Ph.D. DISSERTATION

Storage Synthesis and Optimization Algorithms for High-speed and Low-power Chips

고성능 및 저전력 칩을 위한 저장 공간 합성 및 최적화 알고리즘

BY

SOOMIN KIM

AUGUST 2023

DEPARTMENT OF ELECTRICAL AND
COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Ph.D. DISSERTATION

Storage Synthesis and Optimization Algorithms for High-speed and Low-power Chips

고성능 및 저전력 칩을 위한 저장 공간 합성 및 최적화 알고리즘

BY

SOOMIN KIM

AUGUST 2023

DEPARTMENT OF ELECTRICAL AND
COMPUTER ENGINEERING
COLLEGE OF ENGINEERING
SEOUL NATIONAL UNIVERSITY

Storage Synthesis and Optimization Algorithms for High-speed and Low-power Chips

고성능 및 저전력 칩을 위한 저장 공간 합성 및 최적화 알고리즘

지도교수 김 태 환

이 논문을 공학박사 학위논문으로 제출함

2023년 8월

서울대학교 대학원

전기 · 정보 공학부

김 수 민

김수민의 공학박사 학위 논문을 인준함

2023년 8월

위 원 장:	김 재 하
부위원장:	김 태 환
위 원:	김 장 우
위 원:	최 규 명
위 원:	김 용 환

Abstract

In the physical design of high-speed and low-power design implementation, multi-bit flip-flop synthesis and retention storage allocation problem are important issues. This dissertation presents two methodologies related to storage synthesis and allocation that can contribute to improving the performance and power consumption of the target design.

Firstly, we propose a design and technology co-optimization (DTCO) flow utilizing multi-bit flip-flop cells in a way to enhance routability and timing at the placement and routing stages. Precisely, we make the non-flexible MBFF cell flipping to be fully flexible by generating MBFF layouts supporting diverse D-to-Q flow directions, and enhance the setup and clock-to-Q delay on timing critical flip-flops in MBFF through gate upsizing (i.e., transistor folding) by using the unused space in MBFF. Through experiments with benchmark circuits in advanced node, it is shown that our proposed DTCO flow using MBFFs is very promising to improve routability and timing slack in chip implementation.

Secondly, we propose an optimal solution to the problem of allocating state retention storage in power gated circuit. Precisely, we transform the allocation problems constraining the wakeup latency constraint l to 2 and 3 clock cycles into unate covering problems and solve them optimally with three objective options: minimizing total bits of retention storage, directly minimizing total leakage power consumed by retention storage, and minimizing total implementation area of retention storage. Through experiments with benchmark circuits, it is shown that our optimal algorithm is able to further reduce the total bits of retention storage, the leakage power on retention storage, and the retention storage area while l is set to 3 over that produced by the conventional best-known allocation heuristic.

keywords: multi-bit flip-flop synthesis, DTCO, transistor sizing, retention storage allocation, power gating, leakage power
student number: 2018-20366

Contents

List of Tables

List of Figures

Chapter 1

Introduction

1.1 Multi-bit Flip-flop Methodology

With a set of limited power and thermal budget constraints for modern ASIC designs that are to be implemented with a huge number of transistors and interconnects, minimizing the amount of power consumption is one of the utmost important design objectives for diverse applications. Diverse methodologies aimed at minimizing power consumption have been proposed, including clock gating [7], employment of multi-VT cells [8], dynamic voltage and frequency scaling(DVFS) [9], and utilization of multi-bit registers [10]. Among them, one effective method to reduce power consumption particularly on the flip-flops and driving clock networks is to use *multi-bit flip-flop* (MBFF) (also called *register bank* and *multi-bit bank*) cells [11, 12, 10, 13, 14, 15, 16, 17, 18, 19, 20, 21, 22, 23].

Fig. 1.1 compares the internal structures of two 1-bit flip-flops (left side) and their functionally equivalent 2-bit MBFF (right side). The power saving on the 2-bit MBFF is attributed by the sharing of the two clock inverters among the two master and two slave latches. It is reported that 2-bit MBFF can save power by 14%, at the same time, reducing the area by 4% [21]. Furthermore, as the number of inverters within the flip-flops decreases, the leaf nodes of the clock tree also decrease, simplifying the structure

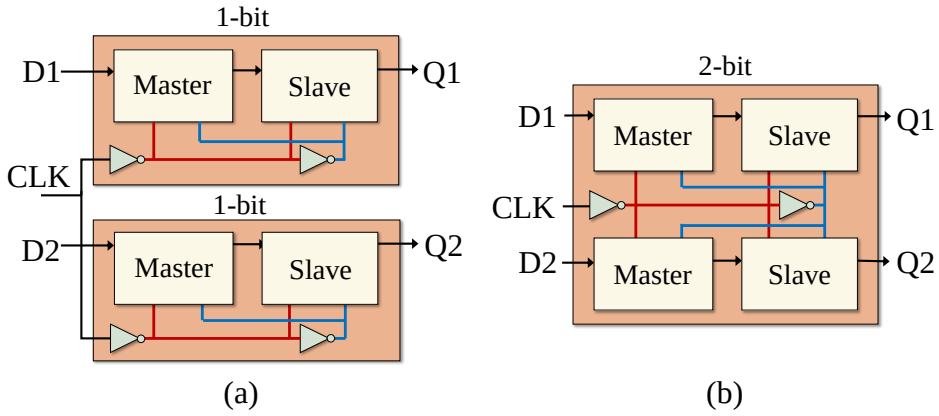
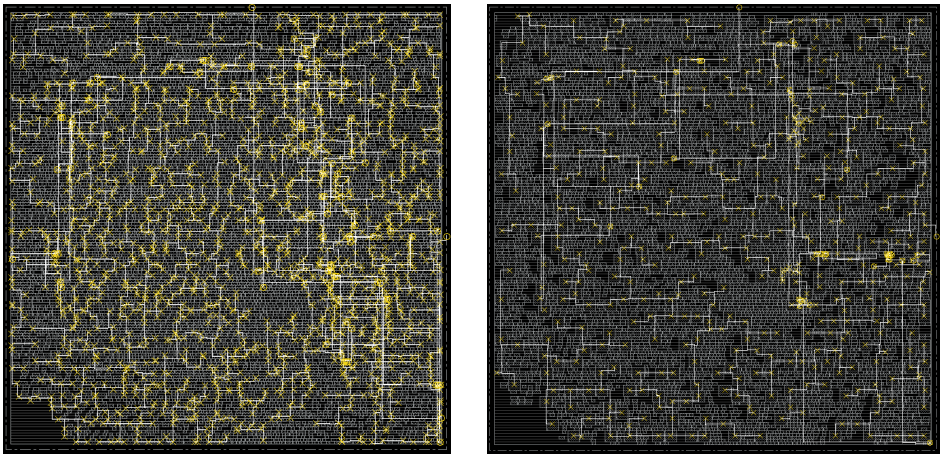


Figure 1.1: (a) Structure of two 1-bit flip-flops. (b) Structure of 2-bit MBFF merging the two 1-bit flip-flops in (a).

of the clock tree as illustrated in Fig. 1.2. The number of clock tree buffers can also be significantly reduced, leading to a potential reduction of up to 40% in clock power consumption, according to the recent Polaris architecture [24] for GPUs (Here, clock power is known to account for 20-35% of the total chip power, indicating its significant impact on power consumption.).

Extensive studies have been conducted for grouping single-bit flip-flops to generate multi-bit flip-flops. [20, 21] grouped single-bit flip-flops during the logic synthesis step. The work in [21] transformed the flip-flop grouping problem into an instance of set cover problem on a conflict graph with the objective of minimizing flip-flop and clock tree power consumption. In [10], flip-flops were grouped using the BFS algorithm in gate-level designs, which resulted in advantageous control of the scan ring structure and clock skew. There are also studies that conducted flip-flop grouping during the placement stage [17, 16, 18]. The work in [18, 16] took into account saving flip-flop power as well as reducing clock latency in placement. In [11, 22, 23, 12, 14], MBFF grouping was conducted during the post-placement stage, which consisted of two major steps: flip-flop clustering and MBFF placement. Clustering involves identifying the feasible placement regions for each flip-flop and grouping together the flip-



(a) Clock tree structure using 1-bit flip-flops

(b) Clock tree structure using 4-bit multi-bit flip-flops

Figure 1.2: Comparison of clock tree structure of circuit using (a) 1-bit flip-flops and (b) 4-bit multi-bit flip-flops.

flops that share common regions. The feasible region for each flip-flop is the intersection of the regions that indicate the maximum allowable distance (usually calculated based on timing slack) from all the pins connected to the flip-flop. In [11, 22, 12, 14], they used graph-based clustering with the goal of reducing total flip-flop power and wirelength. The work in [23] utilizes graph-based clustering to minimize the total number of clock sinks and the total net switching power. MBFF placement involves considering placement density constraints. [14] proposed a placement algorithm that includes capacity-constrained signal rerouting to improve routability. [11, 12] placed MBFFs by creating an optimal bounding box using the coordinates of fan-in and fan-out pins to minimize total wirelength. [23] proposed a weighted median interval to minimize weighted total wirelength, that is total net switching power, for MBFF placement. Recently, [25] tried to debank MBFFs to enhance the flexibility of the application of useful clock skew scheduling and [19] used graph-based clustering with mixed-driving MBFF to minimize the total number of clock sinks and the total net switching power at the post-placement stage. However, to our knowledge, no work has considered the

diversification of MBFF layouts to enable design and technology co-optimization.

In practice, increasing the size of MBFF to accommodate many flip-flops imposes two new challenging problems in physical design, which are (1) *non-flexible MBFF cell flipping* and (2) *unbalanced or wasted use of MBFF footprint space*. Details of the two problems are described in the next two paragraphs.

MBFFs are in general constructed by stacking single-bit flip-flops to retain the regularity of the internal cell layout. (One well-known example is shown in the TSMC patent in [26].) This implies that the cell flipping, which has been used as a useful technique in placement and routing to optimize design parameters (e.g., wirelength and routability), is not always effectively applicable to MBFF cells. This is because an MBFF cell flipping exactly means reversing *all* D-to-Q flow directions in the cell and it does not mean *selectively* reversing the individual flow directions. For example, for a 2-bit MBFF cell instance in a placement, shown in Fig. 1.3(a), with predictable routability, either unflipping (upper one) or flipping (lower) the cell never help to improve the routability to *both* flip-flops in the cell.

The second problem is space waste on MBFF footprint, as illustrated in Fig. 1.3(b). Since one non-dummy gate poly is enough to implement each clock inverter in an MBFF, noting that the advance of process technology below 64nm enables even a minimum sized inverter to still drive multiple master and slave latches in flip-flops without input slew violation [11], the space to occupy exactly two (non-dummy) gate polys suffice to deploy the two internal clock inverters. As a result, for a 4-bit MBFF cell shown in Fig. 1.3(b), the cell layout induces two empty spaces (gray color), one in flip-flop f_1 and the other in flip-flop f_4.

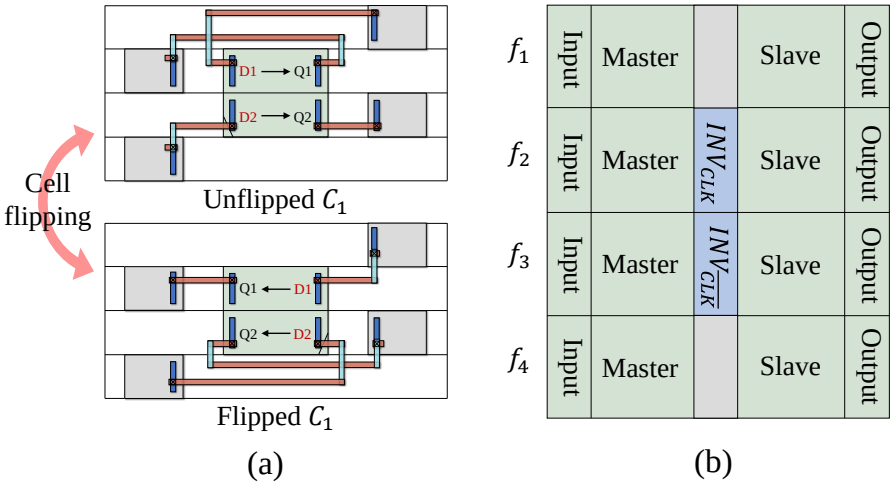


Figure 1.3: Examples illustrating two physical limitations that hinder an extensive use of MBFF cells. (a) Non-flexible cell flipping. (b) Space waste in cell layout.

1.2 State Retention Storage Allocation on Power Gated Circuit

Minimizing leakage power of circuit has been a major issue in according to the semiconductor process node shrinking. One of the technologies to reduce leakage power consumption is power gating, which is shutting down the power supply or ground on circuit so that the circuit no more consumes the power. The power gated circuit requires to save its flip-flops' states before being shut down to enter sleep mode, so that the circuit should continue execution from the saved states when waking up.

Broadly, there are two ways to save and restore states in power gating. One is using scan chain to store all states in memory. Once the states are saved, the circuit is shut down and there are no always-on parts in the circuit if an external memory is used for the state retention. The other is using state retention flip-flops that are composed of the original flip-flops and always-on state retention storages [27, 28].

Replacing every original flip-flop in circuit with a flip-flop having a distinct 1-bit retention storage is called SBRFF (*single bit retention flip-flip*) allocation [27]. Thus,

for a circuit with n flip-flops, they use n total number of retention bits. However, replacing all original flip-flops with 1-bit retention flip-flops occupies a considerable area, so consuming significant always-on leakage power. It is shown that 1-bit retention flip-flop has generally 20% more area than the original flip-flop [27].

Many researches have been performed to reduce the amount of retention storage. The works in [29, 30, 31] proposed so-called SSRPG (*selective state retention power gating*), which allocates retention storage only to some essential flip-flops to satisfy the perfect state recovery. SSRPG assumes power gating to be applied to given a small number of FSM states, equivalently a few known checkpoints in RTL code. Thus, the objective of SSRPG is to identify the (non-essential) flip-flips unrelated to the FSM states to exclude them from state retention. They used simulation and formal verification to extract essential flip-flops for which a full knowledge of the behavior of target circuit or RTL code is required in advance. Chen *et al.* [3] proposed to use multi-bit retention flip-flop (MBRFF). Fig. 1.4 shows the internal structure of a k-bit MBRFF. The k-bit shift retention storage can save up to k consecutive states of the original flip-flop. The always-on retention storage is implemented with high V_{th} transistors to reduce always-on leakage power [32]. The control logic generates signals to control state saving or restoration when the circuit goes to sleep mode or active mode. Since k clock cycles are required for each of state saving and restoration for MBRFFs while one clock cycle suffices for SBRFFs and SSRPG, it is necessary to use a small k value of MBRFFs at the expense of retention storage. The works in [1, 33] proposed MBRFF allocation algorithms that try to extract a minimal number of flip-flops, each of which is to be replaced with strictly k-bit MBRFF, and the rest are left as they are.

Unlike the methods of *uniform* MBRFF allocation in [1, 33], Fan *et al.* [2] proposed a *non-uniform* MBRFF allocation algorithm which determines the size of retention storage of the individual flip-flops so that the total retention bits is minimized. For example, Fig. 1.6(a) shows a uniform MBRFF allocation result for a flip-flop depen-

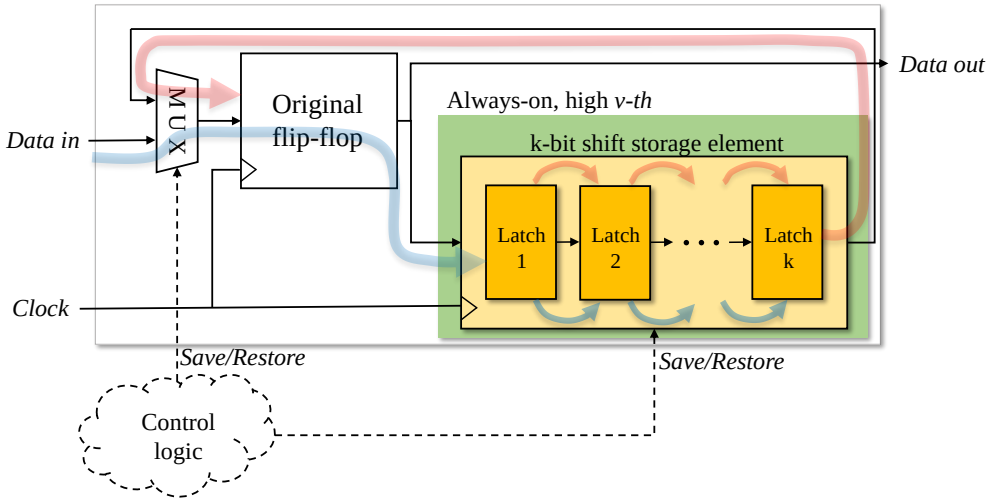


Figure 1.4: The internal structure of k-bit MBRFF in [3]. The blue and red translucent lines indicate the flow of state saving and restoration, respectively.

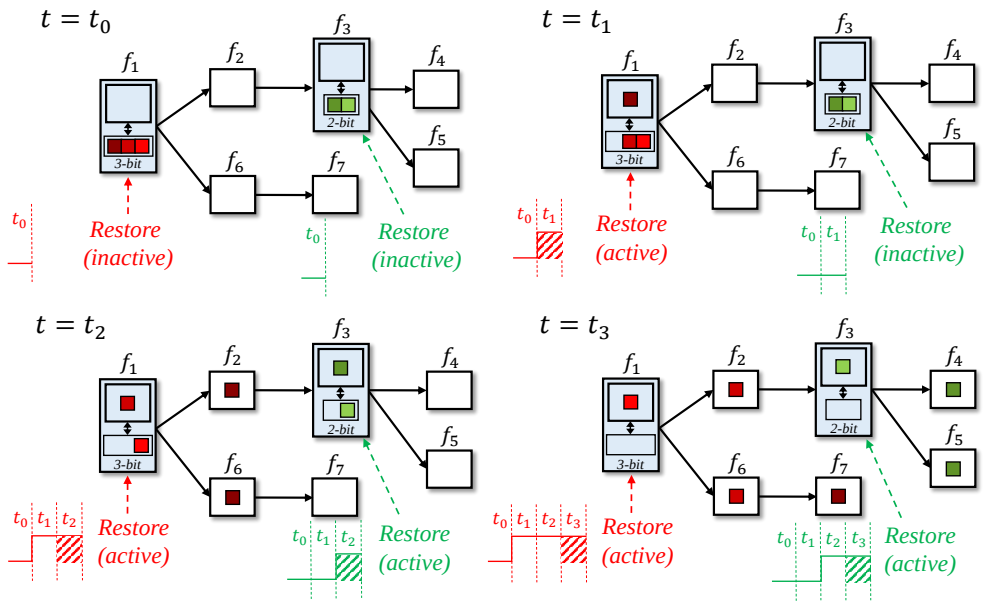
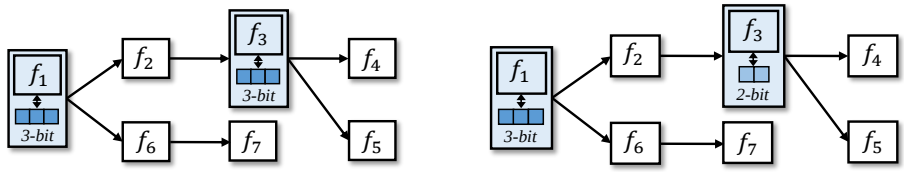


Figure 1.5: Illustration of cycle-by-cycle state restoration for the MBRFF allocation. Initially, at time t_0 flip-flip f_1 retains 3 states and f_3 2 states. During the following three cycles, the states of flip-flops are set through the retention storage or logic propagation from their driving flip-flops.

dency graph[1] in which f_1 and f_3 are replaced with 3-bit MBRFFs though f_3 can be replaced with 2-bit MBRFFs as shown in Fig. 1.6(b).



(a) Uniform MBRFF allocation: 6 total bits, but enabling simple control network

(b) Non-uniform MBRFF allocation: 5 total bits, but increasing control network overhead

Figure 1.6: Comparison of uniform and non-uniform MBRFF allocations in terms of total retention bits and control network overhead.

Recently, a new MBRFF allocation algorithm was proposed by [34], introducing 2-phase operation for single-bit retention flip-flops, so that their state save/recovery can be done either in the first or in the second clock step. However, supporting this 2-phase operation requires exactly 2 clock cycles of save/restoration and demands more complicated control logic.

Fig. 1.5 illustrates a cycle-by-cycle state restoration process for an MBRFF allocation. Initially, at time t_0, flip-flip f_1 retains 3 states and f_3 2 states. At the first cycle (t_1) of restoration, the state of f_1 is set and at the second cycle (t_2), the states of f_2 and f_6 driven from f_1 and the state of f_3 are set. Lastly, at the third cycle, the state of f_7 driven from f_6 and the states of f_4 and f_5 driven from f_3 are set. The state restoration process for MBRFF allocation entails two overheads:

1. (*Long wakeup latency*) The wakeup latency in MBRFF allocation should be at least equal to or greater than the largest bit size of retention storage among the MBRFFs produced by the allocation.

[1]The flip-flop dependency graph of a circuit is a directed graph $G(V, A)$ where nodes in V indicate distinct flip-flops and there is an arc $(f_i, f_j) \in A$ if $f_j \in V$ is driven from a combinational logic path from $f_i \in V$.

2. (*More control signals*) Distinct wakeup control signals to MBRFFs are required if the sizes of their retention storage are different.

Consequently, to reduce the two overheads, it is highly desirable to allocate MBRFFs such that the maximum bit size among retention storages as well as the number of MBRFF groups according to the storage bit size is as small as possible.

The conventional MBRFF allocation algorithms ([3, 1, 33, 2]) have two fundamental limitations in common:

1. (*No cycle, including self-loop*) The allocation algorithms accept a flip-flop dependency graph with no cycle as input. This means the original flip-flop dependency graph of circuit with cycle should be transformed into an acyclic one before applying the allocation algorithms. However, circuits normally contain many self-loops mostly corresponding to mux-feedback loops in flip-flops in circuits[2] as well as many flip-flop cycles, which could clearly worsen the quality of MBRFF allocation.

2. (*Heuristic*) All existing allocation algorithms are in fact heuristic. So far, there is no way to find out how much the allocation results are close to an optimal one in terms of total bits of retention storage, storage leakage power, and storage area.

[2]Most self-loop flip-flops are synthesized from the *if-statements with no else-part* in HDL description.

1.3 Contributions of this Dissertation

In this dissertation, we present several methodologies related to storage synthesis and optimization algorithms to improve the performance and power consumption of target design.

In Chapter 2, we propose a design and technology co-optimization (DTCO) techniques that are able to effectively resolve the two limitations imposed by MBFFs [35, 36]. The main contributions of this work can be summarized as:

1. To overcome the less flexible cell flipping problem, we introduce a new concept of *selective D-to-Q flow flipping* and propose to synthesize MBFF cells of various combinations of D-to-Q flow flipping and non-flipping. Then, we propose a cost-based algorithm, for each MBFF cell instance in placement, to find and synthesize an MBFF of flow flipping/non-flipping combination that leads to a maximal routability benefit by cell instance replacement.

2. To further enhance the timing performance of the chip by utilizing various combinations of D-to-Q flow, we refine the D-to-Q flow, in global routing, on the basis of the data extracted from the timing and net congestion analysis. We propose an iterative greedy approach to adjust the flipping/unflipping of D-to-Q flow for MBFF instances with the aim of optimizing timing without increasing net congestion.

3. To overcome the space waste problem, we use the empty space for *optimizing timing through upsizing transistors* in MBFFs at the post-route stage. We optimize the upsizing level in a way to resolve the timing violations while minimizing the power overhead. We formulated the sizing problem into an ILP and solve it optimally.

4. We propose a DTCO framework integrating items 1, 2, and 3 to explore and resynthesize MBFF cells in the placement and routing stages that leads to effec-

tively improve routability and resolve timing problem on target circuit implementation while minimizing the power consumption overhead.

In Chapter 3, we propose an optimal non-uniform MBRFF allocation algorithm that are able to overcome the two limitations while taking into account minimizing the two overheads of wakeup latency and control signals [37]. The main contributions of this work are summarized as:

- We propose an optimal non-uniform MBRFF allocation algorithm that can be directly applied to the original flip-flop dependency graph of target circuit, equipped with two options of constraining wakeup latency: $l = 2$ and $l = 3$ clock cycles and three options of minimization objectives: total bits of retention storage, total leakage power on retention storage, and total area.
- We formulate the non-uniform MBRFF allocation problem into a weighted unate covering problem by exhaustively generating valid covering segments together with their cost[3] from the input flip-flop dependency graph and solve it optimally. In addition, we handle the scalability problem for large circuits in experiments.
- We provide a set of comprehensive experimental results to measure how the existing SBRFF method and best-known MBRFF allocation heuristic close to an optimum.

[3]It represents retention bit size, leakage power on retention storage, or retention storage area.

Chapter 2

Enhancing Design Qualities Utilizing Multi-bit Flip-flops: A Design and Technology Co-optimization Driven Approach

2.1 Key Observations and Enabling Optimization Directions

We observe two distinct features on MBFFs whose effective utilization could provide a considerable impact on the improvement of the quality of chip implementation.

1. **Utilizing the full flexibility of D-to-Q flows in MBFFs to save route cost**: A flip-flop basically consists of master and slave latches, and two clock inverters to deliver the inverted and non-inverted clock signals to both latches. The D-to-Q flow of a flip-flop refers to the internal logic path from input D to output Q in the flip-flop. Since D is an input to the master latch and Q is an output from the slave latch, the distance from input port D to output port Q in most of conventional flip-flop cells amounts to the cell width. (For example, in Figs. 2.2(a) and (b), the distance is $0.81um$ = $54nm$ (= CPP) $\times$ 15 gate polys according to ASAP7 PDK [4].)

   For a single-bit flip-flop, it is possible to reduce route cost (e.g., wirelength, routability, DRVs (design rule violations)) by flipping the flip-flop cell instance

in placement/routing stage, as shown in Fig. 2.1(a). On the contrary, for an MBFF, entirely flipping the MBFF cell instance does not always save route cost, as illustrated in Fig. 2.1(b), since its D-to-Q flows are compelled to be reversed all together. However, as shown by the case in Fig. 2.1(c), *a full saving of route cost can be achieved if MBFF cells with all combinations of flipped and non-flipped individual flip-flops are available*. Note that since the individual flip-flops themselves are never tapped by flipping, all such MBFFs will preserve the same timing and power characteristics.

2. **Utilizing the unused space in MBFF footprint to resolve the timing problem**: For n-bit MBFFs ($n > 2$), sharing clock inverters by the individual flip-flops stacked in MBFFs inevitably induce empty space on their footprint, as previously shown in Fig. 1.3(b). To exploit this space effectively, we may include additional gate polys (i.e., upsizing transistors by transistor folding) to this empty space in a way to enhance the timing performance of their associated internal flip-flops. For example, as shown in Figs. 2.2(c) and (d), by including one or more gate polys to each of f_1 and f_4 with 2 or 4 fins to the empty space i.e., upsizing the transistors adjacent to the empty space, the timing of the flip-flops can be improved.

   Specifically, Table 2.1 shows the changes of timing, in terms of the sum of worst slack time and worst clock-to-Q delay, on the individual flip-flops of 4-bit MBFF produced by applying transistor upsizing with 2 fins in inverter U1 in master latch (*upsizing to level-1* in Fig. 2.2(d)), 2 fins in transmission gates M3 and M4 in slave latch (*upsizing to level-2* in Fig. 2.2(d)), and 2 fins in U1 and 2 fins in M3 and M4 (*upsizing to level-3* in Fig. 2.2(d)) to f_1 and f_4 in Fig. 2.2(c). (In our experiments, we synthesized 4-bit MBFFs of various sizing by stacking 1-bit flip-flop cell DFFHQNX1 in ASAP 7nm cell library [4] with clock inverter in the middle and characterized MBFF cells by using Cadence and Synopsys tool chain. Note that a spacing of at least two dummy polys is required between the

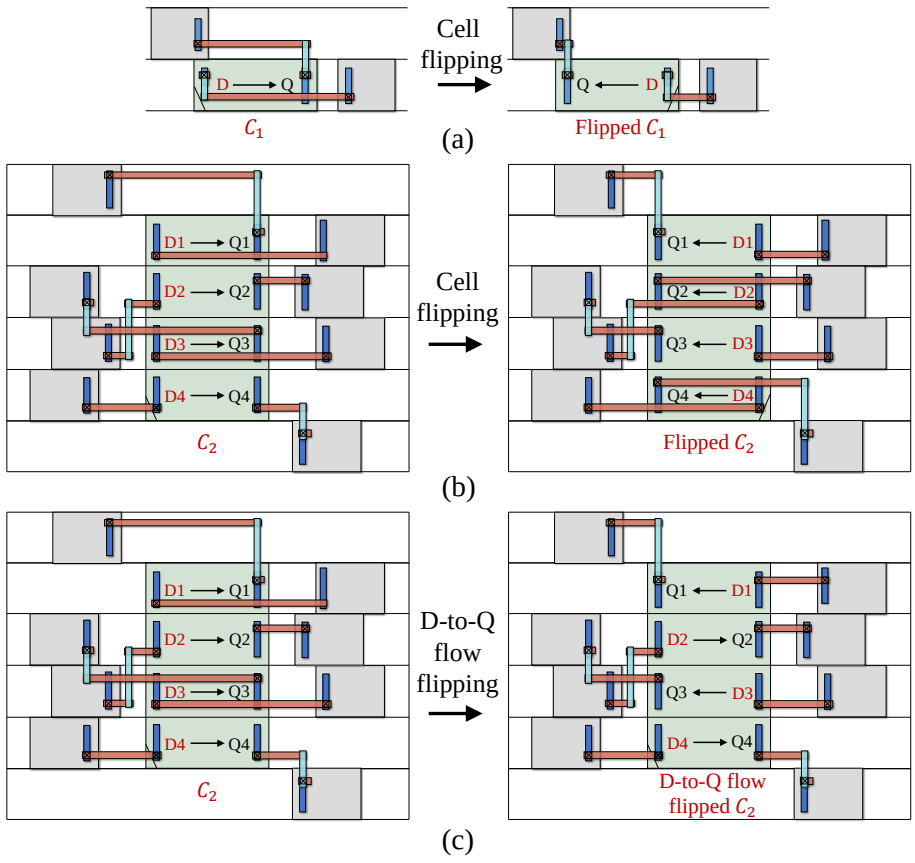


Figure 2.1: Effectiveness of cell flipping on reducing route cost. (a) Flipping a single-bit flip-flop: *Fully effective*. (b) Flipping an MBFF: *Partially or little effective*. (c) Flipping individual D-to-Q flows in MBFF: *Fully effective*.

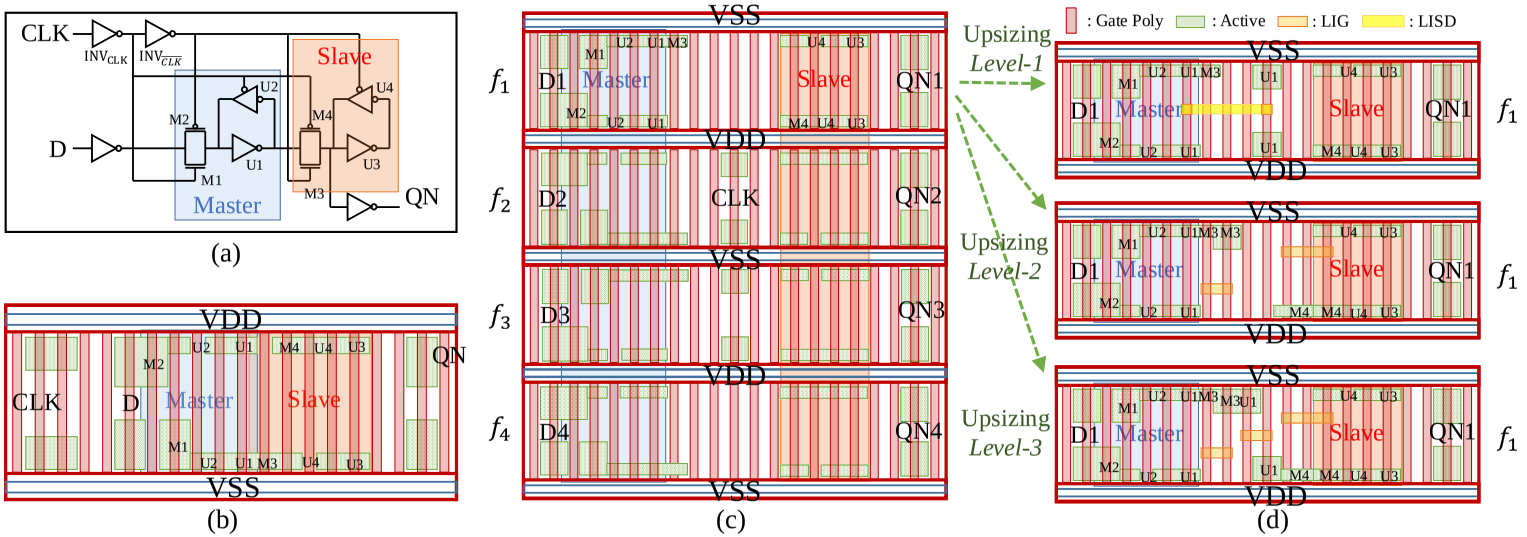


Figure 2.2: Utilization of the empty space induced by the clock inverter sharing in MBFFs. (a) A CMOS circuit of DFFHQNx1 cell in ASAP7nm [4] constructed from two CMOS latches. (b) Layout of DFFHQNx1 cell in ASAP7nm. (c) An MBFF composed of four stacked DFFHQNX1 cells with clock inverters in the middle. (d) Upsizing transistors in f_1 and f_4 by folding on the transistors of U1, M3, or M4 in (a) to the empty space.

source/drain regions of two adjacent transistors in cell layout if they cannot be shared according to ACTIVE design rules in ASAP 7nm PDK.)

We can see from Table 2.1 that it is able to achieve timing improvement of 10.1% (136.1ps $\rightarrow$ 122.2ps) on f_1 and 10.4% (137.9ps $\rightarrow$ 123.5ps) on f_4 in a 4-bit MBFF by upsizing transistors with a few additional fins while retaining the same cell size and having no timing loss on flip-flops f_2 and f_3 at all. Thus, *by building up a cell library with timing-diversified MBFFs through the utilization of empty space, we are able to use the MBFFs selectively in a way to resolve the timing closure problem in chip implementation.* However, it should be noted that the timing benefit counts the cost of power consumed by the inclusion of additional fins. Thus, a careful selection of MBFFs is required to resolve negative timing slacks while minimizing the power overhead.

Table 2.1: Timing (setup time + clock-to-Q delay) on the flip-flops f_1, f_2, f_3, and f_4 in Fig. 2.2 as the transistor upsizing (i.e., folding) to *level-1*, *level-2*, and *level-3* is applied to each of f_1 and f_4 without increasing cell size.

Flip-flop	Unsizing	Transistor upsizing		
		Level-1	*Level-2*	*Level-3*
f_1	136.1ps (1)	**129.1**ps	**130.9**ps	**122.2**ps (0.899)
f_2	137.7ps	138.3ps	135.5ps	136.3ps
f_3	136.5ps	136.8ps	134.0ps	134.5ps
f_4	137.9ps (1)	**130.8**ps	**132.1**ps	**123.5**ps (0.896)

2.2 DTCO Framework for Multi-bit Flip-flops

2.2.1 The Proposed DTCO Flow

Fig. 2.3 shows our proposed DTCO flow, called MBFF-opt, that integrates three new tasks for implementing target designs utilizing MBFF cells: (1) *D-to-Q flow optimization* in Step 1 with the objective of improving routability in the placement stage, (2) *timing driven D-to-Q flow refinement* in Step 2 with the objective of improving routability and timing in the global routing stage, and (3) *timing optimization* in Step 3 with the objective of resolving the timing violations in the post-routing stage. The input of Step 1 is a logic synthesized and placed design produced by using a conventional standard cell library and PDK. (We used ASAP 7nm cells and PDK [4].) We perform Step 1 in two sub-steps: (1.1) *evaluating D-to-Q flow flipping* for MBFF instances and (1.2) *Replacing D-to-Q flow optimized MBFFs* for the MBFFs in (1.1) based on the flipping cost to perform a flow replacement. For a global routing result, we apply Step 2 i.e., (2) *refining D-to-Q flow* of MBFF instances by analyzing timing and congestion data. Meanwhile, the input of Step 3 is the outcome of detailed routing with timing analysis data. We carry out Step 3 in three steps: (3.1) *reordering the flip-flop stacking position* in the MBFF instances with timing violation by assessing timing criticality, (3.2) *upsizing transistors* on the MBFFs in (3.1), and (3.3) *elaborating timing* in case there still exists a negative timing slack. The following three subsections describe the details of Steps 1, 2, and 3.

2.2.2 D-to-Q Flow Optimization

Step 1.1 (Evaluating D-to-Q flow flipping): We refer *D-to-Q flow flipping* of an internal flip-flop f_i, $1 \leq i \leq K$, in a K-bit MBFF to as *reversing the D-to-Q flow* i.e., the Q-to-D flow in the MBFF, which is exactly equivalent to the D-to-Q flow in the flipped flip-flop in a new MBFF produced by flipping the f_i in the initial MBFF. For a K-bit MBFF, since we have two options to choose D-to-Q flow for each internal

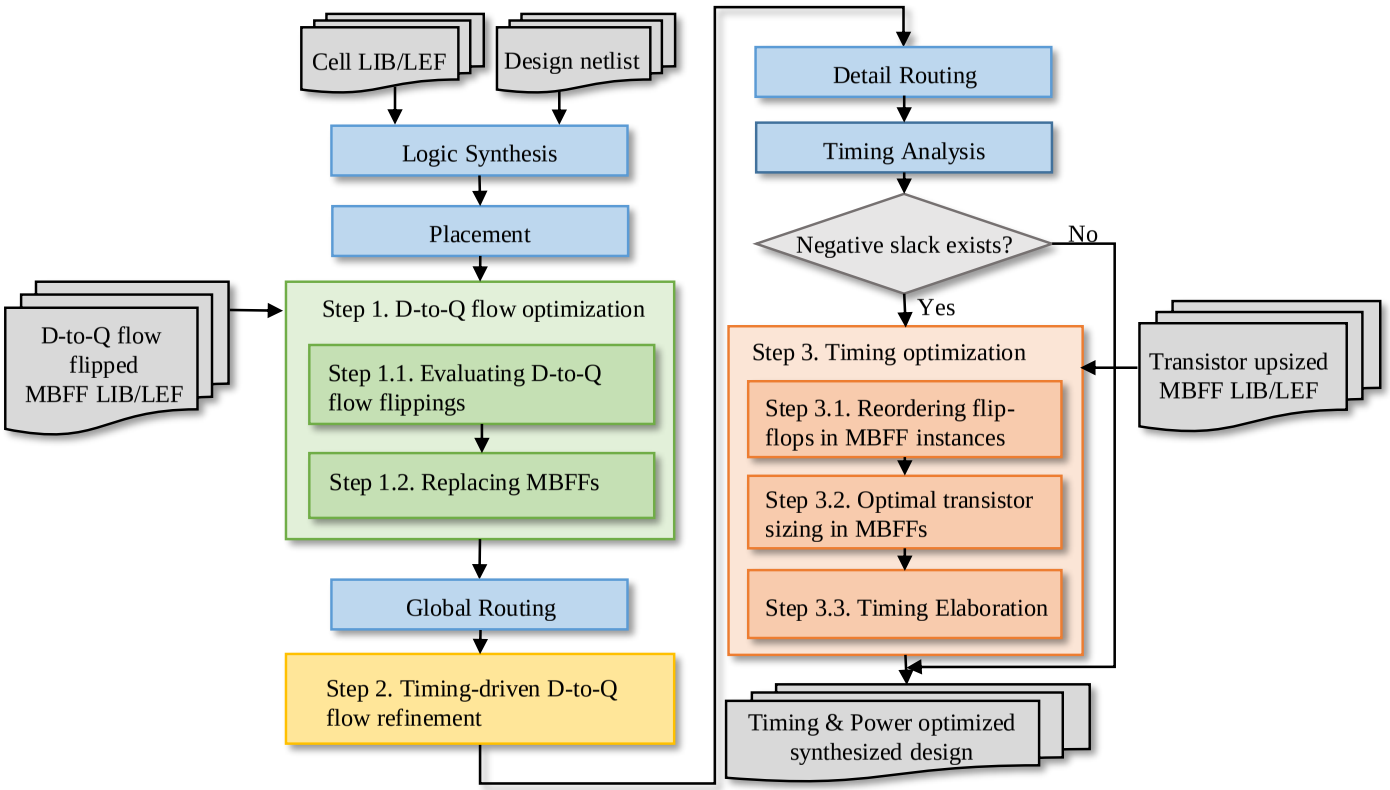


Figure 2.3: Our proposed DTCO flow MBFF-opt for synthesizing and utilizing MBFFs that are most suitable for target designs. The objective in Step 1 is to improve chip routability and Step 2 is to improve timing in global routing stage as well as chip routability while the objective in Step 3 is to resolve timing violations in association with MBFF timing.

flip-flop namely, unflipped D-to-Q flow and flipped D-to-Q flow, there are total of 2^K MBFF cells with different D-to-Q flow configurations.

For every MBFF instance in placement, we want to find the MBFF of D-to-Q flow configuration which has the least total amount of routability cost to connect its $2K$ D and Q ports. Let t_i^D and t_i^Q be the cell terminals that should be connected to ports D_i and Q_i on flip-flop f_i in an MBFF. Then, we compute the quantity of the following equation for each internal flip-flop with *isFlip* = 0 and 1:

$$C(f_i, isFlip_i) = C_{WL}(D_i, t_i^D, isFlip_i) + C_{WL}(Q_i, t_i^Q, isFlip_i) \tag{2.1}$$

where $C_{WL}(D_i, t_i^D, isFlip)$ indicates the cost[1] of physically connecting D_i and t_i^D when D-to-Q flow flipping is unapplied (i.e., $isFlip_i$ = 0) or applied (i.e., $isFlip_i$ = 1) to f_i. Likewise, $C_{WL}(Q_i, t_i^Q, isFlip)$ is defined similarly.

Then, we find a K-dimensional 0/1 vector $\Gamma = [isFlip_1, \cdots, isFlip_K]$ that minimizes the quantity of C_{tot}:

$$C_{tot} = \sum_{i=1}^{K} min\{C(f_i, isFlip), isFlip = 0, 1\} \tag{2.2}$$

We call the *isFlip* vector, Γ, obtained by *Eq.2.2* the *maximal profit vector* of the MBFF instance among all possible 2^K K-dimensional 0/1 vectors.

Fast computation of the maximal profit vector: Since we are interested in determining a better option between D-to-Q flow unflipping and flipping for each internal flip-flop, rather than fully computing the equations in *Eq.2.1* and *Eq.2.2*, we can quickly setup the 0/1 values in the maximal profit vector Γ by merely comparing the interval between D_i and Q_i in x-coordinate with the interval between t_i^D and t_i^Q in x-coordinate:

[1] In this work, we assume the cost is the HPWL value of the bounding box of D_i and t_i^D.

case 1. The interval between t_i^D and t_i^Q contains the interval between D_i and Q_i inclusive.

case 2. The interval between D_i and Q_i contains the interval between t_i^D and t_i^Q inclusive.

case 3. The interval between t_i^D and t_i^Q contains exactly one of D_i and Q_i inclusive.

case 4. The two intervals between D_i and Q_i and between t_i^D and t_i^Q do not overlap.

For an MBFF instance, say M, in placement, we can identify the *case* to which each internal flip-flop f_i belongs and set the corresponding bit $isFlip_i$ to 0 or 1 to produce maximal profit vector Γ_M.

The binary element setting procedure is shown in Algorithm. 1. For *case 1*, we can easily set $isFlip_i$ by calling **isRevDir()**, which distinguishes the two states in Fig. 2.4(a) i.e., **isRevDir()** == TRUE (line 4-7) indicating the upper state, thus setting $isFlip_i = 1$ and otherwise, setting $isFlip_i = 0$. Similarly, we can set $isFlip_i$ for *case 3*. For *case 2*, in line 8-11, we can determine $isFlip_i$ by calculating 2($h_{D_i} + h_{Q_i} - L$) because the wirelength profit $C(f_i, 1) = h_{D_i} + h_{Q_i} - h_{fD_i} - h_{fQ_i}$ becomes 2($h_{D_i} + h_{Q_i} - L$) as $h_{fD_i} = L - h_{D_i}$ and $h_{fQ_i} = L - h_{Q_i}$, as shown in Fig. 2.4(b). For *case 4*, $isFlip_i$ can be set to either 0 or 1 since there is no change in total wirelength (line 16-18).

Step 1.2 (Replacing MBFFs): From the set of Γ configurations obtained from Step 1.1, we utilize an MBFF cell synthesized corresponding to each Γ configuration. The MBFF synthesis is simply stacking individual flip-flops with either unflipping or flipping, obeying their $isFlip$ 0/1 values in Γ. Let $\mathcal{L}$ be the set of MBFF cells produced by this step. Then, we replace every MBFF instance in placement whose original *isFlip*

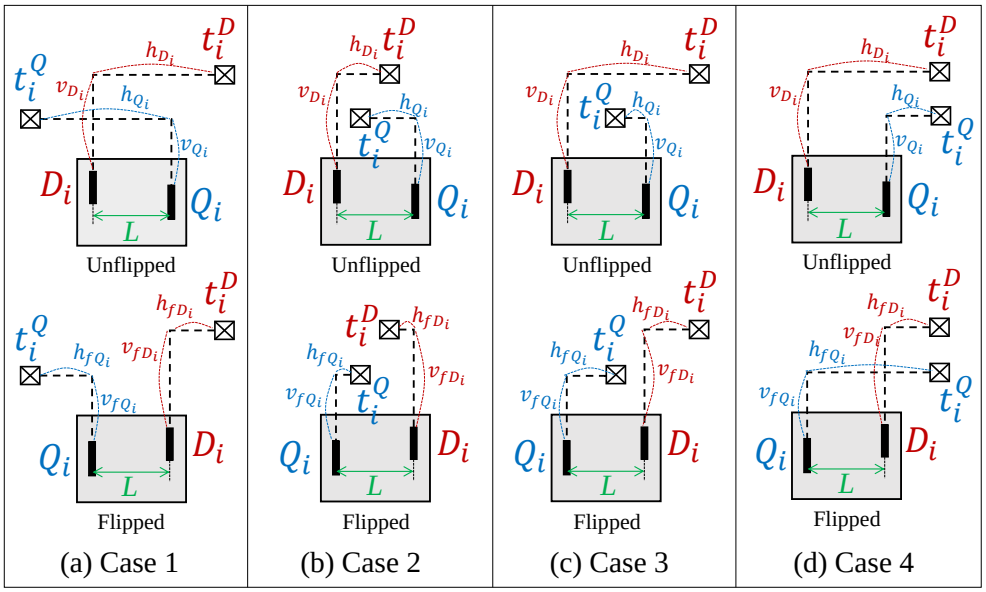


Figure 2.4: Analysis of x-coordinate relation between D_i, Q_i, t_i^D, and t_i^Q where $\mathcal{I}(A, B)$ represents the interval between A and B in x-coordinate and $x(A)$ denotes the x-coordinate of A. (a) $\mathcal{I}(D_i, Q_i) \subset \mathcal{I}(t_i^D, t_i^Q)$. (b) $\mathcal{I}(t_i^D, t_i^Q) \subset \mathcal{I}(D_i, Q_i)$. (c) $\{x(t_i^D) \in \mathcal{I}(D_i, Q_i),\ x(t_i^Q) \notin \mathcal{I}(D_i, Q_i)\}$ or $\{x(t_i^D) \notin \mathcal{I}(D_i, Q_i),\ x(t_i^Q) \in \mathcal{I}(D_i, Q_i)\}$. (d) $\mathcal{I}(D_i, Q_i) \cap \mathcal{I}(t_i^D, t_i^Q) = \phi$.

Algorithm 1 Computing the maximal profit vector of every MBFF instance M

Input: $x(D_i), x(Q_i), x(t_i^D), x(t_i^Q)$ of flip-flop f_i in M

Output: Hashtable $\mathcal{H}$ (key, value) = (Γ, list of MBFF instances)

▷ $L = |x(D_i) - x(Q_i)|$

▷ $h_{D_i} = |x(D_i) - x(t_i^D)|$

▷ $h_{Q_i} = |x(Q_i) - x(t_i^Q)|$

for all M in MBFF instances **do**
  for all i in K flip-flops on M **do**
    $isFlip_i \leftarrow 0$
    if $\mathcal{I}(D_i, Q_i) \subset \mathcal{I}(t_i^D, t_i^Q)$ **then** ▷ *case 1*
      if isRevDir($x(D_i)$, $x(Q_i)$, $x(t_i^D)$, $x(t_i^Q)$) **then**
        $isFlip_i \leftarrow 1$
      end if
    else if $\mathcal{I}(t_i^D, t_i^Q) \subset \mathcal{I}(D_i, Q_i)$ **then** ▷ *case 2*
      if $2(h_{D_i} + h_{Q_i} - L) > 0$ **then**
        $isFlip_i \leftarrow 1$
      end if
    else if $\{x(t_i^D) \in \mathcal{I}(D_i, Q_i), x(t_i^Q) \notin \mathcal{I}(D_i, Q_i)\}$ or $\{x(t_i^D) \notin \mathcal{I}(D_i, Q_i), x(t_i^Q) \in \mathcal{I}(D_i, Q_i)\}$ **then** ▷ *case 3*
      if isRevDir($x(D_i)$, $x(Q_i)$, $x(t_i^D)$, $x(t_i^Q)$) **then**
        $isFlip_i \leftarrow 1$
      end if
    else if $\mathcal{I}(D_i, Q_i) \cap \mathcal{I}(t_i^D, t_i^Q) = \phi$ **then** ▷ *case 4*
      $isFlip_i \leftarrow 0$
    end if
    $\Gamma_M[i] \leftarrow isFlip_i$
  end for
  $\mathcal{H}[\Gamma_M].insert(M)$
end for
Return $\mathcal{H}$

Function - **isRevDir** $(x(D_i), x(Q_i), x(t_i^D), x(t_i^Q))$

1: **if** $(x(D_i) < x(Q_i)$ and $x(t_i^D) > x(t_i^Q))$ or $(x(D_i) > x(Q_i)$ and $x(t_i^D) < x(t_i^Q))$ **then**
2: Return TRUE
3: **else**
4: Return FALSE
5: **end if**

Algorithm 2 Synthesizing and replacing MBFFs

Input: Hashtable $\mathcal{H}$ (key, value) = (Γ, list of MBFF instances)

▷ c_M : MBFF cell with Γ_M flow

1: $\mathcal{L} \leftarrow \phi$, $\mathcal{R} \leftarrow \phi$
2: **for all** key Γ_M in $\mathcal{H}$ **do**
3: $\mathcal{L} = \mathcal{L} \cup c_M$
4: **for all** value M of $\mathcal{H}[\Gamma_M]$ **do**
5: $\mathcal{R} = \mathcal{R} \cup (M, c_M)$
6: **end for**
7: **end for**
8: Replacing MBFF instances according to $\mathcal{R}$ by using $\mathcal{L}$

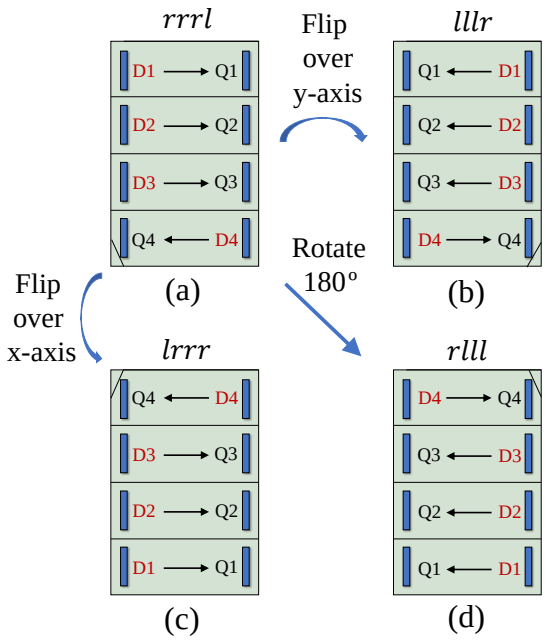
vector is not matched with its maximum profit vector Γ by the MBFF cell in $\mathcal{L}$ with Γ. This step is described in Algorithm using the hashtable $\mathcal{H}$ produced in Step 1.1 as input.

In fact, the MBFF cells that cover all kinds of D-to-Q flow combinations (2^K) entail lots of redundancy. For example, for $K = 4$, Fig. 2.5(a) (d) shows that MBFF cells corresponding to four configurations of D-to-Q flows can be obtained by flipping or rotating a single MBFF cell. Assume that for a flip-flop, r denotes its D-to-Q flow such that D is on the left side and Q is on the right side of the flip-flop, and l denotes the reverse of D-to-Q flow such that D is on the right side and Q is on the left side. For example, the sequence of D-to-Q flow of the MBFF in Fig. 2.5(a) can be expressed as $rrrl$. By using an MBFF cell with $rrrl$, we can represent flow sequence $lllr$ by flipping it over the y-axis as shown in Fig. 2.5(b). We can also represent flow sequence $lrrr$ by flipping the MBFF over x-axis and reversing the order of pins i.e., $\{D1, D2, D3, D4\} \rightarrow \{D4, D3, D2, D1\}$ and $\{Q1, Q2, Q3, Q4\} \rightarrow \{Q4, Q3, Q2, Q1\}$. In addition, we can represent flow sequence $rlll$ by rotating the MBFF in 180 degree and reversing the order of pins. Fig. 2.5(e) shows only 6 flow sequences for 4-bit MBFF, covering all of 16 D-to-Q flow combinations. The minimal number of flow sequences to cover all 2^K flow combinations, N_{MBFF}, is computed by

$$N_{MBFF} = 1 + \sum_{n=1}^{\frac{K}{2}} \frac{\binom{K}{n} - m_n}{2} + m'_n = 1 + \sum_{n=1}^{\frac{K}{2}} \frac{\binom{K}{n} + m_n}{2} \tag{2.3}$$

$$\text{where } m_n = \begin{cases} \binom{\frac{K}{2}}{\frac{n}{2}} & \text{when } n = \text{even number and } 2n \neq K \\ 0 & \text{otherwise.} \end{cases}$$

$$\text{where } m'_n = \begin{cases} \frac{m_n}{2} & \text{when } n = \text{even number and } 2n = K \\ m_n & \text{otherwise.} \end{cases}$$



Flow sequence	Coverable flow sequence
rrrr	*rrrr, llll*
rrll	*rrll, llrr*
rllr	*rllr, lrrl*
lrlr	*lrlr, rlrl*
rrrl	*rrrl, lllr, lrrr, rlll*
rrlr	*rrlr, rlrr, lrll, llrl*

(e)

Figure 2.5: Illustration of covering every D-to-Q flow combination of 4-bit MBFF by using 6 cells.

2.2.3 Timing-driven D-to-Q Flow Refinement

In the prior step, we conducted D-to-Q flow flipping with the goal of minimizing wirelength cost between cells at the post-placement. In this step, we pay attention to timing with timing-driven cost formulation and proceed D-to-Q flow flipping on the global routing result, performing the following two sub-steps.

Step 2.1 (Evaluating D-to-Q flow flippings): From a global routing result, we extract a timing report related to MBFF instances. By extracting timing report on every path between individual flip-flop pins (D, Q) of MBFF instances, we compute flipping cost, p_i, of flip-flop f_i, $1 \leq i \leq K$:

$$p_i = n_{WL}^{-} - n_{WL}^{+} \tag{2.4}$$

in which n_{WL}^{-} represents the number of fan-in or fan-out pins with negative slacks that would decrease the wirelength if D-to-Q flow of f_i were flipped. Conversely, n_{WL}^{+} represents the number of fan-in or fan-out pins that would increase the wirelength when D-to-Q flow of f_i were flipped. The reduced wirelength of each of D and Q pins according to fan-in pin t_i^D and fan-out pins $t_i^{Q_j}$, $1 \leq j$ can be calculated by identifying the *case* (Fig. 2.4) the flip-flop belongs to and using Table 2.2 that represents the wirelength reduction for the cases to be identifed with **isRevDir()** in Algorithm. 1. We flip f_i when p_i is positive, which means the possibility of improving negative slack is higher than that of worsening the negative slack if its D-to-Q flow were flipped.

Step 2.2 (Replacing MBFFs): When D-to-Q flow flipping is to be performed towards improving negative timing slack, but is very likely to make net detours due to congestion, resulting timing may deteriorate. Thus, MBFF cell replacement should consider congestion. A naive approach is to make D-to-Q flow flipping for each of the MBFF instances identified in Step 2.1 one by one, checking if congestion increases and keeping only safe ones. However, this method requires a long run time. Instead, we propose a method of selecting a subset of MBFF instances from the ones obtained Step 2.1 based on the following replacement profit cost, P_{MBFF}, of each MBFF instance, and

Table 2.2: Wirelength reduction between D_i and t_i^D (ΔWL_D), and Q_i and $t_i^{Q_j}$ (ΔWL_{Q_j}) for all cases in Fig. 2.4. (**isRevDir()** is described in Algorithm. 1.)

	ΔWL_D		ΔWL_{Q_j}	
	isRevDir()	**!isRevDir()**	**isRevDir()**	**!isRevDir()**
Case 1	L	$-L$	L	$-L$
Case 2	$2h_D - L$	$L - 2h_D$	$2h_Q - L$	$L - 2h_Q$
Case 3	L	-L	$2h_Q - L$	$L - 2h_Q$
Case 4	L	-L	-L	L

replacing them with the corresponding flipped cells all at once:

$$P_{MBFF} = \sum_{i=1}^{K} max(0, p_i) + \frac{1}{L} \sum_{i=1}^{K} \Delta WL_i \tag{2.5}$$

where the first term corresponds to the sum of D-to-Q flow flipping/non-flipping costs of the internal flip-flops in the MBFF and the second term indicates the sum of HPWL changes, denoted by ΔWL_i, by the flipping/non-flipping of the internal flip-flops, normalized by L, which is the D-to-Q distance in the MBFF cell.

Precisely, ΔWL_i for an internal flip-flop f_i is computed by

$$\Delta WL_i = max(0, \Delta WL_i^D) + \sum_{j=1}^{|FO|} max(0, \Delta WL_i^{Q_j}) \tag{2.6}$$

$$\Delta WL_i^D = \begin{cases} 0 & \text{if } slack_D \geq 0 \\ \Delta WL_D & \text{if } slack_D < 0 \end{cases} \tag{2.7}$$

$$\Delta WL_i^{Q_j} = \begin{cases} 0 & \text{if } slack_{Q_j} \geq 0 \\ \Delta WL_{Q_j} & \text{if } slack_{Q_j} < 0 \end{cases} \tag{2.8}$$

in which FO represents the set of fan-out pins driven by pin Q in f_i, and $slack_D$ and $slack_{Q_j}$ indicate the timing slack from fan-in pin t_i^D to D_i and from Q_i through

fan-out pin $t_i^{Q_j}$, respectively. (Note that ΔWL_D and ΔWL_{Q_j} are computed by the formulation in Table 2.2.

We convert the problem of selecting MBFF instances maximizing P_{MBFF} into an instance of maximal weighted independent set (MWIS) problem in a directed graph $G(V, A)$ such that a node in V indicates a distinct MBFF instance in circuit and an arc from node n_i to node n_j exists in A if there is a timing path from a flip-flop in the MBFF instance of n_i to a flop-flop in the MBFF instance of n_j. Every vertex v_i in V has a weight w_i that represents the replacement cost of the corresponding MBFF instance. We solve the MWIS problem by formulating it into an ILP (integer linear programming):

$$\begin{aligned} \text{Maximize:} \quad & \sum_{i=1}^{|V|} w_i \cdot x_i \\ \text{subject to:} \quad & x_i + x_j \leq 1, \text{ if } \{i, j\} \in A \\ & x_i \in \{0, 1\} \end{aligned}$$

where x_i = 1 indicates that its MBFF instance is selected for MBFF replacement.

We iteratively perform the selection of MBFF instances by finding the independent set of maximal cost followed by replacing them with the MBFFs of minimal P_{MBFF} value. Let M_I be the set of MBFF instances selected in the current iteration. Then, by analyzing the global routing result of the circuit with the MBFF replacement in M_I, we count the number of congestion overflows and timing violated paths. If the number of timing violated paths decreases (i.e., $\Delta TVP < 0$) and the number of congestion overflows does not increase (i.e., $\Delta OF \leq 0$), we accept the replacement of M_I, update $G(V, A)$ by removing nodes in M_I, and repeat the iteration process. Otherwise, we insert the MBFF instances in M_I into queue Q_m in non-increasing order of replacement profit P_{MBFF}. Then, we iteratively pop an element from Q_m and replace it with the corresponding MBFF cell as long as the above condition is satisfied. The iteration process repeats from the circuit with partially replaced MBFF instances. Fig. 2.6 show the flow of MBFF replacement process.

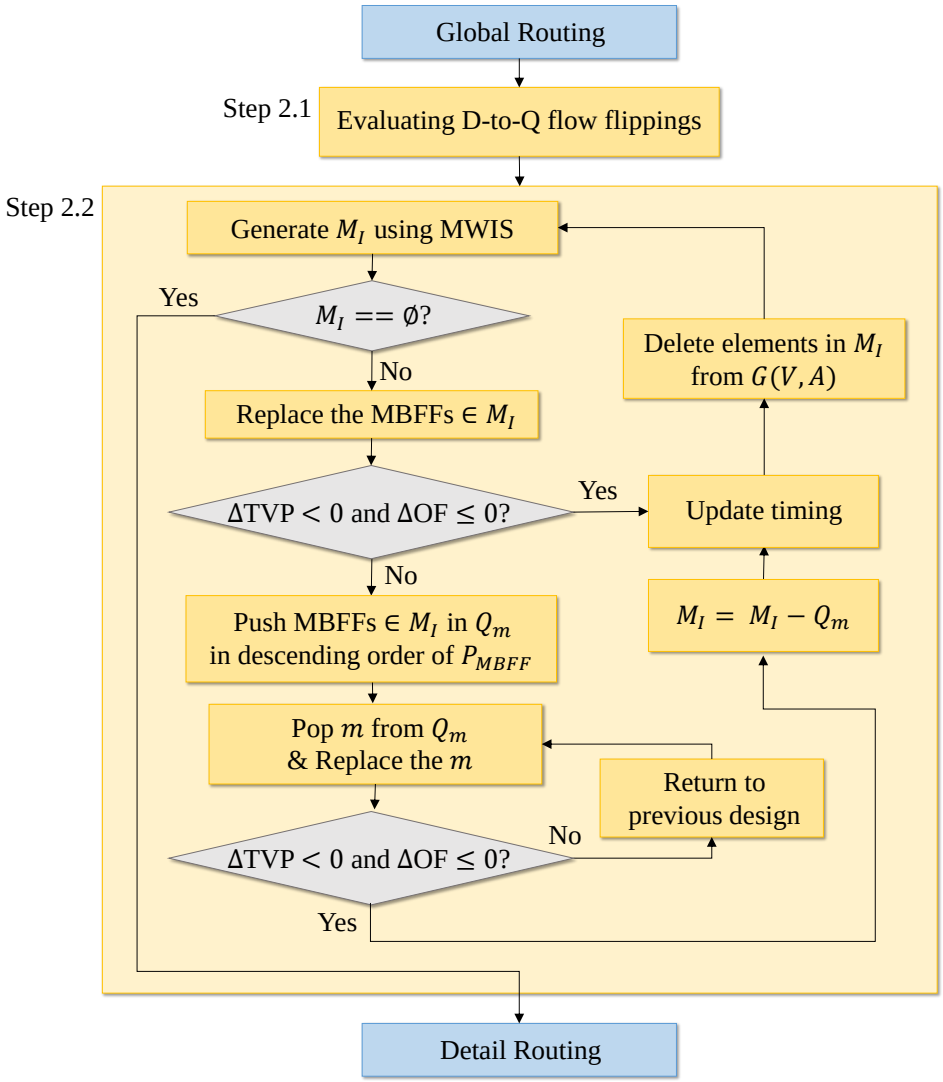


Figure 2.6: Proposed flow of timing-driven D-to-Q flow refinement by MBFF replacement.

2.2.4 Timing Optimization at Post-Route Stage

Step 3.1 (Reordering flip-flops in MBFF instances): This step performs two independent sub-tasks that will be used in Step 3.2:

1. (*Finding an optimal flip-flop reordering*) For each MBFF instance in routing, we find a rearrangement (i.e., mapping ϕ) of the flip-flops in the instance that maximizes the mapping cost $C(\phi)$:

$$C(\phi) = \sum_{i=1}^{K} \bigtriangledown(\phi(i)) \tag{2.9}$$

where $\bigtriangledown(\phi(i))$ indicates an estimation of timing saving by moving flip-flop f_i to the location of flip-flop index $\phi(i)$ and is computed by

$$\bigtriangledown(\phi(i)) = \alpha_1 \cdot y_D(\phi(i)) + \alpha_2 \cdot y_Q(\phi(i)) \tag{2.10}$$

where $y_D(\phi(i))$ and $y_Q(\phi(i))$ represent the distance saving in y-direction by moving f_i to the location of flip-flop index $\phi(i)$, respectively, and α_1 = 1 if no timing violation occurs on port D on f_i or $\phi(i) \neq 1$ and K. (Note that we want to use the empty space in the top and bottom flip-flops (i.e., f_1 and f_K) for transistor upsizing.) Otherwise, α_1 is set to 5 (or a large number greater than 0). α_2 is similarly set to that of α_1.

Fig. 2.7(b) shows an example of computing the values of $\bigtriangledown(\phi(\cdot))$ for mapping ϕ to the flip-flops in an MBFF instance in Fig. 2.7(a). (We set $|y_D(i) - y_D(j)|$ = $|y_Q(i) - y_Q(j)|$ = $|i - j|$, $i, j \in \{1, 2, \cdots, K\}$.)

We formulate the problem of finding an optimal mapping ϕ for an MBFF instance into a problem instance of weighted maximal matching on a complete bipartite graph $G(V1, V2, W)$, in which $V1$ is the set of flip-flops in the instance and $V2$ is the set of the flip-flop bin indices, and the edge between $v_i \in V1$ and $v_j \in V2$ is assigned with a weight $w(v_i, v_j) \in W$ such that $w(v_i, v_j) = \bigtriangledown(\phi(i) = j)$. For example, Fig. 2.7(c) shows $G(V1, V2, W)$ of the

MBFF instance in Fig. 2.7(a), in which the heavy lines indicate the solution of maximal matching, producing the maximal total saving, which is $C(\phi)$ = 6 and generates reordered MBFF instance in Fig. 2.7(d). We formulate the maximal matching problem into an ILP:

$$\begin{aligned}
\text{Maximize:} \quad & \sum_{i=1}^{K}\sum_{j=1}^{K} w(v_i, v_j) \cdot x_{i,j} \\
\text{subject to:} \quad & \sum_{j=1}^{K} x_{i,j} = 1, \text{ for all } i = 1, \cdots, K \\
& \sum_{i=1}^{K} x_{i,j} = 1, \text{ for all } j = 1, \cdots, K \\
& x_i^j \in \{0, 1\}
\end{aligned}$$

where the 0/1 variable $x_{i,j}$ becomes 1 if flip-flop f_i is assigned to bin index j and 0, otherwise.

2. (*Generating MBFF cell library by an exhaustive exploration of transistor upsizing*) For a K-bit MBFF, we use the empty space in the top flip-flop (i.e., f_1) and in the bottom flip-flop (i.e., f_K) for transistor upsizing to $level$-1, $level$-2, and $level$-3 as well as unsizing, as illustrated in Figs. 2.2(c) and (d). Thus, we prepare a K-bit MBFF cell library, $\mathcal{L}'_{MB}$, of 16 (= 4^2) MBFF cells, for K = 4, for each synthesized MBFF cell in the previous steps. Then, we characterize the cells in $\mathcal{L}'_{MB}$ to extract timing and power numbers.

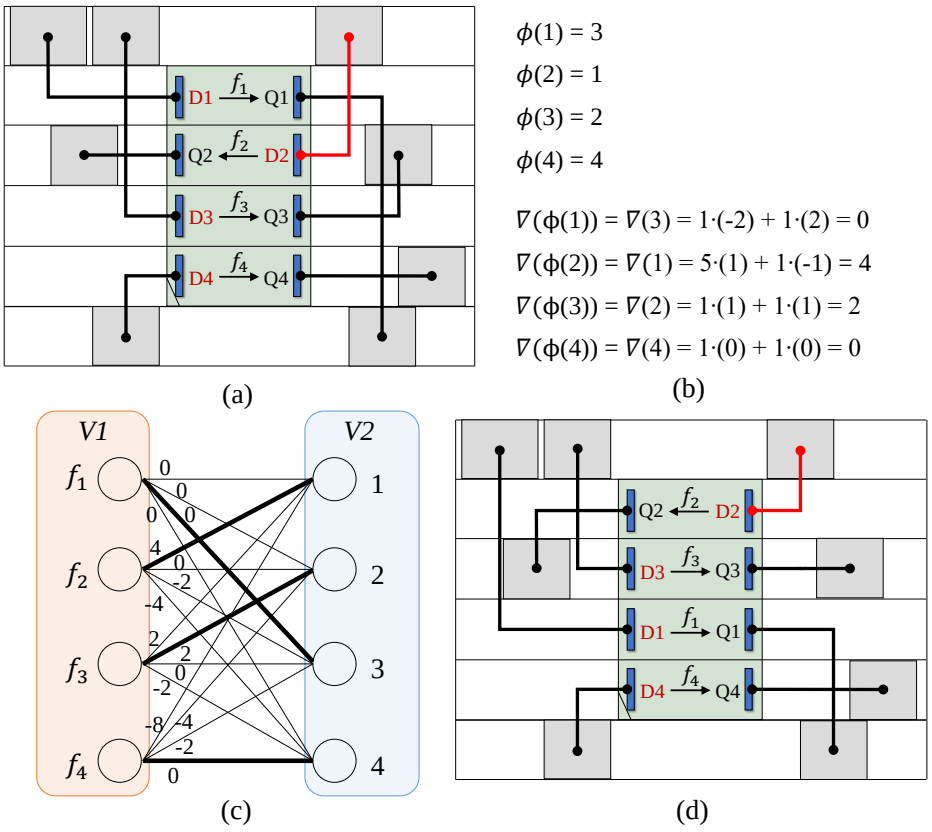


Figure 2.7: Example of finding an optimal flip-flop reordering. (a) An MBFF instance with timing violation on the route path (red color) to pin D2. (b) Mapping function ϕ and computation of the values of $\bigtriangledown(\phi(\cdot))$. (c) Bipartite graph $G(V1, V2, W)$ of the MBFF instance in (a). (d) Reordered MBFF instance according to the solution of maximal matching in (c).

Step 3.2 (Optimal Transistor Sizing in MBFFs): Once the flip-flop rearrangement function is obtained in Step 3.1 for every MBFF instance, we find an optimal solution of MBFF transistor upsizing in the target circuit to resolve all timing violations while minimizing the power overhead. We formulate the problem into an ILP with the preparation of the following two sets of timing data.

1. By characterizing the transistor unsized and upsized MBFF cells in $\mathcal{L}'_{MB}$, for each flip-flop in an MBFF in $\mathcal{L}'_{MB}$, we take (*list* 1) a list of setup time values corresponding to the combinations of a number of transition times (slew rates) at pin D and a number of transition times at pin CLK and take (*list* 2) a list of clock-to-Q delay values corresponding to the combinations of a number of the out-loads at pin Q and a number of transition times at pin CLK. Then, to make a power optimal replacement of all MBFF instances of timing violation in the circuit with MBFF cells in $\mathcal{L}'_{MB}$, an accurate timing calculation related to the MBFF replacement in circuit is essential. To this end, we apply a linear interpolation or extrapolation to the setup time values in *list* 1 to produce a setup time function, $F_{Setup}(tr_D, tr_{CLK})$ as well as to the clock-to-Q delay values in *list* 2 to produce a clock-to-Q delay function, $F_{C2Q}(ld_Q, tr_{CLK})$ where tr_D, tr_{CLK}, and ld_Q indicate the transition time on D, transition time on CLK, and out-load on Q, respectively.

2. We extract information regarding the worst slack timing paths in circuit from the post-route timing report, which includes timing slack, setup time, clock-to-Q delay, required time, arrival time, input pin transition times, and out-load in association with all MBFF instances in the target circuit.

Table 2.3: Notations used in our ILP formulation

	Description
$\mathcal{L}'_{MB}$	Set of MBFF cells with tr. unsizing/sizing produced by step 2.1
$\mathcal{M}$	Set of the MBFF instances in the target circuit
id_M	Index of an MBFF instance M in $\mathcal{M}$
P_i	Set of the timing paths on $M_i \in \mathcal{M}$ ($\|P_i\| = 2K$ if M_i is a K-bit MBFF instance since pins D and Q each has one timing path)
$p_{i,k}$	$p_{i,k} \in P_i$, $k = 1, \cdots, \|P_i\|$
PWR_j	Power consumed by $M'_j \in \mathcal{L}'_{MB}$
x_i^j	1 if $M_i \in \mathcal{M}$ is replaced by $M'_j \in \mathcal{L}'_{MB}$; 0, otherwise.
$req_{i,k}$	Required time on path $p_{i,k}$ before replacement
$arr_{i,k}$	Arrival time on path $p_{i,k}$ before replacement
$init\_setup_{i,k}$	Setup time on pin D of path $p_{i,k}$ before replacement
$init\_clk2q_{i,k}$	Clock-to-Q delay on Q of path $p_{i,k}$ before replacement
$new\_setup_{i,k}$	Setup time on pin D of path $p_{i,k}$ (It will be determined by solving ILP)
$new\_clk2q_{i,k}$	Clock-to-Q delay on pin Q of path $p_{i,k}$ (It will be determined by solving ILP)
$setup_{i,k}^j$	Setup time on pin D of path $p_{i,k}$ when $M_i \in \mathcal{M}$ is replaced with $M'_j \in \mathcal{L}'_{MB}$
$clk2q_{i,k}^j$	Clock-to-Q delay on pin Q of path $p_{i,k}$ when $M_i \in \mathcal{M}$ is replaced with $M'_j \in \mathcal{L}'_{MB}$

Let $Setup_{a,b}$ denote the setup time when the transition time on D is tr_{D_a} and the clock-to-Q delay is tr_{CLK_b} that are specified in *list 1*. Suppose tr_D and tr_{CLK} are not specified in *list 1*, but The tr_D value is in between tr_{D_i} and tr_{D_j} ($tr_{D_i} < tr_{D_j}$) and the tr_{CLK} value is in between tr_{CLK_p} and tr_{CLK_q} ($tr_{CLK_p} < tr_{CLK_q}$). Then, setup time function $F_{Setup}(tr_D, tr_{CLK})$ can be expressed as that in *Eq.*2.11, in which variables A, B, C, and D can be computed by *Eqs.*2.12, 2.13, 2.14, and 2.15. Likewise, $F_{C2Q}(ld_Q, tr_{CLK})$ can be expressed similarly.

$$F_{Setup}(tr_D, tr_{CLK}) = A \cdot tr_D + B \cdot tr_{CLK} + C \cdot tr_D \cdot tr_{CLK} + D \tag{2.11}$$

$$A \cdot tr_{D_i} + B \cdot tr_{CLK_p} + C \cdot tr_{D_i} \cdot tr_{CLK_p} + D = Setup_{i,p} \tag{2.12}$$

$$A \cdot tr_{D_i} + B \cdot tr_{CLK_q} + C \cdot tr_{D_i} \cdot tr_{CLK_q} + D = Setup_{i,q} \tag{2.13}$$

$$A \cdot tr_{D_j} + B \cdot tr_{CLK_p} + C \cdot tr_{D_j} \cdot tr_{CLK_p} + D = Setup_{j,p} \tag{2.14}$$

$$A \cdot tr_{D_j} + B \cdot tr_{CLK_q} + C \cdot tr_{D_j} \cdot tr_{CLK_q} + D = Setup_{j,q} \tag{2.15}$$

The variables (x_i^j, $new\_setup_{i,k}$, $new\_clk2q_{i,k}$) and constant (all the rest) notations used for our ILP formulation of MBFF replacement are summarized in Table 2.3. Then, we can express the objective and all constraints of the ILP formulation as:

$$\text{Minimize:} \quad \sum_{i=1}^{|\mathcal{M}|} \sum_{j=1}^{|\mathcal{L}'_{MB}|} PWR_j \cdot x_i^j \tag{2.16}$$

subject to:

for all $i = 1, \cdots, |\mathcal{M}|,\ j = 1, \cdots, |\mathcal{L}'_{MB}|,\ k = 1, \cdots, |P_i|,$

$$(req_{i,k} + init\_setup_{i,k} - new\_setup_{i,k}) - (arr_{i,k} - init\_clk2q_{i,k} + new\_clk2q_{i,k}) \geq 0 \tag{2.17}$$

$$new\_setup_{i,k} = \begin{cases} init\_setup_{i,k} & \text{if } id1_{i,k} = -1 \\ setup_{i,k}^j & \text{if } x_{id1_{i,k}}^j = 1 \end{cases} \tag{2.18}$$

$$new\_clk2q_{i,k} = \begin{cases} init\_clk2q_{i,k} & \text{if } id0_{i,k} = -1 \\ clk2q_{i,k}^j & \text{if } x_{id0_{i,k}}^j = 1 \end{cases} \tag{2.19}$$

$$id0_{i,k} = \begin{cases} id_M & \text{if the launch register, M, on } p_{i,k} \text{ is in } \mathcal{M} \\ -1 & \text{otherwise} \end{cases} \tag{2.20}$$

$$id1_{i,k} = \begin{cases} id_M & \text{if the capture register, M, on } p_{i,k} \text{ is in } \mathcal{M} \\ -1 & \text{otherwise} \end{cases} \tag{2.21}$$

$$\sum_{j=1}^{|\mathcal{L}'_{MB}|} x_i^j = 1, \quad \text{for all } i = 1, \cdots, |\mathcal{M}| \tag{2.22}$$

$$x_i^j \in \{0, 1\}, \quad \text{for all } i = 1, \cdots, |\mathcal{M}|,\ j = 1, \cdots, |\mathcal{L}'_{MB}| \tag{2.23}$$

The objective shown in *Eq.*2.16 is to minimize the total amount of power consumed by all MBFF instances in the target circuit. *Eq.*2.17 ensures the satisfaction of setup timing constraints of all paths on the internal flip-flops in the MBFFs in target design[2] Precisely, according to the setup time satisfaction condition:

$$Required\_time - Arrival\_time \geq 0$$

where *Required_time* amounts to CLK capture arrival time plus T_{CLK} minus setup time while *Arrival_time* equals to CLK launch arrival time plus combinational delay + clk-to-Q delay.

In addition, *Eq.*2.18 and *Eq.*2.19 explore the MBFF cells in $\mathcal{L}'_{MB}$ to find an MBFF cell consuming the least power to replace each MBFF instance in the circuit while meeting the constraint in *Eq.*2.17. Such exploration of MBFF cells is related to timing values $setup^j_{i,k}$[3] in *Eq.*2.18, $id1_{i,k}$ in *Eq.*2.21, $clk2q^j_{i,k}$[4] in *Eq.*2.19, and $id0_{i,k}$ in *Eq.*2.20 as well as to the power value PWR_j in *Eq.*2.16. In addition, *Eq.*2.22 and *Eq.*2.23 ensure that every MBFF instance in the target circuit should be replaced by exactly one MBFF cells in $\mathcal{L}'_{MB}$.

Step 3.3 (Timing Elaboration):

In the following, we elaborate the timing optimized in Step 3.2 to deal with the case where all timing violations are not completely resolved in Step 3.2: We replace the term '0' in *Eq.*2.17 with an epsilon (negative) value, which varies through iterations in a binary search, to quickly find a minimal absolute epsilon value that satisfies the ILP formulation. For example, suppose we set the epsilon precision to 0.25 and if our ILP formulation with an initial epsilon = -10 succeeds in finding a solution, we reset epsilon to the mean value (= -5) of 0 (target) and -10, and apply the ILP solver again. This process is repeated to approximate the worst negative timing slack as long as the ILP succeeds in finding a solution and the epsilon gap updated in an iteration is not

[2]The hold time constraints can also be expressed in a similar way. We omit the expression for brevity.

[3]The values are computed by the setup time function $F_{Setup}(tr_D, tr_{CLK})$.

[4]The values are computed by the clock-to-Q delay function $F_{C2Q}(ld_Q, tr_{CLK})$.

smaller than the precision of 0.25. When a minimal absolute epsilon value is found, we can continuously optimize the total negative timing slack by iteratively applying the elaborating process. Let M be the set of MBFF instances with sized transistor in the previous iteration. Then, at the beginning of the current iteration, we fix the transistor size in M and remove the timing constraints according to the pins in M from the timing constraints. We then assign the best minimal absolute epsilon value to the initial epsilon value, and continue the process until the best minimal absolute epsilon value from the previous iteration is the same with the best minimal absolute epsilon value in the current iteration. The overall flow of timing elaboration is shown in Fig. 2.8

2.3 Experimental Results

2.3.1 Experimental Setup

We implemented our proposed DTCO flow MBFF-opt by using C++ and Gurobi optimizer [38] as an ILP solver in a linux machine with Intel i7-4770K 3.5GHz CPU and 32GB memory and demonstrate the efficacy of our proposed flow by comparing placement and routing PPA (power-performance-area) results with two conventional flows: (1) placement and routing with no use of MBFFs, which we label Conv. No-banking, (2) placement and routing with use of MBFFs imposing non-flexible of flipping and footprint waste, which we label Conv. MBFF. The placement and routing flows of Conv. No-banking, Conv. MBFF, and our MBFF-opt are depicted in Fig. 2.9.

We used IWLS 2005 OpenCores benchmark circuits [39] for the experiments. Table 2.4 shows the statistics on the benchmark circuits, which include the total number of cell instances ("#cells"), the number of flip-flop cell instances ("#FFs"), and the number of nets ("#nets"). We synthesized and implemented the circuit by using Synopsys *Design Compiler* and Cadence *Innovus*[5] using ASAP7 7nm standard cell library.

[5]For grouping flip-flops, in pre-CTS optimization step, we used commands

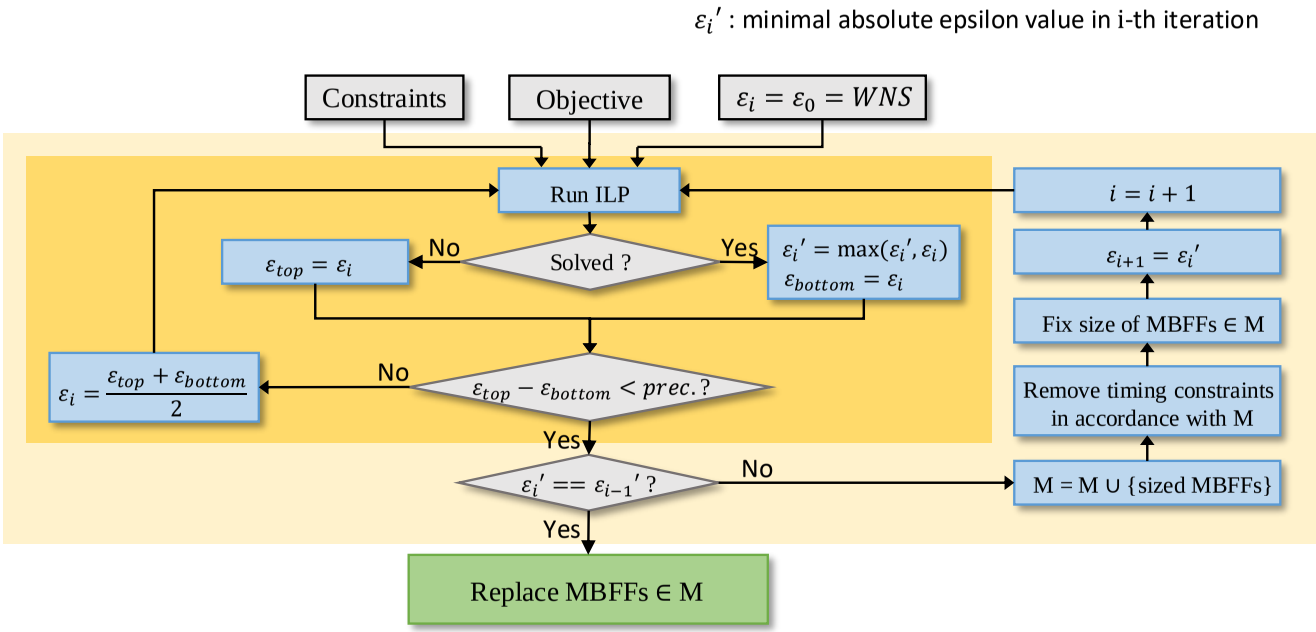


Figure 2.8: Timing elaboration flow

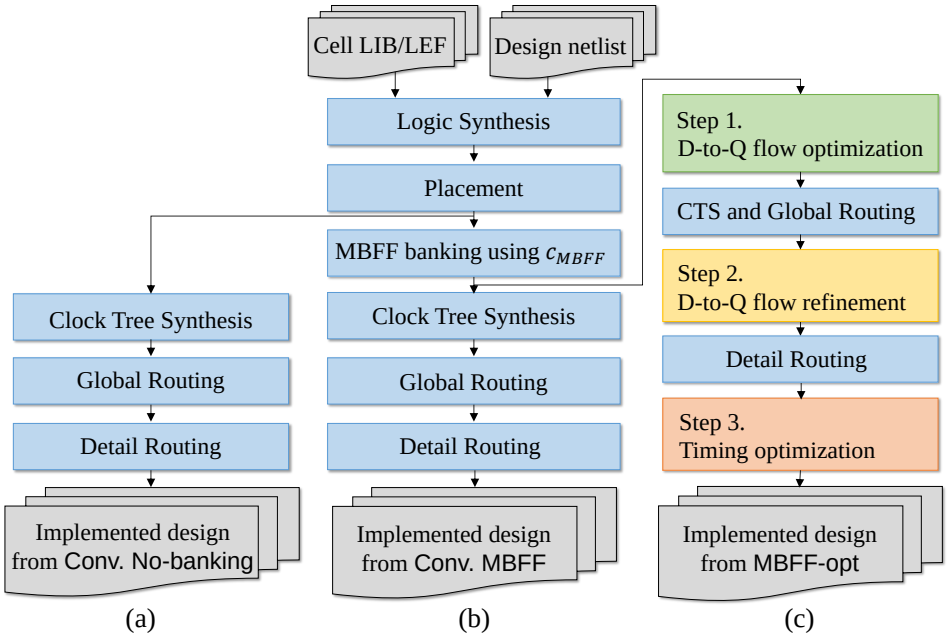


Figure 2.9: Three different flows of placement and routing conducted in our experiments. (a) Conv. No-banking: Conventional flow with no use of MBFFs. (b) Conv. MBFF: Conventional flow with use of MBFFs imposing non-flexible of flipping and footprint waste. (c) MBFF-opt: Our proposed flow with use of MBFFs resolving non-flexible of flipping and footprint waste in a way to enhance routability and timing.

Table 2.4: IWLS benchmark circuits used for the experiments.

Circuit	#cells	#FFs	#nets
MEM_CTRL	5460	1118	5575
USB_FUNCT	7949	1739	8063
AES_CIPHER	11149	530	11408
WB_CONMAX	20294	818	21424
ETHERNET	38484	10543	38580
DES3	53052	8808	53286
NOVA	119611	10864	138300

publicly available in [4][6]. We synthesized 2-bit, 4-bit MBFF cells based on ASAP7 and use them in Conv. MBFF and MBFF-opt.

2.3.2 Comparing MBFF-opt with Conventional MBFF Allocation

To assess the efficacy of our MBFF-opt, we adjusted the chip utilization and clock period for practical worst negative timing slacks (<20% of clock period) of each benchmark circuit. Table 2.5 summarizes the PPA results of the implementations produced by the flows of Conv. MBFF and MBFF-opt, which compare the die area (Area in um^2), total wirelength (WL in um), the number of design rule violations (#DRVs), the worst negative slack (WNS in ps), total negative slack (TNS in ps), and total amount of power consumption (Power in pW) [7].

`setLimitedAccessFeature FlipFlopMergeAndSplit 1` and `setOptMode -multiBitFlopOpt True`

[6]Note that the cell description formats in ASAP7 PDK library are acceptable to Cadence tool platform of placement and routing.

[7]The 5th, 7th, 9th, 11th, and 13th columns in Table 2.5 indicate the reduction rate of PPA, which represents the quantity of (Conv. MBFF - MBFF-opt) / Conv. MBFF.

Table 2.5: PPA comparison of the implementations produced by Conv. MBFF and our MBFF-opt.

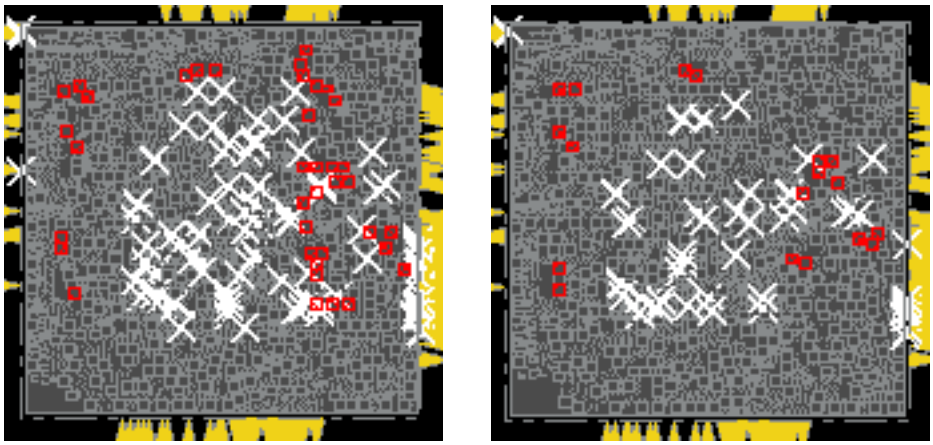
Circuit		Conv.MBFF	MBFF-opt									
			Step 1 only		Step 1 + Step 2		Step 1 + Step 3		Step 2 + Step 3		Step 1 + Step 2 + Step 3	
MEM_CTRL	Area	12445	12411	0.3%	12425	0.2%	12411	0.3%	12445	0.0%	12425	0.2%
	WL	76367	72824	4.6%	72905	4.5%	72816	4.7%	76427	-0.1%	72900	4.5%
	#DRVs	45	44	2.2%	45	0.0%	42	6.7%	45	0.0%	41	8.9%
	WNS	-39.40	-25.00	36.5%	-23.07	41.4%	-24.99	36.6%	-38.04	3.4%	-23.05	41.5%
	TNS	-926.47	-380.80	58.9%	-276.121	70.2%	-380.46	58.9%	-893.11	3.6%	-275.12	70.3%
	Power	1641.25	1559.63	5.0%	1559.38	5.0%	1559.97	5.0%	1641.10	0.0%	1559.63	5.0%
USB_FUNCT	Area	19392	19381	0.1%	19366	0.1%	19381	0.1%	19378	0.1%	19366	0.1%
	WL	131626	132864	-0.9%	132952.9	-1.0%	132899	-1.0%	132080.4	-0.3%	132981	-1.0%
	#DRVs	104	88	15.4%	83	20.2%	56	46.2%	48	53.8%	53	49.0%
	WNS	-42.00	-41.98	0.0%	-38.01	9.5%	-35.57	15.3%	-31.06	26.1%	-35.45	15.6%
	TNS	-495.11	-356.71	28.0%	-344.16	30.5%	-276.44	44.2%	-333.43	32.7%	-246.18	50.3%
	Power	7409.54	7472.88	-0.9%	7491.78	-1.1%	7474.96	-0.9%	7409.28	0.0%	7495.02	-1.2%
AES_CIPHER	Area	18547	18514	0.2%	18547	0.0%	18514	0.2%	18577	-0.2%	18547	0.0%
	WL	164955	164758	0.1%	164881	0.0%	164760.4	0.1%	165714.1	-0.5%	164885	0.0%
	#DRVs	120	121	-0.8%	114	5.0%	100	16.7%	94	21.7%	74	38.3%
	WNS	-29.30	-21.19	27.7%	-19.74	32.6%	-21.01	28.3%	-20.15	31.2%	-19.62	33.0%
	TNS	-1034.01	-467.67	54.8%	-428.57	58.6%	-358.58	65.3%	-406.73	60.7%	-369.13	64.3%
	Power	1908.88	1885.12	1.2%	1880.26	1.5%	1886.57	1.2%	1914.53	-0.3%	1881.74	1.4%
WB_CONMAX	Area	38558	38406	0.4%	38396	0.4%	38406	0.4%	38574	0.0%	38396	0.4%
	WL	466862	463540.88	0.7%	463910	0.6%	463542	0.7%	467293	-0.1%	463910	0.6%
	#DRVs	553	534	3.4%	486	12.1%	528	4.5%	492	11.0%	483	12.7%
	WNS	-86.90	-92.54	-6.5%	-85.56	1.5%	-92.54	-6.5%	-81.64	6.1%	-85.59	1.5%
	TNS	-3193.91	-2168.83	32.1%	-2024.98	36.6%	-2169.86	32.1%	-2544.04	20.3%	-2026.01	36.6%
	Power	6298.65	6073.91	3.6%	6078.82	3.5%	6073.91	3.6%	6306.75	-0.1%	6078.82	3.5%
ETHERNET	Area	103160	103358	-0.2%	103407	-0.2%	103358	-0.2%	103578	-0.4%	103407	-0.2%
	WL	849714	862133	-1.5%	861802	-1.4%	862133	-1.5%	853805	-0.5%	861800	-1.4%
	#DRVs	301	172	42.9%	154	48.8%	172	42.9%	178	40.9%	154	48.8%
	WNS	-182.02	-135.97	25.3%	-76.03	58.2%	-122.71	32.6%	-179.19	1.6%	-63.54	65.1%
	TNS	-3374.43	-2026.65	39.9%	-1617.22	52.1%	-2013.58	40.3%	-2237.14	33.7%	-1607.28	52.4%
	Power	11263.73	11562.93	-2.7%	11543.79	-2.5%	11563.01	-2.7%	11469.42	-1.8%	11543.95	-2.5%
DES3	Area	106526	106705	-0.2%	106724.4	-0.2%	106705.2	-0.2%	106539.8	0.0%	106724	-0.2%
	WL	560175	556374	0.7%	556978.2	0.6%	556349.5	0.7%	560177.8	0.0%	556973	0.6%
	#DRVs	85	82	3.5%	77	9.4%	68	20.0%	64	24.7%	61	28.2%
	WNS	-33.93	-26.64	21.5%	-25.11	26.0%	-26.63	21.5%	-28.49	16.0%	-24.90	26.6%
	TNS	-2830.64	-2947.90	-4.1%	-2724.63	3.7%	-2933.11	-3.6%	-2768.56	2.2%	-2706.88	4.4%
	Power	25568.68	25769.10	-0.8%	25750.42	-0.7%	25769.73	-0.8%	25596.80	-0.1%	25751.38	-0.7%
NOVA	Area	346386	346466	0.0%	346395	0.0%	346466	0.0%	346060	0.1%	346395	0.0%
	WL	3458498	3464932	-0.2%	3465874	-0.2%	3464935	-0.2%	3458250	0.0%	3465880	-0.2%
	#DRVs	2392	2347	1.9%	1996	16.6%	2346	1.9%	1842	23.0%	1997	16.5%
	WNS	-452.25	-216.59	52.1%	-200.42	55.7%	-216.58	52.1%	-353.22	21.9%	-200.41	55.7%
	TNS	-11312.50	-6727.85	40.5%	-2692.37	76.2%	-6720.77	40.6%	-12121.60	-7.2%	-2686.35	76.3%
	Power	6462.16	6419.95	0.7%	6419.51	0.7%	6420.00	0.7%	6449.61	0.2%	6419.58	0.7%
Average	Area	92145	92177	0.0%	92180	0.0%	92177	0.0%	92165	0.0%	92180	0.0%
	WL	815457	816775	-0.2%	817043	-0.2%	816776	-0.2%	816250	-0.1%	817047	-0.2%
	#DRVs	514	484	5.9%	422	17.9%	473	8.0%	395	23.3%	409	20.5%
	WNS	-123.68	-79.99	35.3%	-66.85	46.0%	-77.15	37.6%	-104.54	15.5%	-64.65	47.7%
	TNS	-3309.58	-2153.77	34.9%	-1444.01	56.4%	-2121.83	35.9%	-3043.52	8.0%	-1416.71	57.2%
	Power	8650.41	8677.65	-0.3%	8674.85	-0.3%	8678.31	-0.3%	8683.93	-0.4%	8675.73	-0.3%

The column labeled Step 1 only in Table 2.5 shows the results produced by applying our D-to-Q flow optimization (i.e., Step 1) in MBFF-opt. In summary, it reduces the worst negative slack and total negative slack by 35.3% and 34.9% on average even with 5.9% fewer DRVs in comparison with the implementations produced by Conv. MBFF. By integrating our D-to-Q flow refinement step (i,e., Step 2) into MBFF-opt, as shown in the column labeled Step 1 + Step 2, we are able to further reduce the worst negative slack and total negative slack by 46.0% and 56.4% on average even with 17.9% fewer DRVs in comparison with the implementations produced by Conv. MBFF. In addition, by integrating our timing optimization step (i.e., Step 3) as well as Steps 1 and 2 into MBFF-opt, as shown in the column labeled Step 1 + Step 2 + Step 3, we can further reduce the worst negative slack and total negative slack at a little power cost incurred by the MBFF transistor upsizing for timing. Fig. 2.10 shows the comparison of the distribution of DRVs and timing violation registers on the implementations of circuit USB_FUNCT in Table 2.5 produced by Conv. MBFF and our MBFF-opt. By comparing the results in the columns labeled 'Step 1 + Step 2', 'Step 1 + Step 3', 'Step 2 + Step 3', and 'Step 1 + Step 2 + Step 3', it is observed that the complete MBFF-opt flow makes the highest improvement on the worst and total negative slacks as well as the number of DRVs at a little cost of power increase.

Table 2.6 shows statistics on the number of MBFF cell instances labeled as #MBFF (#M2 for 2-bit MBFFs and #M4 for 4-bit MBFFs) in the initial circuits produced by multi-bit banking, the number of MBFF cell instances labeled as #Flipping 1, replaced in Step 1 (i.e., by D-to-Q flow flipping) of our MBFF-opt, the number of MBFF cell instances labeled as #Flipping 2, replaced in Step 2 (i.e., by D-to-Q flow refinement) of our MBFF-opt, and the number of MBFF cell instances labeled as #Sizing, replaced in Step 3 (i.e., by transistor upsizing) of MBFF-opt.

Table 2.6: The number of MBFF cell instances replaced by our MBFF-opt.

Circuit	Initial		MBFF-opt				
	#MBFF		#Flipping 1		#Flipping 2		#Sizing
	#M4	#M2	#M4	#M2	#M4	#M2	#M4
MEM_CTRL	186	9	128	2	2	0	3
USB_FUNCT	427	9	303	3	1	0	17
AES_CIPHER	128	7	88	3	30	2	13
WB_CONMAX	44	5	9	0	2	0	0
ETHERNET	2296	42	1066	11	90	1	1
DES3	2170	44	889	4	63	1	1
NOVA	6051	112	1194	17	8	0	4



(a) Conv. MBFF : (104 DRVs, 75 timing viols.)

(b) MBFF-opt : (53 DRVs, 21 timing viols.)

Figure 2.10: Comparison of the distribution of DRVs (white crosses) and timing violation registers (red rectangles) on the implementations of circuit USB_FUNCT produced by Conv. MBFF and our MBFF-opt. (a) Conv. MBFF: 104 DRVs and 75 timing violations. (b) MBFF-opt: 53 DRVs and 21 timing violations.

Table 2.7: PPA comparison of the implementations produced by Conv. No-banking, Conv. MBFF, and our MBFF-opt. The unit of Conv. MBFF and MBFF-opt is percentage(%) and the blue-colored numbers indicate improvement in comparison with that in Conv. No-banking.

Circuit	Conv. No-banking						Conv. MBFF (%)						MBFF-opt (%)					
	Area	WL	#DRVs	WNS	TNS	Power	Area	WL	#DRVs	WNS	TNS	Power	Area	WL	#DRVs	WNS	TNS	Power
MEM_CTRL	12395	70778	35	-27.4	-464.0	2862.3	-0.4	-7.9	-28.6	-44.0	-99.7	42.7	-0.2	-3.0	-17.1	15.7	40.7	45.5
USB_FUNCT	18929	121289	66	-6.6	-13.5	16743.0	-2.4	-8.5	-57.6	-541.3	-3559.3	55.7	-2.3	-9.7	18.2	-446.0	-2180.1	55.3
AES_CIPHER	18278	162510	87	0.0	0.0	2944.3	-1.5	-1.5	-37.9	N/A	N/A	35.2	-1.5	-1.5	14.9	N/A	N/A	36.1
WB_CONMAX	38470	470776	518	-177.7	-3369.8	6715.8	-0.2	0.8	-6.8	51.1	5.2	6.2	0.2	1.5	6.8	51.8	39.9	9.5
ETHERNET	100443	780932	427	-63.1	-1210.0	32624.5	-2.7	-8.8	29.5	-188.3	-178.9	65.5	-3.0	-10.4	63.9	-0.6	-32.8	64.6
DES3	104535	525733	44	-9.7	-45.4	52382.6	-1.9	-6.6	-93.2	-250.9	-6130.3	51.2	-2.1	-6.0	-45.5	-167.6	-5361.1	50.8
NOVA	340636	3220901	1974	-39.9	-761.4	13700.6	-1.7	-7.4	-21.2	-1034.5	-1385.7	52.8	-1.7	-7.6	-1.2	-402.7	-252.8	53.1
Average	90527	764703	450	-46.3	-837.7	18281.9	-1.5	-5.7	-30.8	-334.7	-1891.4	44.2	-1.5	-5.2	5.7	-158.2	-1291.0	45.0

2.3.3 Comparing MBFF-opt with Conventional No-Banking Flow

Table 2.7 shows PPA (performance, power, area) comparison of the implementations produced by the conventional flow with no use of MBFFs and MBFF-opt[8] Grouping individual flip-flops to make MBFFs results in an inflexible cell placement, possibly causing inferior circuit timing as well as less routability. However, as shown in Table 2.7, MBFF-opt performs well in overcoming the potential loss of timing and routability even less power consumption.

2.3.4 Runtime Analysis of MBFF-opt

Table 2.8 shows the runtime of Step 1, Step 2, and Step 3 in MBFF-opt. Steps 2 and 3 take 85.1% and 14.7% of the total runtime which takes most of the total runtime, as shown in Table 2.8 due to checking the congestion and timing in iterations in Step 2 and iterative ILP solving in Step 3 (We set time limit to 30 seconds in solving one round of ILP iteration in Step 3.) Fig. 2.11(b) shows ILP runtimes in Step 3 according to the number of MBFF instances in all benchmark circuits. It shows that the ILP runtime intends to increase, as the number of MBFFs increases, which in fact increases the number of variables used in ILP.

2.3.5 Comparing MBFF-opt with Conventional No-Banking flow with more timing-optimized MBFF banking design

To demonstrate the effectiveness of our MBFF-opt approach in a more timing-optimized design, we upgraded our version of Innovus and employed high timing effort to merge and split MBFFs using the commands `setOptMode -MBFFSplitTimingEffort high` and `setOptMode -MBFFMergeTimingEffort high`. Table 2.9 shows a PPA comparison of the implementations produced by the conventional flow with no use of MBFFs and conv. MBFF and our MBFF-opt, in which conv. MBFF yields

[8]The values in columns Conv. MBFF and MBFF-opt indicate reduction rate on PPA, which is the outcome of Conv. No-banking minus Conv. MBFF or MBFF-opt divided by Conv. No-banking.

Table 2.8: Runtime of Steps 1, 2, and 3 in MBFF-opt.

Circuit	Runtime(s)		
	Step 1	Step 2	Step 3
MEM_CTRL	<1	1	93
USB_FUNCT	<1	397	282
AES_CIPHER	<1	699	364
WB_CONMAX	<1	436	2
ETHERNET	7	510	659
DES3	5	1071	720
NOVA	17	12955	660
Average	4	2295	397

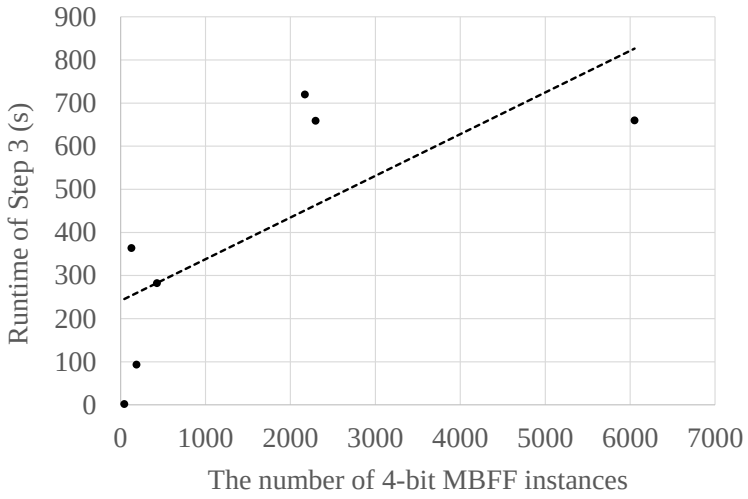


Figure 2.11: Changes of ILP runtime as the number of 4-bit MBFF instances changes in Step 3.

better designs in terms of WNS and TNS compared to the conv. MBFF results shown in Table 2.7. Table 2.9 demonstrates that our MBFF-opt further improves WNS, TNS, and #DRVs compared to both the conv. MBFF and Conv. No-banking designs. On average, the area improvement rate achieved by the conv. MBFF and MBFF-opt is -0.6% and -0.8%, respectively. These results show that our MBFF-opt performs well in mitigating potential timing and routability issues, even in highly timing-optimized designs.

Table 2.9: PPA comparison of the implementations produced by Conv. No-banking, Conv. MBFF, and our MBFF-opt using high timing effort to merge and split MBFF. The unit of Conv. MBFF and MBFF-opt is percentage(%) and the blue-colored numbers indicate improvement in comparison with that in Conv. No-banking.

Circuit	Conv. No-banking				Conv. MBFF (%)				MBFF-opt (%)			
	#DRVs	WNS	TNS	Power	#DRVs	WNS	TNS	Power	#DRVs	WNS	TNS	Power
MEM_CTRL	35	-27.36	-464.03	2862.25	-29%	-44%	-100%	43%	-17%	16%	41%	46%
USB_FUNCT	27	-14.38	-68.27	17667.56	15%	-12%	-46%	50%	33%	-2%	-7%	50%
AES_CIPHER	18	-49.13	-4063.21	3312.96	-56%	-63%	-82%	22%	83%	-17%	-38%	19%
WB_CONMAX	518	-177.65	-3369.80	6715.79	-7%	51%	5%	6%	7%	52%	40%	9%
ETHERNET	144	-7.77	-97.89	35337.98	-31%	-6%	29%	67%	39%	100%	100%	67%
DES3	116	-24.16	-3765.63	60685.98	11%	-53%	-228%	30%	53%	-44%	-202%	30%
NOVA	859	-181.14	-4169.55	17127.87	-13%	-20%	-86%	53%	38%	100%	100%	53%
Average	450.14	-46.32	-837.75	18281.87	-16%	-21%	-72%	39%	34%	29%	5%	39%

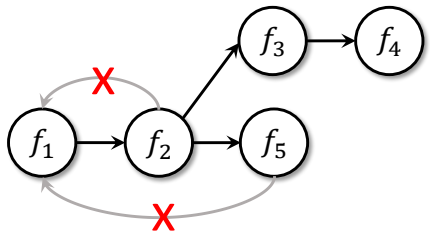
Chapter 3

Minimally Allocating Always-on State Retention Storage for Supporting Power Gating Circuits

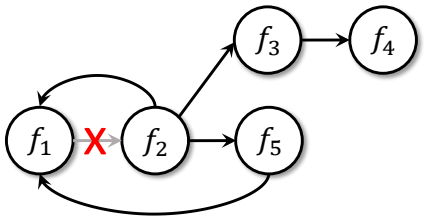
3.1 Motivations

• **Dealing with cycles, including self-loops**: The cycles in a flip-flop dependency graph make the allocation problem hard to be solved optimally since at least one flip-flop in every cycle should be replaced with a flip-flop with retention storage, otherwise we have no idea which flip-flop(s) should initiate the state restoration. All the existing works, as a preprocessing step, cut the cycles to produce a flip-flop dependency graph $G(V, A')$ with no cycle, and apply their allocation algorithms to $G(V, A')$. For example, in [3], an iterative heuristic based on the *minimum cost feedback vertex set (MFVS)* [40] is used to break all cycles with a minimal number of cuts. Note that the pre-processing step worsens the allocation quality due to the fact that the minimal cycle-cut is an NP-hard problem and the cycle-cut process in all existing works is completely decoupled from their allocation algorithms. For example, Figs. 3.1(a) and (b) show two possible cycle-cut results (labeled G_1 and G_2) on a flip-flop dependency graph with two cycles $f_1 \rightarrow f_2 \rightarrow f_1$ and $f_1 \rightarrow f_2 \rightarrow f_5 \rightarrow f_1$. Then, Figs. 3.1(c) and (d) respectively show optimal MBRFF allocations with wakeup latency constraint

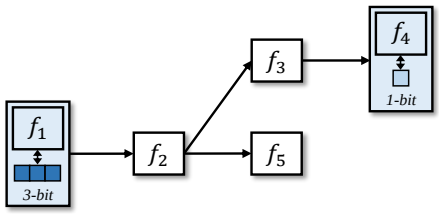
$l = 3$ for G_1 and G_2 in Figs. 3.1(a) and (b), in which G_1 uses 4 retention bits while G_2 uses 3. On the other hand, *our proposed optimal algorithm performs cutting cycles and allocating MBRFFs simultaneously.*



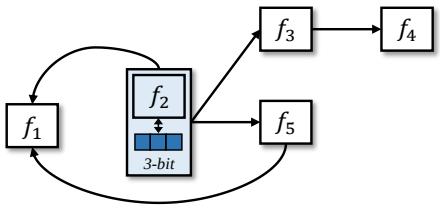
(a) Acyclic graph (G_1) produced by cutting $f_2 \to f_1$ and $f_5 \to f_1$.



(b) Acyclic graph (G_2) produced by cutting $f_1 \to f_2$.



(c) Optimal MBRFF allocation with $l = 3$ for G_1 in (a), using 4 total bits.



(d) Optimal MBRFF allocation with $l = 3$ for G_2 in (b), using 3 total bits.

Figure 3.1: Example illustrating the effect of cycle-cut on allocation quality.

• **Impact of wakeup latency constraint**: More saving on retention storage is generally expected as we increase the value of wakeup latency constraint l. Fig. 3.2 shows a comparison of the amount of retention storage reduction when l is set to 2, 3, and 4 clock cycles, saying that for l exceeding 3, very little saving is achieved. In other words, $l = 2$ or 3 suffices for MBRFF allocation. Consequently, we focus on the MBRFF allocation problems for $l = 2$ and $l = 3$, and *propose optimal solutions to both of $l = 2$ and $l = 3$.*

• **Supporting multiple objective functions**: All existing MBRFF allocation algorithms target one objective, which is to minimize the total number of retention bits because it indirectly minimizes the leakage power on retention storage or total reten-

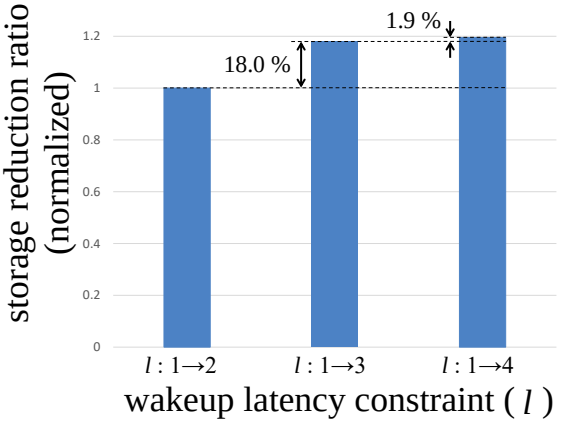


Figure 3.2: Comparison of retention storage reduction produced by [2] on IWLS [5] circuits when the wakeup latency constraint l is set from 1 to 2, 1 to 3, and 1 to 4 clock cycles.

tion area. Table 3.1 shows the leakage power on the retention storage in MBRFFs in Synopsys generic library (using a tri-state buffer between latches) and [1] (using delay buffers and transmission gate between latches), explaining that for some MBRFF implementation, there is a big mismatch between the leakage power in retention storage of k-bit MBRFF and the k times of the leakage power on 1-bit MBRFF. In addition, Table 3.2 shows the area of MBRFFs in Synopsys generic library and [1], indicating a trend similar to that in the leakage power. Thus, for minimizing leakage power or total area, it is necessary to use the leakage power or total area as the direct cost function to be minimized rather than using total retention bits. So far, no works have considered those cost functions in their allocation algorithms. *Our proposed algorithm accepts the three objectives of total bits, leakage, and area.*

3.2 Optimal MBRFF Allocation Algorithm for l = 2

Our MBRFF allocation algorithm called MBRFF-opt accepts the original flip-flop dependency graph, $G(V, A)$, of a target circuit for power gating as input. Then, it

Table 3.1: Leakage power on the always-on retention storage in k-bit MBRFFs in Synopsys 32nm generic library and Chen [1].

k	Synopsys (nW)	ratio	Chen [1] (nW)	ratio
1	61.96	1.00	40.50	1.00
2	137.77	**2.22**	80.89	1.99
3	197.69	**3.19**	121.28	2.99

Table 3.2: Area of k-bit MBRFFs in Synopsys 32nm generic library and Chen [1]. ($k = 0$ indicates flip-flop with no retention storage.)

k	Synopsys (um^2)	ratio	Chen [1] (um^2)	ratio
0 (no ret.)	6.608	-	6.610	-
1	13.215	1.00	11.183	1.00
2	18.502	**1.79**	14.233	**1.67**
3	23.780	**2.60**	17.283	**2.33**

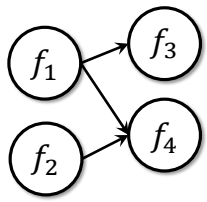
performs three steps: (1) MBRFF-opt transforms G into G' by node replication to facilitate generating every segment that corresponds to an MBRFF allocation; (2) It finds a set of segments that can cover G' with no overlap such that it produces a minimal quantity of the total number of retention bits, total leakage power or total area of retention storage; (3) MBRFF-opt allocates retention storage according to the covering segments obtained in (2).

3.2.1 Transforming Flip-flop Dependency Graph

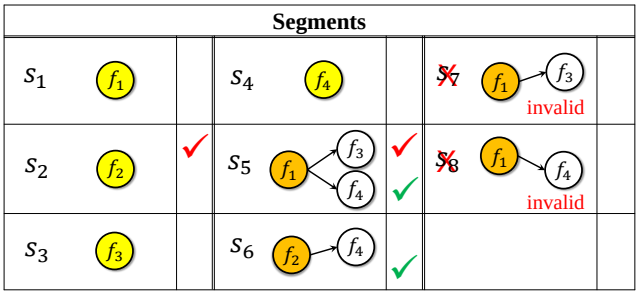
We illustrate why the original graph is required to be updated before applying the subsequent step using an example of MBRFF allocation in Fig. 3.3, in which we want to allocate MBRFFs of a minimal total number of retention bits for the flip-flop dependency graph G in Fig. 3.3(a). Then, Fig. 3.3(b) lists all partial subgraphs (called *covering segments*), each of which can be entirely allocated with a 1-bit or a 2-bit MBRFF. Precisely, segments $s_1, \cdots, s_4$ can be assigned with 1-bit MBRFFs while $s_5, \cdots, s_8$ assigned with 2-bit MBRFFs, but s_7 and s_8 can be removed since $s_7 \subset s_5$ and $s_8 \subset s_5$, and the costs of s_7 and s_8 are not smaller than that of s_5.

Then, minimal covering segments for G are s_2 and s_5 (marked with red checking). Fig. 3.3(c) shows the two-cycle state recovering process for the MBRFF allocation to s_2 and s_5. However, the state of flip-flop f_4 will not be restored since f_2 never feeds f_4 during the wakeup cycles t_1 and t_2. Instead, if we choose covering segments s_5 and s_6 (marked with green checking), resulting in one more retention bits, the state restoration process can be done successfully, as shown in Fig. 3.3(d). As a result, this leads to *transform G to G' by including node replication for every node that is driven by multiple flip-flops*, so that s_2 and s_5 cannot be a complete cover on G'.

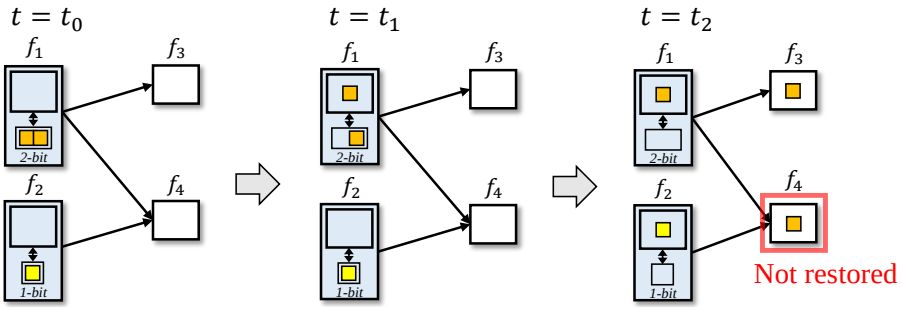
Precisely, we include a new node f_{4_1} and update the segments s_4 and s_6 as shown in Fig. 3.3(e). Note that a node with self-loop will be duplicated as well. For example, Fig. 3.4(b) shows the transformed graph G' of the original graph G in Fig. 3.4(a), in which f_3 is replicated twice since f_3 is driven by three flip-flops including f_3 itself.



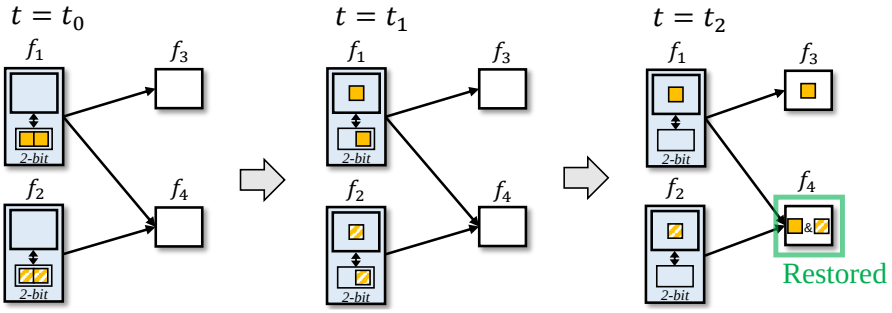
(a) Flip-flop dependency graph G.



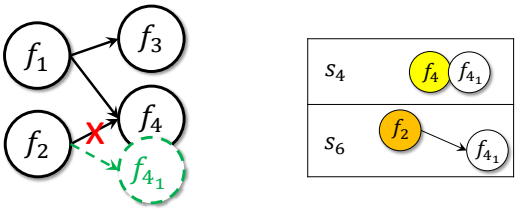
(b) Covering segments of G.



(c) Restoration process with red checked segments (s_2, s_5).



(d) Restoration process with green checked segments (s_5, s_6).



(e) Node replication and updating segments.

Figure 3.3: Example of transforming an original flip-flop dependency graph to a feasible covering graph for MBRFF allocation.

Let $P = \{f_{j_1}, f_{j_2}, \cdots\}$ be the set of nodes (i.e., flip-flops) that drive f_i in G. **Node replication rule** is that: *if* $|P| > 1$*, we replicate* f_i *to produce* $R = \{f_{i_1}, f_{i_2}, \cdots, f_{i_m}\}$ *where* $m = |P| - 1$ *and all replicated nodes together with* f_i *are collectively considered in* G' *when generating edges and covering segments.* Such node replication will constrain that if we want to restore the state of f_i in G with the help of its driving flip-flops, all the driving flip-flops (i.e. P) should be allocated with 2-bit MBRFFs since, otherwise, it causes an incomplete state restoration, as illustrated in Figs. 3.3(c), (d), and (e) where (**incoming edge updating rule**) $R = \{f_{4_1}\}$ is added to G' with edge $f_2 \rightarrow f_{4_1}$ while deleting $f_2 \rightarrow f_4$. This node replication and incoming edge updating lead to the following fact:

Fact 1. *Restoring the state of* f_i *in* G *can be done by completing state restoration on* f_i *in* G' *as well as all replicated nodes in its* R*. (The flip-flops in* R *are assumed to have 'virtual' states.)*

Fig. 3.4 shows a comprehensive example of node replication in G for generating feasible covering segments. Since f_3 in the flip-flop dependency graph in Fig. 3.4(a) has three driving nodes f_1, f_2, and f_3 (i.e., self-loop), two more nodes f_{3_1} and f_{3_2} are created with edges $f_1 \rightarrow f_3$ and $f_2 \rightarrow f_{3_1}$, as shown in Fig. 3.4(b).[1]

Except for f_{3_2} caused by self-loop, (**outgoing edge updating rule**) f_3 and f_{3_1} in G' have outgoing edges to both f_4 and f_5, which are the flop-flops driven by f_3 in G to ensure that an MBRFF allocation to f_3 in G can be done by allocating the MBRFF to $\{f_3, f_{3_1}, f_{3_2}\}$ in G'. This node replication and outgoing edge updating lead to the following fact:

Fact 2. *Allocating an MBRFF to a flip-flop* f_i *in* G *can be done by allocating the MBRFF to* f_i *in* G' *and MBRFFs of the same bit size to all nodes in its* R*. (The flip-flops in* R *are assumed to be allocated with 'virtual' MBRFFs.)*

Based on *Facts* 1 and 2, Fig. 3.4(c) shows all possible covering segments produced from G' in Fig. 3.4(b) where segments $s_1, \cdots, s_6$ and $s_7, \cdots, s_{11}$ are responsible for

[1] $f_3 \rightarrow f_{3_2}$ is not shown since f_3 and f_{3_2} are the same flip-flop.

all possible 1-bit and 2-bit MBRFF allocations, respectively. In particular, segment s_3 intends to allocate a 1-bit MBRFF collectively to $\{f_3, f_{3_1}, f_{3_2}\}$ while segment s_9 intends to allocate a 2-bit MBRFF collectively to $\{f_3, f_{3_1}, f_{3_2}\}$. MBRFF-opt generates all segments from G' for 1-bit and 2-bit MBRFF allocations in a way to satisfy the following fact:

Fact 3. *Every segment of (i) single nodes and (ii) pairs of nodes with driving-driven relation in G' produced by* MBRFF-opt *is responsible for allocating exactly a distinct MBRFF of 1-bit size for (i) and 2-bit size for (ii) in G.*

3.2.2 Minimal-cost Covering for the Transformed Graph

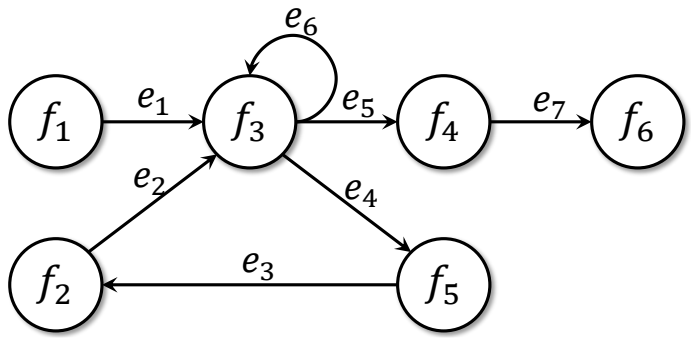
Let $S = \{s_1, s_2, \cdots\}$ be the covering segments produced by Step 1 in MBRFF-opt. Then, we formulate the problem of extracting a subset of S which has the minimal quantity of each objective into a weighted unate covering problem (UCP) and solve it optimally.

The weighted UCP [41] is, given a matrix M of m rows and n columns, for which $M_{i,j}$ is either 0 or 1, the problem of finding a column subset U with a minimum total weight that satisfies

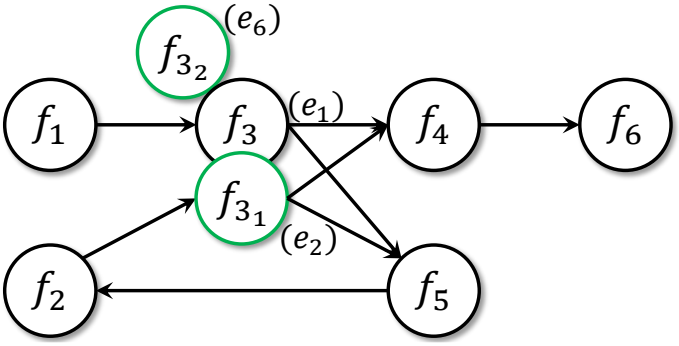
$$\exists_{j \in U} M_{i,j} = 1, \forall i \in \{1, \cdots, m\}. \tag{3.1}$$

That is, the columns in the set U cover M in the sense that every row of M contains an 1-entry in at least one of the columns of U, and there is no smaller-cost set that also covers M. The matrix M is called *constraint matrix*.

We construct a constraint matrix with columns corresponding to S and rows corresponding to the flip-flops including replicated nodes. $M_{i,j} = 1$ if the flip-flop in the i^{th} row is in the covering segment in the j^{th} column, and $M_{i,j} = 0$, otherwise. For example, Fig. 3.5(a) shows the constraint matrix of the transformed flip-flop dependency graph in Fig. 3.4(b), in which the matrix has eight rows, one for each flip-flop, and eleven columns, one for each segment. The columns enclosed by red dashed lines



(a) An initial flip-flip dependency graph G



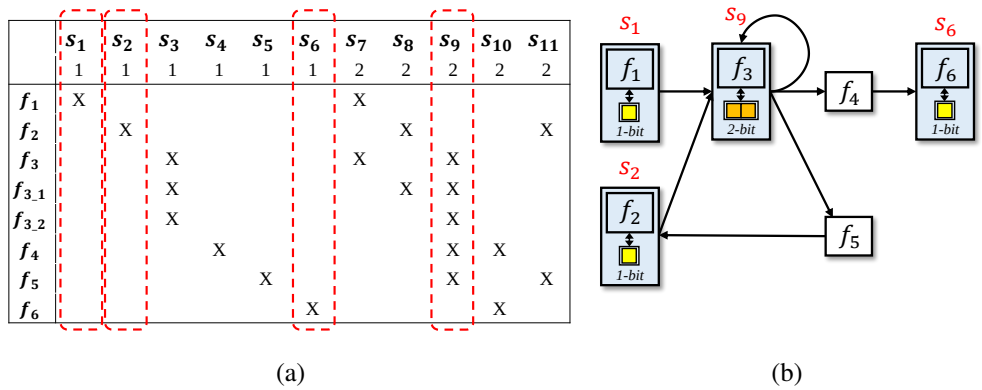
(b) Graph G' transformed from G by node replication and edge updating

Segments		
s_1 f_1	s_5 f_5	s_9 f_{3_2} f_3 f_{3_1} f_4 f_5
s_2 f_2	s_6 f_6	s_{10} f_4 f_6
s_3 f_3 f_{3_1} f_{3_2}	s_7 f_1 f_3	s_{11} f_2 f_5
s_4 f_4	s_8 f_2 f_{3_1}	

f_i : cost = 1 f_i : cost = 2

(c) All covering segments for satisfying *Facts* 1 and 2

Figure 3.4: An illustration of node replication, edge updating, and segment generation to maintain consistency, i.e., *Fact* 3, with the flip-flop state restoration process in G.



(a) (b)

Figure 3.5: (a) Construction of constraint matrix and UCP solution. (b) MBRFF allocation for the solution in (a).

indicate a minimal-cost subset to cover all flip-flops and the sum of the weights of those columns, which is 5, is the value of the minimum total cost.

3.2.3 Allocating MBRFFs According to Minimal-cost Covering

This step allocates an MBRFF to every covering segment extracted in Step 2. For example, Fig. 3.5(b) shows the MBRFF allocation result according to the selection of covering segments in Fig. 3.5(a) by Step 2 of MBRFF-opt. The validation of the correctness of MBRFF allocation results by MBRFF-opt is supported by *Lemma* 1 and *Theorem* 1.

Lemma 1. *For every cycle, including self-loop in* G, MBRFF-opt *with* $l = 2$ *allocates 1-bit or 2-bit MBRFF(s) to at least one flip-flop in the cycle.*

Proof. If a cycle in G is self-loop, the self-loop flip-flip will be duplicated in G'. According to *the incoming edge updating rule*, the duplicated node, say f_d, (e.g., f_{3_2} in Fig. 3.4(b)) has no incoming edge. Thus, to cover f_d, UCP solver must select a segment containing f_d for triggering the state restoration. If a cycle in G is not self-loop, the cycle has at least two nodes in G' with driving-driven relation, say $f_i \rightarrow f_j$. If no MBRFF were allocated to f_j, UCP solver will select a segment consisting of f_i

and f_j to cover f_j, resulting in allocating a 2-bit MBRFF to f_i; if no MBRFF were allocated to f_i, by *Fact 2* UCP solver will select a segment containing f_j to cover f_j, resulting in allocating 1-bit or 2-bit MBRFF to f_j. □

The abundance of self-loop flip-flops in circuits i.e., 62%∼87%, as summarized in Table 3.3 and the claim in *Lemma* 1 clearly explain the inherent ineffectiveness of further reducing the total bits of retention storage even though the wakeup latency l increases to over 3, as shown in Fig.3.2 in Sec. 3.1.

Theorem 1. MBRFF-opt *allocates MBRFFs optimally while ensuring a correct and non-conflicting state restoration with* $l = 2$.

Proof. Since UCP solver (i.e., Step 2 of MBRFF-opt) selects a subset of non-overlapping covering segments to cover G' such that it has the least total cost, it directly leads to an optimal MBRFF allocation according to *Fact* 3. In addition, the correctness of the state restoration is justified by the node replication in G together with edge updating to satisfy *Facts* 1 and 2 as well as by generating covering segments for G' to satisfy *Fact* 3. □

3.3 Extending Optimality of MBRFF-opt for l = 3

3.3.1 Extending Node Replication and Edge Updating

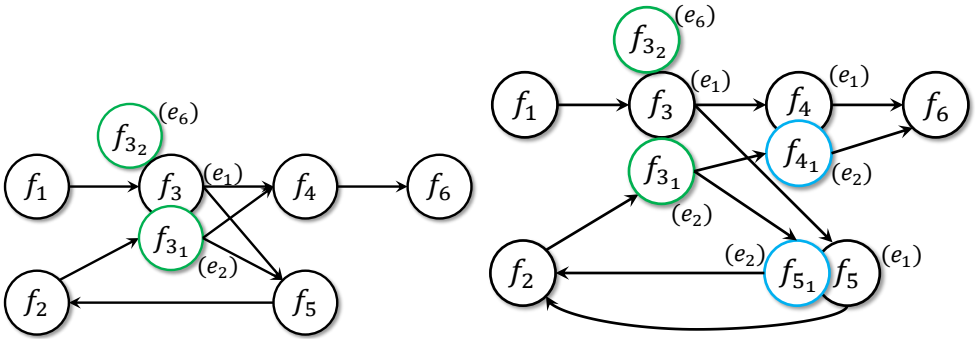
Since l is set to 3, MBRFF-opt needs to consider allocating 3-bit MBRFFs besides 1-bit and 2-bit MBRFFs. Thus, MBRFF-opt requires covering segments of the form $f_i \rightarrow f_j \rightarrow f_k$. MBRFF-opt applies the rules of node replication and edge updating to G, assuming $l = 2$, to produce G'. Again, MBRFF-opt applies node replication and edge updating for the nodes in G' driven by the nodes that were involved in replication in G' to produce G'', so that G'' allows to generate all segments of the form $f_i \rightarrow f_j \rightarrow f_k$. We illustrate the construction of G'' using G in Fig. 3.4(a). First, we construct G' as shown in Fig. 3.6(a) from G to satisfy *Facts* 1, 2, and 3. Then, we construct G'' as

shown in Fig. 3.6(b) from G' by duplicating f_4 and f_5 and updating edges following the rules in Sec. 3.2.1. Thus, G'' contains $R = \{f_{4_1}\}$ for f_4 and $R = \{f_{5_1}\}$ for f_5 in addition to $R = \{f_{3_1}, f_{3_2}\}$ for f_3, all of which are required to support *Facts* 1 and 2. Fig. 3.6(c) shows all segments produced by MBRFF-opt from G'' for 1-bit, 2-bit, and 3-bit MBRFF allocations in a way to satisfy the following fact:

Fact 3 (extended). *Every segment of (i) single nodes (ii) pairs of nodes, and (iii) triples of nodes with driving-driven relation in G'' produced by* MBRFF-opt *is responsible for allocating exactly a distinct MBRFF of 1-bit size for (i), 2-bit size for (ii), and 3-bit size for (iii) in G.*

Fig. 3.7 shows the constraint matrix of G'' in Fig. 3.6(b). The columns enclosed by red dashed lines indicate a minimal-cost subset to cover all flip-flops. The sum of the weigths of those columns, which is 4, corresponds exactly to a minimum total cost of MBRFF allocation as shown in Fig. 3.7(b).

It is quite straightforward to prove the extended versions of *Lemma* 1 and *Theorem* 1 for $l = 3$. (We omit the proofs.)



(a) G' transformed from G to support up to 2-bit MBRFFs

(b) G'' transformed from G' to support up to 3-bit MBRFFs

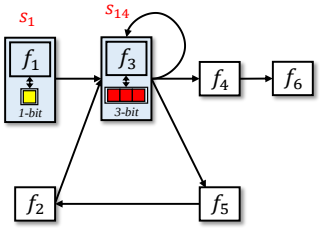
Segments (cost = 1)		Segments (cost = 2)		Segments (cost = 3)	
s_1	f_1	s_7	f_1 f_3	s_{12}	f_1 f_3 f_4
s_2	f_2	s_8	f_2 f_{3_1}	s_{13}	f_2 f_{3_1} f_{4_1} f_{5_1}
s_3	f_3 f_{3_1} f_{3_2}	s_9	f_{3_2} f_3 f_{3_1} f_4 f_{4_1} f_{5_1} f_5	s_{14}	f_{3_2} f_3 f_{3_1} f_4 f_{4_1} f_6 f_2 f_{5_1} f_5
s_4	f_4 f_{4_1}	s_{10}	f_4 f_{4_1} f_6		
s_5	f_5 f_{5_1}	s_{11}	f_2 f_{5_1} f_5	s_{15}	f_{3_1} f_2 f_{5_1} f_5
s_6	f_6				

(c) All covering segments for satisfying *Facts* 1 and 2.

Figure 3.6: An illustration of node replication, edge updating, and segment generation to maintain consistency, i.e., *Fact* 3 (*extended*), with the flip-flop state restoration process in G.

	s_1	s_2	s_3	s_4	s_5	s_6	s_7	s_8	s_9	s_{10}	s_{11}	s_{12}	s_{13}	s_{14}	s_{15}
	1	1	1	1	1	1	2	2	2	2	2	3	3	3	3
f_1	X						X					X			
f_2		X						X			X		X	X	X
f_3			X				X		X			X		X	
f_{3_1}			X					X	X				X	X	X
f_{3_2}			X						X					X	
f_4				X					X	X		X		X	
f_{4_1}				X					X	X			X	X	
f_5					X				X		X			X	X
f_{5_1}					X				X		X		X	X	X
f_6						X				X				X	

(a)



(b)

Figure 3.7: (a) Construction of constraint matrix and UCP solution. (b) MBRFF allocation for the solution in (a).

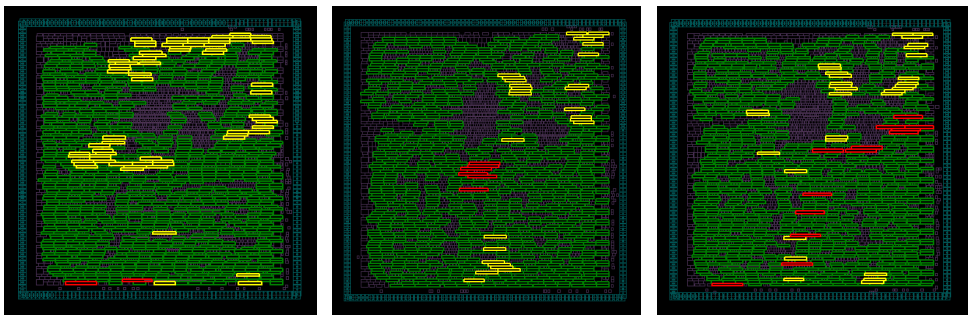
3.4 Experimental Results

We implemented our optimal MBRFF allocation algorithm MBRFF-opt in C++ and solved the weighted UCPs by linking to CPLEX [42]. IWLS benchmark circuits [5] are used to compare our MBRFF allocation results with that of the existing best-known heuristic (Fan [2]) which used MFVS technique [40] to cut cycles in the flip-flop dependency graphs of the circuits. The benchmark circuits were synthesized and implemented using Synopsys *Design Compiler* and *IC Compiler* with Synopsys 32/28nm Generic Library that contains standard cells of retention flip-flops. We set the operating clock frequency to 200MHz for all circuits. We performed our experiments on 4.7GHz Intel Core machine under the Linux operating system with 64GB memory.

Table 3.3 shows the detailed benchmark circuit information, including the number of flip-flops ("#FFs"), the number of edges ("#Edges"), and the number of self-loop flip-flops ("#Self-loop FFs (ratio)") in the flip-flop dependency graphs of the circuits.

Table 3.3: IWLS benchmark circuits

Circuit	#FFs	#Edges	#Self-loop FFs (ratio)
SPI	229	3690	195 (85%)
WB_DMA	523	7351	328 (62%)
WB_CONMAX	818	12174	658 (80%)
MEM_CTRL	1118	59954	872 (78%)
USB_FUNCT	1737	19876	1245 (72%)
AC97_CTRL	2199	14891	1723 (78%)
PCI_BRIDGE32	3403	78633	2973 (87%)



(a) (b) (c)

Figure 3.8: MBRFF distribution of the allocation results for MEM_CTRL in Table 3.4 and 3.6 where the green, yellow, and red small rectangles indicate 1-bit, 2-bit, and 3-bit MBRFFs, respectively. (a) Distribution by Fan [2] in Table 3.4 and 3.6. (b) Distribution by MBRFF-opt in Table 3.4, reducing 2-bit MBRFFs by 42 at the expense of 4 more 3-bit MBRFFs. (c) Distribution by MBRFF-opt in Table 3.6, which optimizes total area by using 11 more 3-bit MBRFFs over that in (a).

Table 3.4: Comparisons of the effectiveness of the existing best-known MBRFF allocation algorithm (Fan [2]) and our MBRFF-opt in total bits of retention storage of view. ("*" indicates the circuit is partitioned to a manageable size.)

Circuit	Fan [2] ($l = 2$ / $l = 3$)				MBRFF-opt ($l = 2$ / $l = 3$)					
	#1-bit RFF	#2-bit RFF	#3-bit RFF	#RetBits	#1-bit RFF	#2-bit RFF	#3-bit RFF	#RetBits	%Red.	Run time (sec)
SPI	229 / 229	0 / 0	- / 0	229 / 229	229 / 229	0 / 0	- / 0	229 / 229	0.00% / 0.00%	0.2 / 2.6
WB_DMA	315 / 314	101 / 100	- / 1	517 / 517	368 / 373	74 / 70	- / 1	516 / 516	0.19% / 0.19%	0.7 / 12.2
WB_CONMAX	562 / 562	128 / 128	- / 0	818 / 818	562 / 704	128 / 57	- / 0	818 / 818	0.00% / 0.00%	2.0 / 48.5
MEM_CTRL*	972 / 969	72 / 70	- / 2	1116 / 1115	1028 / 1013	44 / 28	- / 6	1116 / 1086	0.00% / 2.51%	12.4 / 201.9
USB_FUNCT	1296 / 1291	215 / 210	- / 5	1726 / 1726	1374 / 1397	176 / 120	- / 10	1726 / 1667	0.00% / **3.42%**	6.3 / 122.4
AC97_CTRL	2123 / 2123	33 / 33	- / 0	2189 / 2189	2145 / 2147	22 / 17	- / 2	2189 / 2187	0.00% / 0.09%	6.1 / 29.4
PCI_BRIDGE32*	3255 / 3213	71 / 29	- / 42	3397 / 3397	3295 / 3347	51 / 16	- / 6	3397 / 3397	0.00% / 0.00%	50.5 / 201.5
DES_PERF*	- / 59	- / 4283	- / 61	- / 8808	- / 6776	- / 584	- / 288	- / 8808	- / 0.00%	- / 180.7
ETHERNET*	- / 10363	- / 64	- / 15	- / 10536	- / 10394	- / 33	- / 24	- / 10532	- / 0.04%	- / 163.4
Avg.	1250 / 1095	89 / 607	- / 14	1427 / 3350	1286 / 2931	71 / 103	- / 37	1427 / 3249	0.03% / 0.78%	11.2 / 107.0

Table 3.5: Comparisons of the effectiveness of the existing best-known MBRFF allocation algorithm (Fan [2]) and our MBRFF-opt in total leakage power on retention storage of view. ("*" indicates the circuit is partitioned to a manageable size.)

Circuit	Fan [2] ($l = 2$ / $l = 3$)	MBRFF-opt ($l = 2$ / $l = 3$)				
	Power (μW)	#1-bit RFF	#2-bit RFF	#3-bit RFF	Power (μW)	%Red.
SPI	14.19 / 14.19	229 / 229	0 / 0	- / 0	14.19 / 14.19	0.00% / 0.00%
WB_DMA	33.43 / 33.43	523 / 523	0 / 0	- / 0	32.41 / 32.41	3.07% / 3.07%
WB_CONMAX	52.46 / 52.46	818 / 818	0 / 0	- / 0	50.68 / 50.68	3.38% / 3.38%
MEM_CTRL*	70.14 / 70.08	1115 / 1073	1 / 0	- / 5	69.22 / 67.47	1.31% / 3.72%
USB_FUNCT	109.92 / 109.91	1707 / 1637	10 / 7	- / 7	107.14 / 103.78	2.53% / 5.58%
AC97_CTRL	136.09 / 136.09	2165 / 2165	12 / 8	- / 2	135.80 / 135.64	0.21% / 0.33%
PCI_BRIDGE32*	211.46 / 211.38	3385 / 3380	6 / 1	- / 5	210.56 / 210.55	0.43% / 0.39%
DES_PERF*	- / 605.78	- / 8808	- / 0	- / 0	- / 545.74	- / **9.91%**
ETHERNET*	- / 653.87	- / 10470	- / 4	- / 18	- / 652.83	- / 0.16%
Avg.	89.67 / 209.69	1420 / 3234	4 / 2	- / 4	88.57 / 201.48	1.56% / 2.95%

Table 3.6: Comparisons of the effectiveness of the existing best-known MBRFF allocation algorithm (Fan [2]) and our MBRFF-opt in total implementation area of retention storage of view. ("*" indicates the circuit is partitioned to a manageable size.)

Circuit	FFs (μm^2)	Fan [2] ($l = 2$ / $l = 3$)	MBRFF-opt ($l = 2$ / $l = 3$)				
		Area (μm^2)	#1-bit RFF	#2-bit RFF	#3-bit RFF	Area (μm^2)	%Red.
SPI	1513.69	1047.22 / 1047.22	229 / 176	0 / 0	- / 21	1047.22 / 1028.98	0.00% / 1.74%
WB_DMA	3457.03	2210.42 / 2208.90	312 / 311	102 / 101	- / 1	2204.32 / 2202.80	0.28% / 0.28%
WB_CONMAX	5406.98	3545.77 / 3545.77	562 / 562	128 / 128	- / 0	3545.77 / 3545.77	0.00% / 0.00%
MEM_CTRL*	7389.98	4993.81 / 4986.19	968 / 987	74 / 32	- / 13	4990.77 / 4896.24	0.06% / 1.80%
USB_FUNCT	11481.57	7565.55 / 7557.94	1167 / 1210	286 / 217	- / 12	7516.87 / 7315.60	0.64% / 3.21%
AC97_CTRL	14535.39	9960.04 / 9960.04	2071 / 2067	60 / 21	- / 27	9928.06 / 9900.65	0.32% / 0.60%
PCI_BRIDGE32*	22493.83	15426.35 / 15362.38	3222 / 3125	90 / 60	- / 52	15420.28 / 15303.00	0.04% / 0.39%
DES_PERF*	58220.88	- / 33570.17	- / 303	- / 815	- / 2293	- / 32071.55	- / **4.46%**
ETHERNET*	69695.84	- / 48038.00	- / 10317	- / 47	- / 41	- / 47975.50	- / 0.13%
Avg.	19769.85	6392.74 / 14030.73	1219 / 2118	106 / 158	- / 273	6379.04 / 13804.45	0.19% / 1.40%

Table 3.7: Problem sizes before and after partitioning flip-flop dependency graphs for large circuits.

Circuit	#FFs	#Edges	Before partition		After partition (total)	
			#Segments	#Nodes	#Segments	#Nodes
MEM_CTRL	1118	59954	2852	5268K	2411	741K
PCI_BRIDGE32	3403	78633	9747	1344K	5157	422K
DES_PERF	8808	34736	26176	207K	23973	77K
ETHERNET	10544	248887	30927	2752K	13466	150K

3.4.1 Minimizing Total Number of Bits of Retention Storage

To demonstrate the efficacy of MBRFF-opt on minimizing the total number of retention bits, we set the costs of the covering segments to the bit sizes of MBRFFs accordingly while constraining the wakeup latency (l) to 2 and 3. Table 3.4 summarizes the comparison of the results, in which the values in column "#x-bit RFF" are the numbers of x-bit MBRFFs used by Fan [2] and MBRFF-opt, and the values in column "#RetBits" are the total numbers of retention bits. Note that the flip-flops with primary inputs were included in the vertex set of flip-flop dependency graph for fair comparison with the previous work. Column "Run time (sec)" indicates the run time spent by MBRFF-opt.

We partition the flip-flop dependency graphs of the circuits MEM_CTRL, PCI_BRIDGE32, and two additional circuits marked by "*" in Table 3.4, 3.5 and 3.6 into several subgraphs by using KL (Kernighan-Lin) min-cut partitioning heuristic, modifying KL to max-cut on nodes with many fanin flip-flops, to reduce the problem complexity in terms of the number of node replications and covering segments before applying MBRFF-opt with $l = 3$. Table 3.7 summarizes the reduction in the number of nodes and covering segments for the transformed graphs (i.e., G'' in Sec. 3.3.1 by the partitioning). The comparison in Table 3.4 shows that MBRFF-opt is able to save 3.42%

more retention bits over the best-known algorithm Fan [2]. In addition, we can find that the existing algorithm ([2]) performs well for $l = 2$ as its MBRFF allocations are nearly close to or the same as our optimal ones for $l = 2$.

3.4.2 Minimizing Total Leakage Power on Retention Storage

To demonstrate the efficacy of MBRFF-opt on minimizing total leakage power on the always-on retention storage, we set the covering segment costs to the (normalized) leakage powers on the retention storages in MBRFF implementations in Synopsys generic library. Table 3.5 shows the comparison of the leakage power consumed by Fan [2] and MBRFF-opt. Specifically, MBRFF-opt reduces the leakage power by 3.38% further for circuit WB_CONMAX when $l = 2$ and 5.58% further for USB_FUNCT for $l = 3$. In particular, for DES_PERF, MBRFF-opt saves the leakage power by 9.91% when $l = 3$, which clearly reveals that MBRFF-opt employing the KL based graph partitioning for large circuits works well.

3.4.3 Minimizing Total Area of Retention Storage

We set the covering segment costs to the (normalized) implementation area of the retention storage in the MBRFF implementations by [1]. Table 3.6 shows the comparison of retention storage area used by Fan [2] and MBRFF-opt, in which "#FFs (μm^2)" means total original flip-flop area. Specifically, MBRFF-opt reduces the area by 4.46% further for DES_PERF when $l = 3$.

Figs. 3.8(a) and (b) show the distribution of the MBRFFs in layouts of circuit MEM_CTRL produced by Fan [2] and MBRFF-opt for $l = 3$ with the objective of minimizing total bits of retention storage in Table 3.4. The comparison shows that MBRFF-opt reduces the number of 2-bit MBRFFs (yellow ones) from 77 to 28 at the expense of 4 more 3-bit MBRFFs (red ones). On the other hand, Fig. 3.8(c) shows the distribution of the MBRFFs in layouts of circuit MEM_CTRL produced by MBRFF-opt for $l = 3$ with the objective of minimizing total area in Table 3.6.

Chapter 4

Further Consideration

4.1 Multi-bit Flip-flops in Power Gated Circuits

For power-gated circuits, each flip-flop has its own state retention storage [43, 44, 6], commonly implemented with a high-Vt latch, to retain the flip-flop state during the sleep period of the circuits. Fig. 4.1 shows a typical flip-flop used in power gated circuits, in which signal RETAIN is set to high just before entering sleep mode so that the latch should retain the flip-flop state or just before waking up so that the flip-flop should restore the retained state from the latch. Since the state retention storage is non-trivial in terms of power and area, retention storage of non-uniform size is proposed to save the total bits of retention storages [45, 13, 46, 47, 6, 37]. For example, Figs. 4.2(a) and (b) show respectively, for the same flip-flop dependency graph of circuit, the retention storage allocation of a total of 3 bits with two clock cycles of wakeup latency and the naive allocation of a total of 4 bits, one for each flip-flop, with one cycle of latency. Consequently, in terms of power and cell area, the allocation in Fig. 4.2(a) is preferred. However, in view of chip implementation with multi-bit flip-flops, the inefficiency of the non-uniform storage allocation like that in Fig. 4.2(a) is outstanding: As illustrated with 4-bit MBFF cells in Figs. 4.2(c) and (d), the MBFF cell in Fig. 4.2(c) requires multiple RETAIN signals to drive save/restore to flip-flops f_1 and f_4 differently while

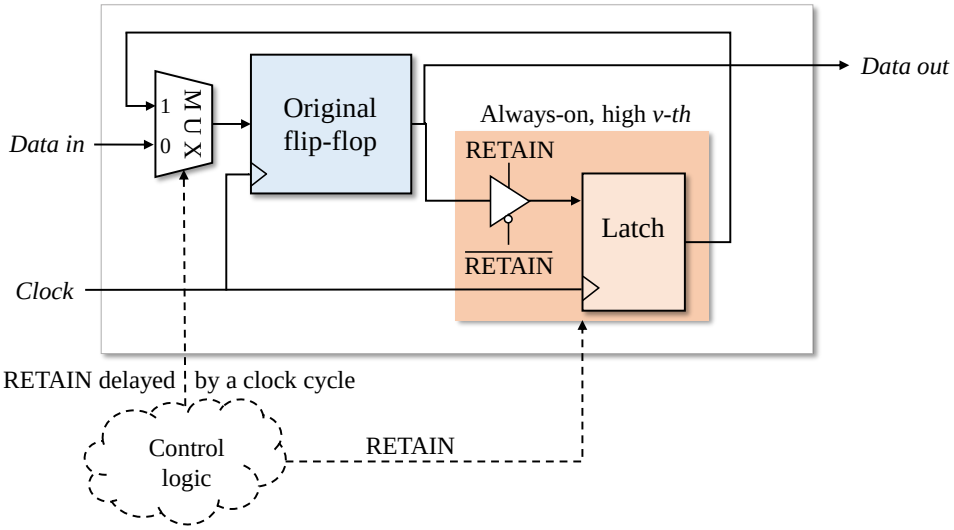


Figure 4.1: Retention flip-flop structure used in power gated circuit [6].

the cell in Fig. 4.2(d) needs just a single RETAIN signal. In addition, the regularity of the MBFF cell layout in Fig. 4.2(d) is superior to that in Fig. 4.2(c), even with no waste of layout space. To sum up, the decision on choosing one of the options of MBFF cells in power gated circuits should be made with a comprehensive consideration of the target design goals and constraints.

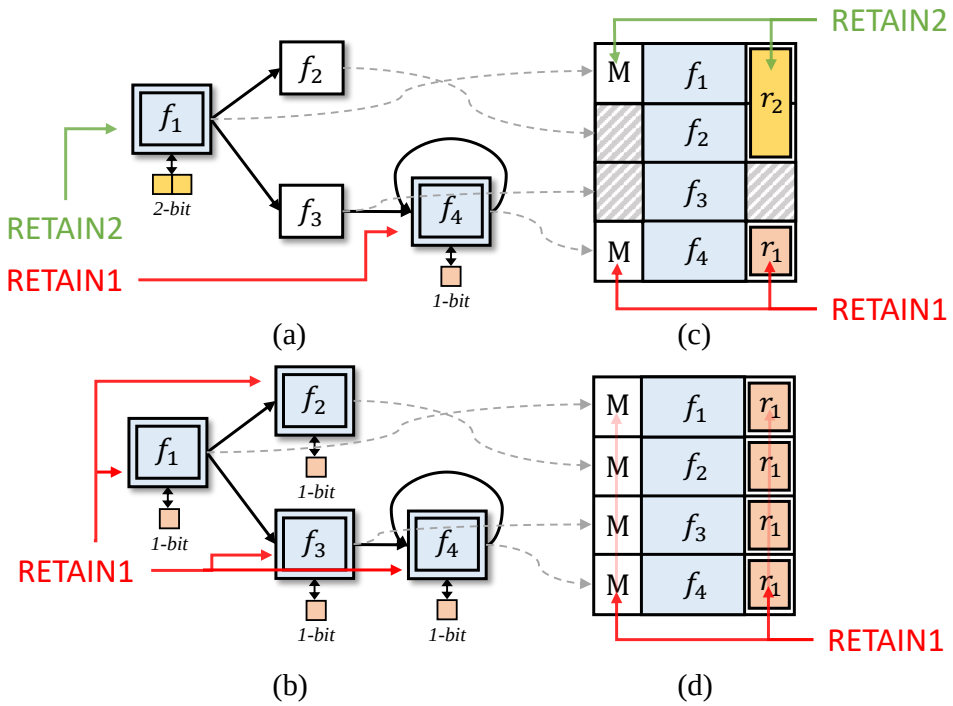


Figure 4.2: Illustration of state retention storage allocation on a flip-flop dependency graph. (a) Total of 3 bits with latency of two clock cycles using non-uniform retention storage. (b) Total of 4 bits with latency of one clock cycle using uniform retention storage. (c) 4-bit MBFF cell corresponding to the allocation in (a). (d) 4-bit MBFF cell corresponding to the allocation in (b).

Chapter 5

Conclusions

5.1 Chapter 2

In Chapter 2, we addressed two new inherent and challenging problems related to multi-bit flip-flop (MBFF) cells in physical design, which are (1) *non-flexible MBFF cell flipping* for multiple D-to-Q signals and (2) *unbalanced or wasted use of MBFF footprint space*. We tackled the two problems in a way to enhance chip routability and timing at the placement and routing stages. Specifically, for problem 1, we resolved the non-flexible MBFF cell flipping to be fully flexible by generating MBFF layouts supporting diverse D-to-Q flow directions in the detailed placement to improve routability while for problem 2, we enhanced the setup time and clock-to-Q delay on timing critical internal flip-flops in MBFF cell instances through transistor upsizing by utilizing the unused space in MBFFs to improve timing slack at the post-routing stage. Our experiments with benchmark circuits showed that our proposed DTCO flow optimizing MBFF cells amenable to the target circuit solved the two problems very effectively, producing chip implementations with 20.5% fewer design rule violations and 47.7% reduced worst negative timing slack with a little power fluctuation than that produced by the conventional design flow with MBFFs.

5.2 Chapter 3

In Chapter 3, we proposed an optimal MBRFF allocation algorithm by formulating the problem into a weighted unate covering problem, supporting three objectives of minimizing total bits of state retention storage, total leakage power consumed by the always-on retention storage, and total implementation area of the retention storage. According to design objectives, our algorithm could always guarantee optimal MBRFF allocations in terms of a weighted sum of total bits of storage, total leakage power, and total implementation area. Besides, it should be noted that our MBRFF allocation algorithm can seamlessly link to the allocation of multi-bit flip-flops (MBFFs) either before or after the step of MBFF allocation and also can seamlessly apply to the flip-flop dependency graph created using selected essential flip-flops from [30].

Bibliography

[1] Y.-G. Chen, H. Geng, K.-Y. Lai, Y. Shi, and S.-C. Chang, "Multibit retention registers for power gated designs: Concept, design, and deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 4, pp. 507–518, 2014.

[2] G.-G. Fan and M. P.-H. Lin, "State retention for power gated design with non-uniform multi-bit retention latches," in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2017, pp. 607–614.

[3] Y.-G. Chen, Y. Shi, K.-Y. Lai, G. Hui, and S.-C. Chang, "Efficient multiple-bit retention register assignment for power gated design: Concept and algorithms," in *Proceedings of the International Conference on Computer-Aided Design*, 2012, pp. 309–316.

[4] L. Clark, V. Vashishtha, L. Shifren, A. Gujja, S. Sinha, B. Cline, C. Ramamurthy, and G. Yeric, "Asap7: A 7-nm finfet predictive process design kit," *Microelectronics Journal*, vol. 53, pp. 105–115, 2016. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S002626921630026X

[5] C. Albrecht, "Iwls 2005 benchmarks," Tech. Rep., 2005.

[6] G. Hyun and T. Kim, "Allocation of multibit retention flip-flops for power gated circuits: Algorithm-design unified approach," *IEEE Transactions on Computer-*

Aided Design of Integrated Circuits and Systems, vol. 40, no. 5, pp. 892–903, 2021.

[7] Q. Wu, M. Pedram, and X. Wu, "Clock-gating and its application to low power design of sequential circuits," *IEEE Transactions on Circuits and Systems I: Fundamental Theory and Applications*, vol. 47, no. 3, pp. 415–420, 2000.

[8] T. Luo, D. Newmark, and D. Z. Pan, "Total power optimization combining placement, sizing and multi-vt through slack distribution management," in *2008 Asia and South Pacific Design Automation Conference*, 2008, pp. 352–357.

[9] G. Semeraro, G. Magklis, R. Balasubramonian, D. Albonesi, S. Dwarkadas, and M. Scott, "Energy-efficient processor design using multiple clock domains with dynamic voltage and frequency scaling," in *Proceedings Eighth International Symposium on High Performance Computer Architecture*, 2002, pp. 29–40.

[10] R. Pokala, R. Feretich, and R. McGuffin, "Physical synthesis for performance optimization," in *[1992] Proceedings. Fifth Annual IEEE International ASIC Conference and Exhibit*, 1992, pp. 34–37.

[11] Y.-T. Chang, C.-C. Hsu, M. P.-H. Lin, Y.-W. Tsai, and S.-F. Chen, "Post-placement power optimization with multi-bit flip-flops," in *2010 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2010, pp. 218–223.

[12] I. Jiang, C. Chang, and Y. Yang, "Integra: Fast multibit flip-flop clustering for clock power saving," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 31, no. 2, pp. 192–204, 2012.

[13] M. Lin, C. Hsu, and T. Chang, "Recent research in clock power saving with multi-bit flip-flops," in *2011 IEEE 54th International Midwest Symposium on Circuits and Systems (MWSCAS)*, 2011, pp. 1–4.

[14] Z. Chen and J. Yan, "Routability-constrained multi-bit flip-flop construction for clock power reduction," *Integration*, vol. 46, no. 3, pp. 290–300, 2013. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0167926012000181

[15] Y.-T. Shyu, J.-M. Lin, C.-P. Huang, C.-W. Lin, Y.-Z. Lin, and S.-J. Chang, "Effective and efficient approach for power reduction by using multi-bit flip-flops," *IEEE Transactions on Very Large Scale Integration (VLSI) Systems*, vol. 21, no. 4, pp. 624–635, 2013.

[16] C. Hsu, Y. Chen, and M. Lin, "In-placement clock-tree aware multi-bit flip-flop generation for power optimization," in *2013 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2013, pp. 592–598.

[17] C.-C. Tsai, Y. Shi, G. Luo, and I. H.-R. Jiang, "Ff-bond: Multi-bit flip-flop bonding at placement," in *Proceedings of the 2013 ACM International Symposium on Physical Design*, ser. ISPD '13. New York, NY, USA: Association for Computing Machinery, 2013, p. 147–153. [Online]. Available: https://doi.org/10.1145/2451916.2451955

[18] M. P.-H. Lin, C.-C. Hsu, and Y.-C. Chen, "Clock-tree aware multibit flip-flop generation during placement for power optimization," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 34, no. 2, pp. 280–292, 2015.

[19] M.-Y. Liu, Y.-C. Lai, W.-K. Mak, and T.-C. Wang, "Generation of mixed-driving multi-bit flip-flops for power optimization," in *2022 IEEE/ACM International Conference On Computer Aided Design (ICCAD)*, 2022, pp. 1–9.

[20] Y. Kretchmer and L. Logic, "Using multi-bit register inference to save area and power: the good, the bad, and the ugly," *EE Times Asia*, 2001.

[21] D. Yi and T. Kim, "Allocation of multi-bit flip-flops in logic synthesis for power optimization," in *2016 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2016, pp. 1–6.

[22] J.-T. Yan and Z.-W. Chen, "Construction of constrained multi-bit flip-flops for clock power reduction," in *The 2010 International Conference on Green Circuits and Systems*. IEEE, 2010, pp. 675–678.

[23] S.-H. Wang, Y.-Y. Liang, T.-Y. Kuo, and W.-K. Mak, "Power-driven flip-flop merging and relocation," in *Proceedings of the 2011 international symposium on Physical design*, 2011, pp. 107–114.

[24] I. Advanced Micro Devices. The polaris architecture. [Online]. Available: https://www.amd.com/en.html

[25] I. Seitanidis, G. Dimitrakopoulos, P. M. Mattheakis, L. Masse-Navette, and D. Chinnery, "Timing-driven and placement-aware multibit register composition," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 38, no. 8, pp. 1501–1514, 2019.

[26] C. Liu, T. Chiang, J. Kao, H. ZHUANG, L. Lu, S. Hsieh, and C. Huang, "Flip-flop with delineated layout for reduced footprint," May 2017, patent No. 9641161, Filed May 2nd., 2016, Issued May. 2nd., 2017.

[27] D. Flynn, R. Aitken, A. Gibbons, and K. Shi, *Low power methodology manual: for system-on-chip design*. Springer Science & Business Media, 2007.

[28] H. Mahmoodi-Meimand and K. Roy, "Data-retention flip-flops for power-down applications," in *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No. 04CH37512)*, vol. 2. IEEE, 2004, pp. II–677.

[29] A. Darbari, B. M. Al Hashimi, D. Flynn, and J. Biggs, "Selective state retention design using symbolic simulation," in *2009 Design, Automation & Test in Europe Conference & Exhibition*. IEEE, 2009, pp. 1644–1649.

[30] S. Greenberg, J. Rabinowicz, and E. Manor, "Selective state retention power gating based on formal verification," *IEEE Transactions on Circuits and Systems I: Regular Papers*, vol. 62, no. 3, pp. 807–815, 2014.

[31] M. A. Sheets, "Standby power management architecture for deep-submicron systems," Ph.D. dissertation, EECS Department, University of California, Berkeley, May 2006. [Online]. Available: http://www2.eecs.berkeley.edu/Pubs/TechRpts/2006/EECS-2006-70.html

[32] E. Pakbaznia and M. Pedram, "Design and application of multimodal power gating structures," in *2009 10th International Symposium on Quality Electronic Design*. IEEE, 2009, pp. 120–126.

[33] S.-H. Lin and M. P.-H. Lin, "More effective power-gated circuit optimization with multi-bit retention registers," in *2014 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2014, pp. 213–217.

[34] G. Hyun and T. Kim, "Allocation of state retention registers boosting practical applicability to power gated circuits," in *2019 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*. IEEE, 2019, pp. 1–6.

[35] S. Kim and T. Kim, "Design and technology co-optimization utilizing multi-bit flip-flop cells," in *Proceedings of the 41st IEEE/ACM International Conference on Computer-Aided Design*, 2022, pp. 1–7.

[36] ——, "Optimizing timing in placement through i/o signal flipping on multi-bit flip-flops," in *2022 IEEE International Symposium on Circuits and Systems (ISCAS)*, 2022, pp. 2623–2624.

[37] ——, “Minimally allocating always-on state retention storage for supporting power gating circuits,” in *2021 22nd International Symposium on Quality Electronic Design (ISQED)*, 2021, pp. 482–487.

[38] Gurobi Optimization, LLC, “Gurobi Optimizer Reference Manual,” 2022. [Online]. Available: https://www.gurobi.com

[39] C. Albrecht, “Iwls 2005 benchmarks,” in *International Workshop for Logic Synthesis (IWLS): http://www. iwls. org*, 2005.

[40] P. Ashar and S. Malik, “Implicit computation of minimum-cost feedback-vertex sets for partial scan and other applications,” in *Proceedings of the 31st annual Design Automation Conference*, 1994, pp. 77–80.

[41] G. D. Hachtel and F. Somenzi, *Logic synthesis and verification algorithms*. Springer Science & Business Media, 2006.

[42] I. I. Cplex, “V12. 1: User’s manual for cplex,” p. 157, 2009.

[43] D. Flynn, R. Aitken, A. Gibbons, and K. Shi, *Low power methodology manual: for system-on-chip design*. Springer Science & Business Media, 2007.

[44] H. Mahmoodi-Meimand and K. Roy, “Data-retention flip-flops for power-down applications,” in *2004 IEEE International Symposium on Circuits and Systems (IEEE Cat. No.04CH37512)*, vol. 2, 2004, pp. II–677.

[45] Y.-G. Chen, Y. Shi, K.-Y. Lai, G. Hui, and S.-C. Chang, “Efficient multiple-bit retention register assignment for power gated design: Concept and algorithms,” in *2012 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2012, pp. 309–316.

[46] G.-G. Fan and M. P.-H. Lin, “State retention for power gated design with non-uniform multi-bit retention latches,” in *2017 IEEE/ACM International Conference on Computer-Aided Design (ICCAD)*, 2017, pp. 607–614.

[47] Y.-G. Chen, H. Geng, K.-Y. Lai, Y. Shi, and S.-C. Chang, "Multibit retention registers for power gated designs: Concept, design, and deployment," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 33, no. 4, pp. 507–518, 2014.

초 록

고속 및 저전력 칩 구현의 물리적 설계에서는 다중 비트 플립플롭(multi-bit flip-flop)의 합성 및 상태 보존 저장소(state retention storage) 할당 문제가 중요한 문제이다. 본 논문에서는 대상 디자인 설계의 성능과 전력 소비를 향상시킬 수 있는 두 가지의 저장소 합성 및 할당 방법론을 제안한다.

먼저, 본 논문에서는 스탠다드 셀 배치 및 라우팅 단계에서 라우팅 가능성과 타이밍을 향상시키기 위해 다중 비트 플립플롭 셀을 활용한 설계 및 기술 공동 최적화 플로우를 제안한다. 구체적으로 다양한 D-to-Q 플로우 방향을 지원하는 다중 비트 플립플롭 셀 레이아웃을 생성함으로써 비융통성 있는 다중 비트 플립플롭 셀의 뒤집기를 완전히 유연하게 만들고, 다중 비트 플립플롭 내의 미사용 공간을 이용하여 다중 비트 플립플롭의 타이밍 크리티컬한 플립플롭의 셋업시간(setup time) 및 클럭-큐(Q) 딜레이(clock-to-Q delay)를 향상시킨다. 최신 공정을 적용한 벤치마크 회로에 대한 실험을 통해, 제안된 다중 비트 플립플롭을 사용한 설계 및 기술 공동 최적화 플로우가 대상 디자인의 라우팅 가능성과 타이밍 여유를 크게 향상시킬 수 있음을 보인다.

두번째로, 본 논문에서는 전력 게이팅이 적용된 디자인에서 최적으로 상태 보존 저장소를 할당하는 알고리즘을 제안한다. 구체적으로, 우리는 깨움 대기 시간 제약 조건(wakeup latency constraint)을 2와 3 클록 주기로 제한하는 할당 문제를 단방향 커버링 문제로 변환하고, 총 보존 저장소 비트를 최소화하는 것, 보존 저장소에 의해 직접 소비되는 총 누설 전력을 최소화하는 것, 그리고 보존 저장소의 총 구현 영역을 최소화하는 것과 같은 세 가지 목적 옵션을 사용한 최적 할당 알고리즘을 제안한다. 28나노 공정을 적용한 벤치마크 회로에 대한 실험을 통해, 제안된 보존 저장소의

할당 알고리즘이 가장 최근에 제안된 휴리스틱한 방법으로 보존 저장소를 할당하는 알고리즘에 비해 깨움 대기 시간 제약 조건을 3으로 설정할 때 보존 저장소의 총 비트, 총 누설 전력, 그리고 총 구현 영역을 더 감소시킬 수 있음을 보인다.

주요어: 다중 비트 플립플롭 합성, 설계 및 기술 공동 최적화, 트랜지스터 크기 조정, 보존 저장소 할당, 전력 게이팅, 누설 전력

학번: 2018-20366