 1(d), and both the sink count and clock wirelength can be further reduced.

1.3 Our Contributions

Motivated by the example given in Fig. 1, in this paper, a mixed-driving MBFF generation approach is proposed and it can be applied in the post-placement stage for power optimization. The contributions of this paper are as follows:

- A mixed-driving MBFF generation approach is presented, and it can be extended to generate MBFFs with empty bits. To our best knowledge, this is the first work that considers mixed-driving MBFFs.
- Our approach is aware of timing and power by using a cost function to select appropriate MBFF candidates.
- Our approach is tightly integrated with a commercial place and route tool which is used to do the placement legalization, timing analysis, CTS, and routing.
- According to our experiment results on ICCAD 2015 contest benchmarks [12], our approach outperforms a prior work [2] that considers uniform-driving MBFFs only in terms of the clock sink count, the FF area, the FF power, the clock buffer count, and the routed clock wirelength.

The remainder of this paper is organized as follows. Section 2 defines the mixed-driving MBFF generation problem. Details of our

approach are introduced in Section 3. Experiment results are shown in Section 4, and Section 5 concludes this paper.

2 PROBLEM FORMULATION

An MBFF library containing uniform-driving and mixed-driving MBFFs, where the driving strength is from 1x to nx and the amount of bits is from 2 to k, is adopted.

The **Mixed-Driving MBFF Generation Problem** is defined as follows:

- Input: A netlist and an associated legal placement.[1] If there are uniform-driving MBFFs in the design, they will be pre-processed by decomposing them into multiple single-bit FFs while keeping the driving strengths intact.
- Output: A refined netlist containing mixed-driving MBFFs, and a refined placement without timing violations.

The objective is to minimize the total power consumption and minimize the amount of oversized MBFFs for power optimization, resulting in a low-power design.

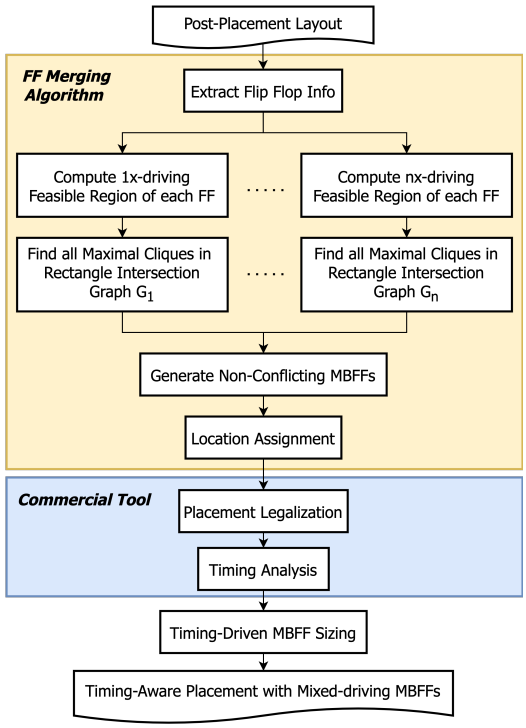

Figure 3: The overall approach.

3 PROPOSED APPROACH

To obtain a mixed-driving MBFF design for maximum power reduction under timing constraints, the key idea is to allow upsizing of the original individual FFs to increase their merging choices when forming the MBFFs, then at the end downsizing some bits in the MBFFs formed after placement legalization while preserving timing. The proposed approach is shown in Fig. 3. First, for each original FF in the design, its fan-in/fan-out pin locations and their timing slacks under different driving strengths for the FF are obtained. Second, we compute the timing feasible regions of each FF under different driving strengths. Third, maximal cliques in each rectangle intersection graph corresponding to maximal groups of FFs that can be merged without timing violations are found to facilitate the generation of potential MBFFs. Next, a set of MBFFs are formed from the original FFs in the design to maximize power reduction. Then, these MBFFs formed are assigned to preferred locations and the design layout is saved. After placement legalization and timing analysis are done by a commercial tool, bits in the MBFFs with extra timing slacks will be downsized to further reduce the power. More details are introduced in the following sub-sections.

[1]The legal placement has no timing violation.

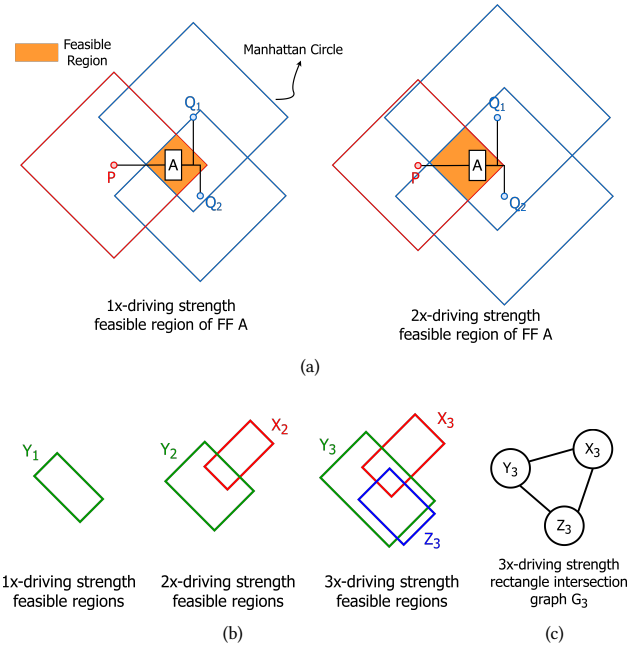

Figure 4: (a) The 1x and 2x-driving feasible regions of FF A are connected to a fan-in P and two fan-outs Q1 and Q2. (b) Different driving strength feasible regions are stored separately. (c) The rectangle intersection graph G_3 for the 3x-driving feasible regions in Fig. 4(b).

3.1 Compute Feasible Regions of Flip-Flop

To relocate a FF without timing violation, multiple feasible regions for each FF are computed in this stage assuming that a FF can be upsized. A timing feasible region of a FF is the intersection area of several Manhattan circles, where each Manhattan circle is centered at the fan-in/fan-out pin of the FF with its radius determined by the timing slack between the corresponding fan-in/fan-out pin and the FF. The timing slack is converted to an estimated wirelength bound by table lookup [13]. Since the feasible region is computed with timing awareness, the corresponding FF can be placed inside it without timing violation. We compute multiple feasible regions for each single-bit FF assuming different driving strengths that are at least as large as their originally given driving strength. For example, the feasible regions of FF A with 1x-driving strength and 2x-driving strength are shown in Fig. 4(a). The reason we consider the option

of sizing up a FF is that the feasible region of a FF is usually larger when the FF is upsized.

Feasible regions of different driving strengths are stored separately, as shown in Fig. 4(b). Suppose the cell library provides 1x, 2x, and 3x-driving FFs. Furthermore, suppose in the original netlist, FFs X, Y, and Z are 2x-driving, 1x-driving, and 3x-driving, respectively. Then, the 2x and 3x-driving feasible regions for X are computed. The 1x, 2x, and 3x-driving feasible regions for Y are computed. Only the 3x-driving feasible region for Z is computed.

The intersections of the ix-driving feasible regions of the original FFs can be described by a rectangle intersection graph G_i where each vertex corresponds to a ix-driving feasible region of an original FF and there is an edge between two vertices if their corresponding ix-driving feasible regions intersect. For example, Fig. 4(c) shows the rectangle intersection graph G_3 for 3x-driving feasible regions X_3, Y_3, and Z_3. We note that an intersection of the ix-driving feasible regions of multiple original FFs corresponds to a clique in the rectangle intersection graph G_i. Since any set of original FFs that can form an ix-driving MBFF without timing constraint violation must belong to some maximal cliques in G_i, it is of interest to find all maximal cliques of sizes ≥ 2 in each rectangle intersection graph.

3.2 Find all Maximal Cliques in Multiple Rectangle Intersection Graphs

In this stage, maximal cliques with sizes ≥ 2 in all of the rectangle intersection graphs are found. It is known that the maximal cliques can be computed efficiently without constructing the rectangle intersection graph [7]. Considering the ix-driving feasible regions computed in the previous stage, a sweep line method [17] is applied after the entire chip is rotated by 45° as in Fig. 5. The plane is divided into a number of vertical strips by extending all vertical boundaries of the ix-driving feasible regions. We scan each strip from top to bottom to identify the top boundaries of the rectangles called in-edges and the bottom boundaries of the rectangles called out-edges. A clique (i.e., an intersection of multiple rectangles) is maximal within a strip if its size is greater than the clique directly above it and the clique directly below it in the same strip. In Fig. 5, cliques $\{B_1, C_1\}$ and $\{A_1, B_1\}$ are maximal within the middle strip of 1x-driving feasible regions, and clique $\{A_2, B_2, C_2\}$ is maximal within the second strip from the right of 2x-driving feasible regions.

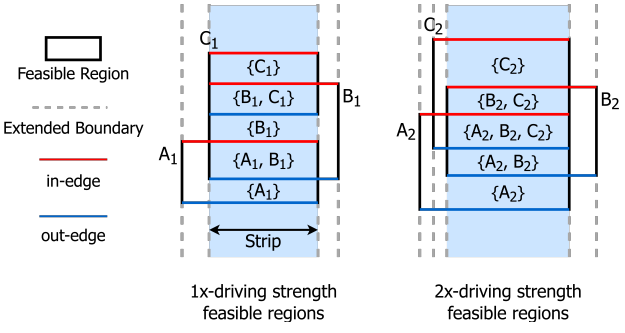

Figure 5: Sweep line method is used to find maximal intersections of feasible regions of same driving strength.

Note that the cliques that are maximal in each strip still need to be compared with those found in previous strips, and any of these cliques which is a proper subset of another one will be removed. Then, we can get all the maximal cliques in G_i.

3.3 Non-Conflicting MBFFs Generation

Any set of original FFs that can form an ix-driving MBFF without timing constraint violation must belong to some maximal cliques of rectangle intersection graph G_i. But enumerating all possible MBFF candidates for every maximal clique to find the best ones to form is extremely time-consuming. So, we implemented a best sampling procedure called **MBFF_Candidate(c)**. For each maximal clique c, we randomly generate min $(\binom{|c|}{k}, Sample_{bound})$ MBFF candidates for each kind of k-bit ($k \geq 2$) MBFFs in the library[2], and then selects the one with the minimum cost[3] to be the MBFF candidate for c.

Forming MBFFs with empty bits is considered in this paper to further increase the flip-flop merging rate and enhance the overall power reduction. We can allow MBFFs with empty bits to be generated by modifying our sampling procedure; the value of k needs not be limited to the available bit-sizes in the MBFF library, but it can be any integer less than or equal to min $(|c|, max_{bit})$ where max_{bit} is the maximum bit-size in the MBFF library. As a result, our algorithm can consider merging 3 FFs into a 4-bit MBFF, and 5/6/7 FFs into an 8-bit MBFF.

To form a set of non-conflicting MBFFs[4] for maximum power reduction, we define a cost function for selecting the appropriate MBFFs to be formed. Our cost of forming a new MBFF μ depends on the average merging flexibility of the FFs in μ, the number of used bits in μ, the reduction in FF power for μ, and the estimated increase in the signal switching power for the signal nets connected to μ. The merging flexibility δ for a FF is defined as the total number of maximal cliques in every intersection graph G_i that contains the FF. For the example in Fig. 6, $\delta(e)$ is 3 since FF e is contained in three maximal cliques. The smaller the merging flexibility of a FF means the lesser choices for it to merge with others, so the earlier our algorithm will attempt to merge it. Below is the formula for the cost of forming a new MBFF.

$$cost(\mu) = \lambda_1 \times \frac{D(\mu)}{B(\mu)} + \lambda_2 \times \frac{Power(\mu)}{OriPower(\mu)} + \lambda_3 \times \frac{\sum_{net_i \in Net(\mu)} (\alpha_i \times EstWL_i)}{\sum_{net_i \in Net(\mu)} (\alpha_i \times OriWL_i)} \quad (1)$$

where $D(\mu)$ is the average merging flexibility of the FFs in μ[5], $B(\mu)$ is the number of used bits in μ, $Power(\mu)$ is the FF power for μ, $OriPower(\mu)$ is the sum of FF powers of the original FFs that form μ, and α_i is the switching rate of signal i. $OriWL_i$ is the original half-perimeter wirelength (HPWL) of signal i. $EstWL_i$ is the estimated HPWL of signal i incident with μ calculated using the estimated location for μ, which is computed by the weighted median values

[2] $Sample_{bound}$ is set to 50 in the experiments.
[3] The cost function is defined in Equation (1).
[4] In other words, each FF in the original design can become a member of at most one formed MBFF.
[5] $D(\mu)$ is the average merging flexibility over the used bits in μ. For example, for a potential 4-bit MBFF $\{e_2, f_2, g_2, -\}$ with an empty bit in Fig. 6, its average merging flexibility is $(\delta(e) + \delta(f) + \delta(g))/3$.

Algorithm 1 Non-Conflicting MBFFs Generation

Input: The set of all maximal cliques C.
Output: A set of non-conflicting MBFFs N_M.

```
1:  Initialize an empty priority queue Q of MBFF candidates
2:  N_M ← ∅ ;
3:  for each c in C do
4:      Push MBFF_Candidate(c) into Q.
5:  repeat
6:      μ ← Q.top( )
7:      flag ← true
8:      for each FF f in MBFF μ do
9:          if marked(f) = true then
10:             flag ← false    // μ conflict with N_M
11:             break;
12:     if flag != false then
13:         N_M ← μ ∪ N_M
14:         for each FF f in μ do
15:             marked(f) ← true
16:     Update the clique c(μ) which generated μ
17:     Push a new MBFF_Candidate(c(μ)) into Q if |c(μ)| ≥ 2
18: until Q is empty
19: return N_M
```

of the x-(y-)coordinates of all cells connected to μ. λ_1, λ_2, and λ_3 are user defined parameters.

The reason for considering the value of $\frac{D(\mu)}{B(\mu)}$ in determining the priority for forming a new MBFF μ is as follows. It encourages a larger number of FFs to merge into a single MBFF with higher priority, especially those FFs with less merging flexibility, so the expected final reduction in the number of clock sinks when the algorithm terminates will be larger. Next, the smaller is the value of $\frac{Power(\mu)}{OriPower(\mu)}$, the higher is the FF power reduction if μ is formed[6]. Lastly, we have to balance with the increase in signal switching power if MBFF μ is formed, so we consider the value of $\frac{\sum_{net_i \in Net(\mu)}(\alpha_i \times EstWL_i)}{\sum_{net_i \in Net(\mu)}(\alpha_i \times OriWL_i)}$.

Algorithm 1 shows the pseudo-code for forming a set of non-conflicting MBFFs with various driving strengths for power reduction. In lines 3-4, after all of the MBFF candidates for each maximal clique are generated, they are pushed into a priority queue Q according to their costs. Each time an MBFF candidate μ with the minimum cost in the queue is selected in line 6 and the FFs in μ are checked if they conflict with the MBFFs already generated in set N_M by the for-loop of lines 8-11. If not, in lines 12-15, μ is put into N_M, and every FF in μ is marked. In line 16, the maximal clique $c(\mu)$ that contains μ will be reduced in size by removing all marked FFs. If the reduced clique's size is still larger than or equal to 2, a new MBFF candidate for it will be generated by sampling the reduced clique again and the new MBFF candidate is pushed into Q in line 17. The selection process in lines 5-18 will iterate until Q is empty.

Here is an example that illustrates the whole process. Assume that an MBFF library with 1/2/4-bits MBFFs for both 1x and 2x-driving is given, and we run our algorithm on the design in Fig. 1(a).

[6]Considering $\frac{Power(\mu)}{OriPower(\mu)}$ also prevents the algorithm from selecting higher driving MBFFs unnecessarily as the FF power increases when the driving strength goes up. If two MBFF candidates with different driving strengths cover the same set of original FFs, the smaller driving one will have a smaller cost and be chosen over the other one.

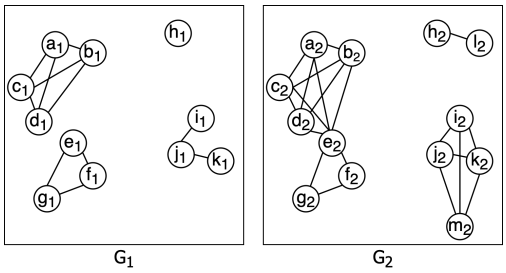

Figure 6: The 1x and 2x-driving rectangle intersection graphs G_1 and G_2 for the design of Fig. 2(a).

The corresponding intersection graphs G_1 and G_2 are shown in Fig. 6. By performing the sweep line method, all maximal cliques with size ≥ 2 are found:

- G_1 has four maximal cliques: $U_{1,1}$ = {a_1, b_1, c_1, d_1}, $U_{1,2}$ = {e_1, f_1, g_1}, $U_{1,3}$ = {i_1, j_1}, $U_{1,4}$ = {j_1, k_1}
- G_2 has four maximal cliques: $U_{2,1}$ = {a_2, b_2, c_2, d_2, e_2}, $U_{2,2}$ = {e_2, f_2, g_2}, $U_{2,3}$ = {h_2, l_2}, $U_{2,4}$ = {i_2, j_2, k_2, m_2}

After initial sampling, an MBFF candidate $s_{i,j}$ is found for each maximal clique $U_{i,j}$.

- $s_{1,1}$ = {a_1, b_1, c_1, d_1}, $s_{1,2}$ = {e_1, f_1, g_1, - }, $s_{1,3}$ = {i_1, j_1}, $s_{1,4}$ = {j_1, k_1}
- $s_{2,1}$ = {a_2, b_2, c_2, d_2}, $s_{2,2}$ = {e_2, f_2, g_2, - }, $s_{2,3}$ = {h_2,l_2}, $s_{2,4}$ = {i_2, j_2, k_2, m_2}
- Cost: $s_{1,1} < s_{2,4} < s_{1,2} < s_{2,1} < s_{2,3} < s_{2,2} < s_{1,3} \leq s_{1,4}$

Here is the process that generates N_M:

(1) In the first iteration, the minimum cost MBFF candidate $s_{1,1}$ is selected and put into N_M. Then, FFs a, b, c, d are marked. The maximal clique $U_{1,1}$ is updated to { }. Q becomes $s_{2,4} < s_{1,2} < s_{2,1} < s_{2,3} < s_{2,2} < s_{1,3} \leq s_{1,4}$.
(2) In the second iteration, $s_{2,4}$ is selected and put into N_M without conflict. Then, FFs i, j, k, m are marked. The maximal clique $U_{2,4}$ is updated to { }. Q becomes $s_{1,2} < s_{2,1} < s_{2,3} < s_{2,2} < s_{1,3} \leq s_{1,4}$.
(3) In the third iteration, $s_{1,2}$ is selected and put into N_M without conflict. Then, FFs e, f, g are marked. The maximal clique $U_{1,2}$ is updated to { }. Q becomes $s_{2,1} < s_{2,3} < s_{2,2} < s_{1,3} \leq s_{1,4}$.
(4) In the fourth iteration, $s_{2,1}$ is selected but since FFs a, b, c, d are marked, $s_{2,1}$ is discarded. Then, the maximal clique $U_{2,1}$ is updated to {e_2} but no new MBFF candidate for $U_{2,1}$ needs to be generated since its size is now one. Q becomes $s_{2,3} < s_{2,2} < s_{1,3} \leq s_{1,4}$.
(5) In the fifth iteration, $s_{2,3}$ is selected and put into N_M without conflict. Then, FFs h, l are marked. The maximal clique $U_{2,3}$ is updated to { }. Q becomes $s_{2,2} < s_{1,3} \leq s_{1,4}$.
(6) In the following iterations, the rest of the MBFF candidates in Q all contain some marked FFs thus conflict with those MBFFs in N_M. When Q becomes empty, the algorithm terminates.

The final MBFFs formed in the non-conflicting set N_M are $s_{1,1}$ = {a_1, b_1, c_1, d_1}, $s_{1,2}$ = {e_1, f_1, g_1, - } with an empty bit, $s_{2,3}$ = {h_2,l_2}, and $s_{2,4}$ = {i_2, j_2, k_2, m_2}. The final result is shown in Fig. 1(d). One observation can be made is that if merging FFs with empty bits is

not an option, the maximal clique $U_{1,2}$ will have to be decomposed to a 1-bit FF and a 2-bit MBFF, resulting in two sinks instead of one.

3.4 Location Assignment

To minimize the wirelength of the signal nets incident with the MBFFs, each MBFF in the non-conflicting set is assigned to an initial location within its preferred region. The preferred region of an MBFF is computed as follows. First, the weighted median interval of the x-(y-) coordinates of its fan-in and fan-out pins are calculated, where the weight of a pin is the switching rate of the signal net between the MBFF and the pin. Then, the preferred region is formed by the intersection of these two weighted median intervals.

We divide the placement area into bins. The bins covered by the preferred region of an MBFF are called its preferred bins. The rank of the preferred bins of an MBFF is set to 0. Increasing ranks are given to other bins within the feasible region of an MBFF[7] according to their distance from its preferred bins. Each MBFF is assigned to the lowest-ranked bin inside its feasible region. The MBFF will be assigned to a vacant slot within the selected bin that is closest to the preferred region. In our experiments, the slot size is set to the size of 2x-driving 2-bit MBFF. This is to ensure that there will be a minimal overlap of the MBFFs with other cells so that any overlap can be resolved using a commercial tool with little disturbance to the original placement in order to preserve timing.

3.5 Timing-Driven MBFF Sizing

After the location assignment is completed, the netlist and placement are updated and saved in the same formats used by a commercial tool, such as DEF and Verilog files. Placement legalization and timing analysis will then be done by a commercial tool. We note that so far all bits in the same MBFF have identical driving strength. After timing analysis, we will try to downsize some bits of an MBFF if possible to save more power while maintaining zero negative slack.

Without loss of generality, we assume that a mixed-driving MBFF mixes two driving strengths, low and high. Given an MBFF where all bits have high driving strength initially, we try to replace it with a mixed-driving MBFF having low and high driving strengths as shown in Algorithm 2. Bits with large timing slacks are candidates to be downsized. To reach the lowest power consumption possible without violating timing constraints in one shot, we determined the optimal value for β used in line 3 empirically. We tried different values of β from 1 to 0.85 with a 0.01 step size, and the optimal value was found to be 0.95. If the timing slack of an output pin is larger than β times the average slack of all MBFF outputs in the design, the driving strength of that bit can be safely lowered to save power in lines 2-5. Besides, all empty bits in an MBFF can always be downsized. After downsizing the appropriate bits, an initial high-driving MBFF will be replaced by a mixed-driving MBFF with l 1ow-driving bits and h high-driving bits which are always no bigger than the initial MBFF in the area so that no further placement legalization is necessary.

[7]The feasible region of an MBFF is the intersection of the feasible regions of its corresponding FFs.

Table 1: MBFF Library.

#bit	normalized power per bit	normalized area per bit	driving	power(μW)	area(μm^2)
1	1.00	1.00	1x	100	16.245
			2x	170	17.382
2	0.70	0.96	1x	140	31.190
			2x	238	33.374
4	0.60	0.72	1x	240	46.786
			2x	408	50.061
8	0.50	0.65	1x	400	83.174
			2x	680	88.997

Algorithm 2 Timing-Driven MBFF Downsizing

Input: A k-bit high-driving strength MBFF MBk_H.
Average output pin slack avg_{slack} of all MBFFs formed.
Output: A mixed-driving strength MBFF MBl_Lh_H.
1: $l \leftarrow 0$ // l is #bits that can be downsized
2: **for** each output pin q in MBk_H **do**
3: **if** $q_{slack} > \beta \times avg_{slack}$ **then**
4: Reduce bit strength for q
5: $l \leftarrow l + 1$
6: **if** MBk_H is an MBFF with empty bits **then**
7: Reduce bit strength for each empty bit
8: $l \leftarrow l$ + #empty bits in MBk_H
9: $h \leftarrow k - l$
10: **return** MBl_Lh_H

4 EXPERIMENTAL RESULTS

The FF merging and MBFF sizing steps of our approach are implemented using C++ and the overall approach is tested on a 64-bit Ubuntu server with ThreadRipper 3970X and 256GB memory. The ICCAD 2015 contest benchmarks [12] are used in the experiments, and all FFs in each benchmark are of 1-bit and 1x-driving strength. The clock period is modified to remove the timing violations in each benchmark. An MBFF library containing 1x- and 2x-driving 1/2/4/8-bit MBFFs, as shown in Table 1, is adopted in our experiments. In addition, the library also includes all types of mixed-driving MBFFs whose power and area are estimated by a linear scaling method based on their 1x- and 2x-driving counterparts.[8] For example, for 4-bit mixed-driving MBFFs, three mixed-driving types, i.e., $MB3_{1x}1_{2x}$, $MB2_{1x}2_{2x}$, and $MB1_{1x}3_{2x}$ are included, and their power values are $282\mu W$, $324\mu W$, and $366\mu W$, respectively. Our approach is combined with Cadence Innovus [8] which is utilized to perform placement legalization, timing analysis, CTS, and routing.

4.1 Comparison with Mean Shift [2]

Mean Shift [2] is one of the most recent works on FF clustering that forms and re-locates clusters of FFs according to the FF distributions in a placement, and it is also one of the methods used in DATC-RDF 2020 [4] to help generate MBFFs with the OPENROAD design flow [1]. However, since Mean Shift only moves multiple 1x-driving 1-bit FFs into one cluster without timing violations, no MBFFs are generated. Thus, a post-processing merging method is implemented by us and for it to generate MBFFs based on its clustering results. For example, if a FF cluster produced by Mean Shift has a size equal to 6,

[8]The MBFF library is designed based on the 1x-driving FF cell "DFF_X80" in ICCAD 2015 contest benchmarks. To save space, mixed-driving MBFFs are not shown in Table 1.

Table 2: Comparing the clock sink count, total FF area, and total FF power obtained by running Mean Shift to those obtained by running our approach.

Benchmarks	Original (1x-driving FFs only)			Mean Shift (1x-driving MBFFs only)			Ours (1x & 2x-driving MBFFs)				
									FF power (W)		
	#sinks	FF area (μm^2)	FF power (W)	#sinks	FF area (μm^2)	FF power (W)	#sinks	FF area (μm^2)	No mixed-driving MBFFs	With mixed-driving MBFFs	
										Case1 (all types)	Case2 (one type)
superblue1	144266	2.34	14.43	42522	1.73	8.70	23682	1.57	8.12	7.64	7.66
superblue3	167923	2.73	16.79	48091	2.00	10.06	24463	1.79	9.10	8.67	8.69
superblue4	176895	2.87	17.69	53794	2.13	10.77	31992	1.96	10.32	9.63	9.67
superblue5	114103	1.85	11.41	32010	1.36	6.79	18936	1.24	6.44	6.03	6.04
superblue7	270219	4.39	27.02	77098	3.22	16.17	50874	3.02	16.19	14.83	14.88
superblue10	241267	3.92	24.13	68837	2.87	14.42	36031	2.59	13.15	12.52	12.55
superblue16	142543	2.32	14.25	37254	1.68	8.33	29034	1.62	8.69	7.89	7.91
superblue18	103544	1.68	10.35	28645	1.23	6.15	17106	1.13	5.92	5.54	5.57
Norm. Avg.	1	1	1	0.284	0.733	0.598	0.170	0.675	0.572	0.534	0.536

Table 3: Comparing the CTS results obtained by running Mean Shift to those obtained by running our approach.

Benchmarks	Original (1x-driving FFs only)			Mean Shift (1x-driving MBFFs only)			Ours (1x & 2x-driving MBFFs)		
	clock WL (um)	#buffers	clock cap (pF)	clock WL (um)	#buffers	clock cap (pF)	clock WL (um)	#buffers	clock cap (pF)
superblue1	1050438	4037	425811	601957	2119	179695	550550	1690	140638
superblue3	1067898	4671	469028	619060	2244	193692	544870	2047	143864
superblue4	1008295	4542	468146	587658	2036	195432	533487	1648	149021
superblue5	929333	4043	359005	567367	2197	158013	543751	2162	135204
superblue7	1704440	7121	746192	963531	3206	301213	929400	2781	252873
superblue10	1611340	6817	689520	929246	3168	285720	819056	2658	210580
superblue16	937115	3900	405910	530077	1760	156463	519620	1518	144049
superblue18	628651	2659	281784	354889	1140	111890	324052	974	86763
Norm. Avg.	1	1	1	0.578	0.474	0.412	0.537	0.411	0.330

it will be formed into a 4-bit MBFF and a 2-bit MBFF of 1x-driving. In order for our approach to have a fair comparison with Mean Shift that considers such objectives as FF power, clock wirelength, and clock buffer count, the user-defined parameter λ_3 in our proposed cost function (i.e., Equation (1)) is set to 0. Additionally, setting the other two parameters λ_1 and λ_2 to 1.5 and 0.21 respectively can help our approach produce the best results.

Table 2 shows and compares the results of the clock sink count, the total FF area, and the total FF power for the original benchmarks (i.e., the input benchmarks) and those produced by Mean Shift and our approach.[9] Since the results of the total FF area before and after downsizing by our approach do not have a significant difference, the table only shows the results before downsizing. Compared with the original benchmarks, our approach greatly reduced the sink count, the total FF area, and the total FF power by 83%, 32.5%, and 46.6%, respectively. Besides, compared with Mean Shift, our approach achieves 11.4% fewer sinks, 5.8% less FF area, and 2.6% less FF power before timing-driven MBFF downsizing; after timing-driven MBFF downsizing, the FF power reduction is increased to 6.4% as shown in the column of "Case 1". On average, near 90% of 2x-driving MBFFs can be downsized to mixed-driving MBFFs and save more power. As for the runtime, our approach is comparable to Mean Shift, and it takes less than four minutes to perform the merging and sizing steps for the most complex benchmark. Moreover, its runtime becomes much less significant when compared with the time spent on CTS and routing which can take up to more than eight hours. Note that our approach also works when only 1x-driving MBFFs are available, and it achieves similar merging efficacy as Mean Shift, but due to the page limit, its results are omitted.

Table 3 illustrates the CTS results in terms of the clock wirelength (clock WL), the clock buffer count (#buffer), and the clock capacitance (clock cap). Compared with the original benchmarks, the clock wirelength, the clock buffer count, and the clock capacitance are reduced by 46.3%, 58.9%, and 67%, respectively, by our approach. In addition, 4.1% less clock wirelength, 6.3% fewer clock buffer count, and 8.2% less clock capacitance, are achieved by our approach when compared to Mean Shift.

4.2 Limited Selection of Mixed-Driving MBFFs

To see whether it is necessary to include all types of mixed-driving MBFFs in the library, we consider a scenario where only one type of mixed-driving MBFF is available for every k-bit MBFF.

By conducting experiments on the same set of benchmarks, we find that $MB1_{1x}1_{2x}$, $MB3_{1x}1_{2x}$, and $MB6_{1x}2_{2x}$ can be respectively chosen to be the only available 2-bit, 4-bit and 8-bit mixed-driving MBFFs as they help the downsizing step produce the best FF power results overall. We may need a slight modification when downsizing an MBFF. For example, if a 2x-driving 8-bit MBFF has 7 bits that can be downsized, it will be converted to $MB6_{1x}2_{2x}$, instead of $MB7_{1x}1_{2x}$, since only $MB6_{1x}2_{2x}$ is available. The results are shown

[9]Note that the placement and routing results of each benchmark produced by our approach and Mean Shift all meet timing.

Table 4: Comparing the signal switching power and normalized total power obtained by running Mean Shift to those obtained by running our approach.

Benchmarks	Original		Mean Shift		Ours w/o considering SSP		Ours considering SSP					
									FF power (W)			
	SSP (W)	Total Norm.	SSP (W)	Total Norm.	SSP (W)	Total Norm.	#sinks	FF area (μm^2)	No mixed-driving MBFFs	With mixed-driving MBFFs	SSP (W)	Total Norm.
superblue1	2.86	1	2.96	0.67	3.04	0.61	24104	1.58	8.15	7.68	3.00	0.61
superblue3	4.09	1	4.18	0.68	4.28	0.62	24782	1.80	9.14	8.72	4.24	0.62
superblue4	2.89	1	2.97	0.67	2.99	0.61	32124	1.96	10.33	9.67	2.98	0.61
superblue5	3.15	1	3.24	0.69	3.28	0.64	19387	1.25	6.48	6.07	3.26	0.64
superblue7	5.84	1	5.99	0.67	6.04	0.63	51552	3.03	16.24	14.94	6.01	0.63
superblue10	5.26	1	5.35	0.67	5.46	0.61	36344	2.59	13.17	12.56	5.39	0.61
superblue16	2.10	1	2.21	0.64	2.23	0.62	29422	1.62	8.72	7.93	2.21	0.62
superblue18	1.72	1	1.78	0.65	1.81	0.61	17366	1.13	5.93	5.58	1.79	0.61
Norm. Avg.	1	1	1.031	0.667	1.046	0.617	0.173	0.677	0.574	0.537	1.037	0.618

Table 5: Experimental results of our approach that considers leaving empty bits in an MBFF.

Benchmarks	Ours with empty bits allowed						
			FF power (W)				
	#sinks	FF area (μm^2)	No mixed-driving MBFFs	With mixed-driving MBFFs	clock WL (um)	#buffers	clock cap (pF)
superblue1	21464	1.58	7.87	7.64	533096	1651	133185
superblue3	22857	1.78	8.85	8.61	536854	1736	137868
superblue4	28076	1.96	9.86	9.56	508954	1568	138936
superblue5	17209	1.24	6.22	6.01	526789	2118	128385
superblue7	43763	3.00	15.18	14.59	891358	2666	233701
superblue10	33514	2.57	12.78	12.42	806080	2621	205839
superblue16	24560	1.60	8.12	7.81	493856	1495	132489
superblue18	15043	1.12	5.61	5.43	317125	910	82772
Norm. Avg. compared with Original	0.152	0.672	0.547	0.530	0.517	0.392	0.312

in the "Case 2(one type)" column of Table 2, which indicates that the total FF power only degrades by 0.2% when compared with the case where all mixed-driving types are available (see the results in the "Case 1(all types)" column of Table 2). Thus, even if a limited selection of mixed-driving MBFFs is provided, it can still closely approximate the ideal result.

4.3 Considering Signal Switching Power

To consider signal switching power, we set the three parameters λ_1, λ_2, and λ_3 in the cost function to 1.2, 0.01, and 1.0, respectively as these values can help our approach produce the best signal switching power results. The routed wirelength of each net connecting one or more FFs is generated from Innovus.

Table 4 shows and compares the results of the signal switching power (SSP) and the normalized total power (Total Norm.) which is the normalized sum of the total FF power, the signal switching power, and the clock power. By considering the signal switching power in the cost function, the number of sinks and the FF power will be slightly increased when compared with the one that does not (cf. Table 2), but the signal switching power has a 0.9% improvement and is more similar to that of Mean Shift. The number of clock sinks produced by our approach is 11.1% less than Mean Shift. The better sink count reduction will lead to an increase in the wirelength of the nets connecting FFs, inevitably resulting in a little bit higher signal switching power than Mean Shift. Nevertheless, our approach still achieves 6.1% less FF power and 4.9% less normalized total power than Mean Shift.

4.4 Considering MBFFs with Empty Bits

As mentioned before, the number of sinks and power can be further reduced if we allow an MBFF to have empty bits. To compare with the experimental results of our approach that does not consider MBFFs with empty bits in Table 2, λ_1, λ_2, and λ_3 in the cost function are set to the same values as those in Section 4.1 when using our approach that allows MBFFs with empty bits.

Table 5 shows the experiment results of our approach by leaving some empty bits in an MBFF, and the normalized average values in the last row are calculated by comparing with the original benchmarks. On average, 5% of MBFFs have empty bits. The number of sinks and the total FF power before downsizing can be further reduced by 1.8% and 2.5%, respectively. The CTS results in terms of clock tree wirelength, clock buffer count, and clock capacitance also have 2%, 1.9%, and 1.8% improvements. Therefore, leaving some empty bits in an MBFF can have some benefits as expected.

5 CONCLUSION

In this paper, we present the first mixed-driving MBFF generation approach for effective power reduction. Our approach is able to leave some empty bits in MBFFs to achieve further improvements. Experimental results show huge advantages of our approach over a recent work [2] that considers only uniform-driving MBFFs in terms of the clock sink count, the FF power, the clock buffer count, and the routed clock wirelength.

REFERENCES

[1] T Ajayi, D Blaauw, TB Chan, CK Cheng, VA Chhabria, DK Choo, M Coltella, S Dobre, R Dreslinski, M Fogaça, et al. 2019. OpenROAD: Toward a Self-Driving, Open-Source Digital Layout Implementation Tool Chain. In *Proceedings of Government Microcircuit Applications Critical Technology Conference*. 1105–1110.

[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 International Symposium on Physical Design*. 11–18.

[3] Yao-Tsung Chang, Chih-Cheng Hsu, Mark Po-Hung Lin, Yu-Wen Tsai, and Sheng-Fong Chen. 2010. Post-Placement Power Optimization with Multi-Bit Flip-Flops. In *Proceedings of International Conference on Computer-Aided Design*. 218–223.

[4] Jianli Chen, Iris Hui-Ru Jiang, Jinwook Jung, Andrew B. Kahng, Victor N. Kravets, Yih-Lang Li, Shih-Ting Lin, and Mingyu Woo. 2020. DATC RDF-2020: Strengthening the Foundation for Academic Research in IC Physical Design. In *Proceedings of International Conference on Computer-Aided Design*. Article 71, 6 pages.

[5] Sheng-Hsiung Chen, Shao-Huan Wang, Wen-Hao Chen, Chun-Yao Ku, and Hung-Chih Ou. 2019. Integrated Circuit And Method Of Forming Same And A System. US Patent US10990745B2.

[6] Wenting Hou, Dick Liu, and Pei-Hsin Ho. 2009. Automatic Register Banking for Low-Power Clock Trees. In *Proceedings of International Symposium on Quality Electronic Design*. 647–652.

[7] Hiroshi Imai and Takao Asano. 1983. Finding the Connected Components and a Maximum Clique of an Intersection Graph of Rectangles in the Plane. In *Proceedings of Journal of Algorithms*, Vol. 4. 310–323.

[8] Cadence Innovus. 2020ver. http://www.cadence.com

[9] Iris Hui-Ru Jiang, Chih-Long Chang, and Yu-Ming Yang. 2012. INTEGRA: Fast Multibit Flip-Flop Clustering for Clock Power Saving. In *Proceedings of IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 31. 192–204.

[10] Andrew B. Kahng, Jiajia Li, and Lutong Wang. 2016. Improved Flop Tray-Based Design Implementation for Power Reduction. In *Proceedings of International Conference on Computer-Aided Design*.

[11] Ajay Kapoor, Cas Groot, Gerard Villar Piqué, Hamed Fatemi, Juan Echeverri, Leo Sevat, Maarten Vertregt, Maurice Meijer, Vibhu Sharma, Yu Pu, and José Pineda de Gyvez. 2014. Digital Systems Power Management for High Performance Mixed Signal Platforms. In *Proceedings of IEEE Transactions on Circuits and Systems I: Regular Papers*, Vol. 61. 961–975.

[12] Myung-Chul Kim, Jin Hu, Jiajia Li, and Natarajan Viswanathan. 2015. CAD Contest in Incremental Timing-Driven Placement and Benchmark Suite. In *Proceedings of International Conference on Computer-Aided Design*. 921–926.

[13] Synopsys Liberty. https://www.synopsys.com/community/interoperability-programs/tap-in.html

[14] Mark Po-Hung Lin, Chih-Cheng Hsu, and Yu-Chuan Chen. 2015. Clock-Tree Aware Multibit Flip-Flop Generation During Placement for Power Optimization. In *Proceedings of IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 34. 280–292.

[15] Sean Shih-Ying Liu, Wan-Ting Lo, Chieh-Jui Lee, and Hung-Ming Chen. 2013. Agglomerative-Based Flip-Flop Merging and Relocation for Signal Wirelength and Clock Tree Optimization. In *Proceedings of ACM Transactions on Design Automation of Electronic Systems*, Vol. 18. New York, NY, USA, Article 40, 20 pages.

[16] Ioannis Seitanidis, Giorgos Dimitrakopoulos, Pavlos M. Mattheakis, Laurent Masse-Navette, and David Chinnery. 2019. Timing-Driven and Placement-Aware Multibit Register Composition. In *Proceedings of IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 38. 1501–1514.

[17] Michael Ian Shamos and Dan Hoey. 1976. Geometric Intersection Problems. In *Proceedings of Symposium on Foundations of Computer Science*. 208–215.

[18] Ya-Ting Shyu, Jai-Ming Lin, Chun-Po Huang, Cheng-Wu Lin, Ying-Zu Lin, and Soon-Jyh Chang. 2013. Effective and Efficient Approach for Power Reduction by Using Multi-Bit Flip-Flops. In *Proceedings of IEEE Transactions on Very Large Scale Integration Systems*, Vol. 21. 624–635.

[19] Chang-Cheng Tsai, Yiyu Shi, Guojie Luo, and Iris Hui-Ru Jiang. 2013. FF-Bond: Multi-Bit Flip-Flop Bonding at Placement. In *Proceedings of International Symposium on Physical Design*. 147–153.

[20] Shao-Huan Wang, Yu-Yi Liang, Tien-Yu Kuo, and Wai-Kei Mak. 2012. Power-Driven Flip-Flop Merging and Relocation. In *Proceedings of IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, Vol. 31. 180–191.

[21] 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 Design Automation Conference*.

[22] Dongyoun Yi and Taewhan Kim. 2016. Allocation of Multi-Bit Flip-Flops in Logic Synthesis for Power Optimization. In *Proceedings of International Conference on Computer-Aided Design*.