Synthesis of Multi-Bit Flip-Flops for Clock Power Reduction

An-Chi CHANG and Ting-Ting HWANG*

Department of Computer Science, National Tsing Hua University, Hsinchu, Taiwan 30013

Received May 18, 2012; final version accepted September 14, 2012

Power optimization has always been an important issue for modern IC design. In this paper, we present a power optimization technique for clock tree by applying multi-bit flip-flops and reducing total wire length. Through merging flip-flops into MBFFs, we effectively reduce power consumption caused by clock buffers. Moreover, by judiciously merging and placing the MBFFs, the total wire length is also significantly reduced. The combined effect of both techniques leads to a strong reduction in total power consumption of the clock network.

KEYWORDS: multi-bit flip-flop, power consumption, clock network

1. Introduction

In modern VLSI design, power has become a critical issue. With limited power/thermal budget, as well as the increasing demand of reducing power dissipation, minimizing power consumption has become one of the most important objectives.

Power consumption of an IC chip can be categorized into two types: dynamic and static power consumption. Clock tree is one of the major causes of both types of power dissipation; it may consume as much as 40% of the total power [1] of the IC due to its frequent switching activity; clock tree is also the major consumers of leakage power because of the large number of buffers it contains. A lot of work related to reducing power consumption of clock tree has been proposed. Some address this problem when constructing clock network, reducing power consumption by planning a suitable topology and inserting buffer wisely [11]. Creating multiple supply voltage [3] is another approach. Donno et al. [9] and Mahmoodi et al. [8] use clock gating to resolve this problem. Energy recovery is also a feasible approach adopted in [8]. Lu et al. [6] and Lou et al. [10] focus on minimizing clock networks through replacing non-timing-critical cells with their high Vt counter parts.

Hou [5], Kretchmer [7], and Chang [2] took another direction: applying multi-bit flip-flop (MBFF), or register banks. MBFF is one of the most effective methodologies in saving both chip area and power consumption. Hou [5] proposed an incremental clock tree placement flow applying MBFF. Kretchmer [7] introduced a design methodology to create the models of multi-bit registers in the cell library to be inferred by logic synthesis tools. Chang [2] incrementally applied MBFFs at post-placement stage.

Figure 1 shows an example of merging two 1-bit flip-flops into one 2-bit flip-flop. Originally each flip-flop requires two inverters to generate clock signal respectively. However, due to the manufacturing ground rules, inverters in flip-flops tend to be oversized; as the process technology advances into smaller geometry nodes like 65 nm and beyond, even the minimum-sized inverter-chain can still drive more than one flip-flop. We can merge multiple flip-flops into a multi-bit flip-flop; through sharing of the clock signal, the number of inverters required, as well as the power consumption and area occupied can be significantly reduced; Table 1 shows comparisons of power consumption and area of flip-flops with different bit numbers. Also, through choosing proper clusters and placement location, the wire length can also be reduced.

We address the problem of applying multi-bit flip-flops when performing synthesis of clock network. We propose a novel power optimization method of clock tree by applying MBFF with a clique-based approach. The cluster problem of flip-flops as formulated into clique-finding and maximum independent-set problems. Through iterative window-based optimization, the flip-flops are gradually clustered into MBFFs, reducing the total power consumption and area of the clock tree; the total wire length is also taken into consideration to avoid additional driving load and further reduce power consumption caused by long metal wires.

The remaining chapters of this paper is organized as follows: Chapter 2 gives the motivation of our approach; the weakness of previous work is analyzed. Chapter 3 describes the problem formulation. Chapter 4 details our proposed algorithm. Chapter 5 presents the experimental results. Chapter 6 gives the conclusion of this paper.

* Corresponding author. E-mail: u942019@oz.nthu.edu.tw; tingting@cs.nthu.edu.tw
2. Previous Work and Motivation

In Chang et al.’s work [2], they proposed an algorithm to reduce the power consumption of clock tree by replacing flip-flops into multi-bit flip-flops. To realize this process, each time a window is selected. For flip-flops within the window, a set of cliques consisting of these flip-flops is computed, where each clique denotes a cluster combination that satisfies the given constraints. To solve the conflict between the cliques, a greedy selection method is adopted to generate independent set of the cliques. The priority to select the clique is evaluated in terms of power consumption and wire length overhead. The algorithm repeats the above procedure until the whole chip is processed. This flow is capable of effectively merging flip-flops into multi-bit flip-flops. However, there are three key factors of this algorithm left to be discussed.

First is the size of the selected window. In Chang’s work, the window size is fixed to two values. We found it is better to adjust it dynamically. A simple example of our reasoning is shown in Fig. 2. Initially the distribution of flip-flops is relatively dense; a smaller window size suits [Fig. 2(a)]. However, as the clustering process proceeds, the distribution of flip-flops becomes sparser. A larger window size is required in order to reach solutions involving clustering distant flip-flops [Fig. 2(b)]. We proposed a mechanism to dynamically adjust the size of the selected window to reflect the distribution of flip-flops.

The second observation is that in Changs work, once a flip-flop is merged into a multi-bit flip-flop, other solutions involving this said flip-flop being merged is not considered anymore. As shown in Fig. 3, Fig. 3(a) is the original flip-flops. If the cluster state of Fig. 3(b) is reached first, the solution of clustering the flip-flops as Fig. 3(c) will not be considered anymore. In order to rectify this deficient, we proposed disruptive collection, which is to decompose the merged multi-bit flip-flops so that a broader solution space can be explored.

Last is the process of generating clique set that corresponds to cluster combinations satisfying the given constraints. An accurate and fast wire estimation for cliques is proposed to identify potential long metal wires. A pruning mechanism using this method to eliminate inferior cluster combinations is adopted in our synthesis flow. Benchmarking results show that by applying this estimation mechanism, we significantly reduce the total wire length for merged flip-flops and the resulting power consumption.
3. Problem Formulation

This chapter is to describe the problem formulation in detail. The section is organized as follows: Section 3.1 gives the detail and definition of input data; Section 3.2 formally defines the objective functions; Section 3.3 specifies three major constraints of this problem.

3.1 Input

This problem has the following inputs:
- The width $W_c$ and height $H_c$ of the chip $C$.
- The width $W_g$ and height $H_g$ of placement grid. Placement grid is the basic resolution unit; pins and flip-flops must be placed on grids. There can be only one pin or one flip-flop on each placement grid.
- A set of pre-placed flip-flops $F$. Each flip-flop $f_i \in F$ is with its coordinate $(x_{f_i}, y_{f_i})$, and can be either 1-bit or multi-bit.
- A cell library $L$ containing a set of flip-flop cells. For an $m$-bit flip-flop $f_i^m$ in $L$, the power consumption $PC_i^m$ of $f_i^m$, and the area of $f_i^m$ $A_i^m$ are specified.
- A set of pins $P$. Each pin $p_i \in P$ has its coordinate $(x_{p_i}, y_{p_i})$, which can be either an input pin or an output pin.
- A set of net $N$ describing the connections between $F$ and $P$. For each $m$-bit flip-flop $f_i^m \in F$, there are total $2m$ nets, where $m$ nets connect to $m$ input pins of $f_i^m$ and $m$ nets connecting $m$ output pins of $f_i^m$. For a net $n_i(p_i, f_j)$ connect to a pin $p_i \in P$ and a flip-flop $f_j \in F$, there is a specified slack $S_{ij}$, where $S_{ij} \geq 0$.
- A set of bins $B$ with given width $W_b$ and height $H_b$. The chip $C$ is covered by $b_i \in B$.
- A set of pre-placed logic blocks $K$. Each $k_i \in K$ is with its own coordinate $(x_{k_i}, y_{k_i})$, and the area of $k_i$, $A_{ki}$. The maximum placement density $D_{max}$.

3.2 Objective function

The Synthesis of Multi-bit Flip-flops for Clock Power Reduction Problem is to minimize the total power consumption of all flip-flops $f_i \in F$ by replacing flip-flops with MBFFs specified in the given cell library $L$, as well as the total wire length of every net $n_i(p_i, f_j) \in N$.

The total power consumption of all flip-flops $f_i \in F$ is calculated by aggregating the power consumption $PC_i$ of each flip-flop $f_i \in F$.

$$PC_F = \sum PC_i, \forall f_i \in F.$$  \hfill (1)

The total wire length of $N$ is the aggregation of the wire length of every net $n_i \in N$.

$$L_N = \sum L_{n_i}, \forall n_i \in N.$$  \hfill (2)

The wire length $L_{n_i}$ of net $n_i(p_i, f_j)$ is defined as the Manhattan distance between $p_i$ and flip-flop $f_j$. Let $(x_{p_i}, y_{p_i})$ be the coordinate of pin $p_i$, and $(x_{f_j}, y_{f_j})$ be the coordinate of flip-flop $f_j$,

$$L_{n_i} = |x_{p_i} - x_{f_j}| + |y_{p_i} - y_{f_j}|.$$  \hfill (3)

3.3 Constraint

In this optimization problem, there are three constraints: Non-overlap constraint, placement-density constraint, and timing slack constraint. We discuss them in details in the following subsections.

3.3.1 Non-overlap constraint

Placement grid is the basic resolution unit; pins and flip-flops must be placed on grids. Each placement grid can only be occupied by either one pin or one flip-flop. For a chip $C$ with width $W_c$ and height $H_c$, there are total $(W_c/W_g) \times (H_c/H_g)$ placement grids. When generating multi-bit flip-flops, one must avoid placing the new flip-flop onto an occupied grid.

The details of our algorithm will be explained in Chap. 4.
Also note that the coordinate of pins cannot be changed. However, the flip-flops in original design can be re-placed in order to optimize the total wire length.

3.3.2 Placement density constraint

The total area consumption $D_i$ of a bin $b_i$ is the aggregation of the area of all flip-flops and pre-placed logic blocks within $b_i$. In order to avoid routing congestion, there is a placement density constraint $D_{max}$ to all bins. When a new flip-flop is generated, it can only be placed into a bin where the area consumption of the bin after adding up the area of the new flip-flop does not exceed $D_{max}$ as stipulated.

An example is shown in Fig. 4. Each square denotes a bin. The number on each bin is the total area of pre-placed logic blocks and flip-flops within the bin. Let $D_{max}$ equal to 10. In this example, the grey bins in Fig. 4 violate the placement density constraint since the total area of those bins exceed $D_{max}$, while the white bins satisfy it.

To further define this constraint: Let $A_{K_{bi}}$ be the total area of pre-placed logic blocks within bin $b_i$, and $A_{F_{bi}}$ be the total area of flip-flops in $b_i$.

$$A_{K_{bi}} + A_{F_{bi}} \leq D_{max}, \forall b_i \in B.$$  

(4)

3.3.3 Timing slack constraint

For each $n_{ij}(p_i, f_j) \in N$, there is a slack $S_{ij}$ given in the input file for the net. $S_{ij}$ is the additional driving load that pin $p_i$ could afford. The timing slack constraint demands the slack of every net remains larger than or equal to zero. This constraint may be violated when reposition of flip-flops. If a flip-flop $f_j$ of a net $n_{ij}(p_i, f_j)$ is relocated, the slack of the net will increase/decrease according to additional distance between the pin and flip-flop. When merging several flip-flops into a multi-bit flip-flop, the location of the newly generated flip-flop must be chosen wisely, so that the additional distance would not lead to negative slacks on any connected nets.

Figure 5 is an example of invalid slack. There are two 1-bit flip-flops, $f_1$ and $f_2$; $f_1$ is connected to $p_1$ and $p_2$ by net $n_{11}$ and $n_{21}$ respectively; $f_2$ is connected to $p_1$ and $p_4$ by net $n_{32}$ and $n_{42}$. $f_1$ and $f_2$ are to be merged into a 2-bit flip-flop $f_3$, which connects to all four pins, $p_1$, $p_2$, $p_3$ and $p_4$. As shown in Fig. 5(b), due to relocating and merging $f_1$ and $f_2$, the additional wire length and the resulting driving load lead to a negative slack on $n_{13}(p_1, f_3)$ and $n_{33}(p_3, f_3)$, violating the timing slack constraint.

The timing slack constraint can be formulated as follows:

Let $T_{ij\text{MAX}}$ be the maximum timing tolerance of net $n_{ij}(p_i, f_j)$, where $f_j$ is the original locations of flip-flop sink specified in the golden input.

![Fig. 4. An example of density constraint.](image)

![Fig. 5. After merging $f_1$ and $f_2$ into $f_3$, the additional wire length caused timing violation on net $n_{13}$ and $n_{33}$.](image)
After repositioning flip-flops or merging them into MBFFs, for every net $n_{ij}(p_i, f_j) \in N$ on the chip, the corresponding $T_{ij}$ must remain equal to or less than its $T_{ij}^{\text{MAX}}$.

$$T_{ij} = |x_{p_i} - x_{f_j}| + |y_{p_i} - y_{f_j}| + S_{ij}. \quad (5)$$

The golden input design should also meet all aforementioned constraints before performing power/wire length optimization.

## 4. Synthesis of MBFF

Based on the aforementioned problem descriptions, we proposed a window-based cluster algorithm, which iteratively selects a window from the chip and merges the flip-flops within the window into multi-bit flip-flops. Our optimization process is divided into four phases, as shown in Fig. 6. Each phase performs a variation of our window-based cluster algorithm. The first phase, Regular Non-Disruptive Clustering (RNDC) generates an initial solution; the second phase, Dynamic Non-Disruptive Clustering (DNDC) and the third phase, Dynamic Disruptive Clustering (DDC) further refine the result; the last phase, Corner Case Refinement (CCR) focuses on enhancing corner cases. The detailed differences and transitions of these four phases will be explained in the later sections of this chapter.

A detailed pseudo code of our window-based optimization is shown in Fig. 7. In the first stage, a window $W$ is selected. In the second stage, we collect $F_{\text{local}}$, the set of flip-flops within $W$, and based on $F_{\text{local}}$ we compute $F_{\text{target}}$, which is the target set for our algorithm to generate MBFFs. In the third stage, a clique set $C$ for $F_{\text{target}}$ is computed, and each $c_i^m \in C$ is a legal cluster corresponding to a cell type $f_j^m \in L$. The algorithm greedily selects a $c_i^m \in C$ with the lowest cost to generate its corresponding multi-bit flip-flop. All cliques conflicting with the selected clique are then removed from $C$. The process is repeated until $C$ becomes an empty set.

There are two key factors of our window-based clustering algorithm. First is the size of the selected window $W$. Second is the mechanism to compute $F_{\text{target}}$. Instead of adopting all flip-flops in $F_{\text{local}}$ directly as in [2], we introduce disruptive collection, optionally decomposing flip-flops in $F_{\text{local}}$ before merging them. The details will be discussed in the later sections.

The remaining sections of this chapter are organized as follows: Section 4.1 renders the transitions between the four phases of our optimization process; Section 4.2 focuses on deciding the size of selected window. Section 4.3 gives details of disruptive collection, our proposed mechanisms to compute target set of flip-flops to merge. Section 4.4 explains the clique-based merging process. The generation of cliques and pruning of solution space will also be discussed.

### 4.1 Phase transition

#### 4.1.1 RNDC

In the first phase of our optimization process, Regular Non-Disruptive Clustering (RNDC), the chip is equally divided into windows with same size. This phase ends after clustering of all windows is completed.

#### 4.1.2 DNDC and DDC

For the second and the third phases (DNDC and DDC), these two phases end when the solution converges. The
definition of convergence is as follows: Each time a window is selected and the flip-flops within this window is clustered is called a round. After completing a round, the reduction of total power consumption is computed. Due to the nature of our algorithm, the gain is guaranteed to be either equal to or larger than zero. If the number of rounds with zero gain consecutively is larger than a threshold $T$, our optimization process will end current phase and continue to next phase.

Threshold $T$ is defined as follows:

$$T = \frac{\text{# of FFs in input}}{\text{# of bits of MBFF in L with smallest bit number}},$$

where $L$ is the library of MBFFs given in the input.

Ideally during each round the process should be able to merge several flip-flops into one MBFF with the minimum bit number. We use the ratio of the original number of flip-flops in the input to the number of bits of the smallest MBFF as the threshold to indicate time to end current phase. For $DNDC$, $\alpha$ is set as 1; for $DDC$, in order to pursuit the quality of solution more persistently, $\alpha$ is set to 5 to allow more rounds of attempts to search for better cluster.

### 4.1.3 CCR

The last phase of our optimization process, Corner Case Refinement focuses on enhancing flip-flops with the highest 20% power consumption. Each round one of those flip-flops is used as the center of the selected window to perform our window-based clustering. This phase ends after all the aforementioned flip-flops are processed.

### 4.2 Select window

We apply a window-based approach is to reduce the problem size. Through processing only a window instead of the whole chip at a time, the original problem is divided into smaller ones to be conquered.

In Chang et al.’s work [2], the size of the selected window is fixed as either $2 \times 2$ or $4 \times 4$ bins. We observe that the window size should relate to the specific volume of the chip instead of bins. Moreover the window size should be able to adjust more freely. Our algorithm adopts fixed and dynamic window size according to different stages of clustering. Here we introduce the specific value $\sigma$ to control the window size.

$$\sigma = \frac{\text{Width} \times \text{Height}}{\text{number of FFs}}.$$  

The details of our mechanism will be discussed in the following subsections.
4.2.1 Fixed window size

When computing an initial solution of our algorithm in RNDC phase, a fixed window size is preferable. In consideration of run time and the relatively dense nature of the initial input, we use a fixed specific value \( \sigma_{\text{fixed}} \) in this stage.

\( \sigma_{\text{fixed}} \) is set as:

\[
\sigma_{\text{fixed}} = \sigma_{\text{golden}} = \frac{W_c \times H_c}{\text{number of FFs of golden input}},
\]

where \( W_c \) and \( H_c \) are the width and height of the chip respectively.

Since the number of flip-flops in the input, as well as \( W_c \) and \( H_c \) are all constants, \( \sigma_{\text{fixed}} \) is also constant. The chip is divided into identical windows with the same width and height (\( \sigma_{\text{fixed}} \)) to be processed.

4.2.2 Dynamic window size

After the initial solution is computed, in order to retrieve a further refined solution, the algorithm should allow distant flip-flops being merged together. A larger window size is thus required. It is dynamically adjusted according to \( \sigma_{\text{dynamic}} \):

\[
\sigma_{\text{dynamic}} = \frac{W_c \times H_c}{\text{current number of FFs on the chip}},
\]

where \( W_c \) and \( H_c \) are the width and height of the chip respectively and are both constant. As the clustering process proceeds, the number of FFs on the chip decreases, leading to a larger \( \sigma_{\text{dynamic}} \). This trend fits our requirement of a growing window size.

In DNDC and DDC phases, the exact coordinates of the selected window are randomly decided, with the size computed as \( \sigma_{\text{dynamic}} \).

In CCR phase, we focus on clustering poorly merged flip-flops. The flip-flops with the highest 20% power consumption are to be re-processed. The window size in this stage is computed in same fashion as \( \sigma_{\text{dynamic}} \). However, the flip-flops to be processed are set as the center of the selected windows.

4.3 Compute target FF set

In Chang et al.’s work [2], once a flip-flop is merge into a multi-bit flip-flop, other solutions involving this said flip-flop being merged is not considered anymore. In order to overcome this weakness, when computing \( F_{\text{target}} \), the set of flip-flops to merge into MBFFs, we propose the option of disruptive collection, which is to decompose the merged flip-flops into its component flip-flops and re-cluster these component flip-flops instead of directly adopting the existing \( F_{\text{local}} \) (non-disruptive collection). The details of both mechanisms will be elucidated in the following subsections.

4.3.1 Non-disruptive collection

In our algorithm, each time for a window \( W \) selected to be processed, \( F_{\text{local}} \) is the set of flip-flops whose coordinated are within \( W \) and \( F_{\text{target}} \) is the set of flip-flops to be merged. If \( F_{\text{target}} \) is computed in the fashion of directly adopting \( F_{\text{local}} \), it is called non-disruptive collection. The advantage of this method is that it is easy for computation with fast run time. However, as mentioned before, this mechanism lacks the ability to reach for a broader range of solution space. We utilize this approach in RNDC phase to compute initial solution, generating MBFFs with fast run time and fair quality.

4.3.2 Disruptive collection

After initial solution is computed, in order to search the solution space more thoroughly, we propose disruptive collection, which decomposes the merged flip-flops and re-cluster.

As shown in Fig. 8, each newly merged multi-bit flip-flop is constructed by several component flip-flops. For a merged 4-bit flip-flop \( f^4 \), it is composed of three flip-flops: two 1-bit flip-flops \( f^1 \) and \( f^2 \), and one 2-bit flip-flop \( f^2 \).

![Fig. 8. Possible composition of a merged MBFF: the 4-bit flip-flop \( f^4 \) is composed two 1-bit flip-flops \( f^1 \) and \( f^2 \), and a 2-bit flip-flop \( f^2 \).](image-url)
Fig. 9. Decomposition the 4-bit flip-flop \( f_1^4 \): (a) Original clustering. (b) Target set after disruptive collection. (c) A better solution.

while \( f_2^4 \) is composed of two 1-bit flip-flops, \( f_3^1 \) and \( f_1^1 \). \( f_1^1 \) can in fact be decomposed back into \( f_1^1 \), \( f_2^1 \), and \( f_3^1 \), or into \( f_1^1 \), \( f_2^1 \), and \( f_4^1 \).

In the example in Fig. 9, originally \( F_{\text{local}} \) has two 1-bit flip-flops, \( f_1^1 \) and \( f_2^1 \), and a 4-bit flip-flop \( f_4^4 \), where \( f_4^4 \) is composed of four 1-bit flip-flop \( f_1^1 \), \( f_2^1 \), \( f_3^1 \), and \( f_4^1 \), and \( f_1^4 \) and \( f_2^4 \) cannot be merged together due to violation of timing slack constraint. If we adopt non-disruptive collection, we cannot find any better solution. However, if disruptive collection is performed and \( f_4^4 \) is decomposed to four 1-bit flip-flops as shown in Fig. 9(b), re-clustering the six 1-bit flip-flop can potentially lead to a better clustering with one 2-bit flip-flop and one 4-bit flip-flop \( f_2^4 \) as shown in Fig. 9(c).

The procedure to decompose a given flip-flops is shown in Algorithm 1. \( F \) is the global set of flip-flops of the clustering result. \( F_{\text{target}} \) is the set of target flip-flops to be merged, \( f_i \) is the flip-flops to be decomposed, and \( f_i \) is recursively broken down into its component flip-flops.

**Algorithm 1**

```java
def decompose_ff(F, F_target, f_i):
    if f_i is not basic FF then
        for all f_i's component FF f_j do
            decompose_ff(F, F_target, f_j)
        end for
        F ← F − f_i
    end if
    F_target < −F_target + f_i
```

In our algorithm, disruptive collection is adopted in DNDC, DDC, and CCR phases to compute \( F_{\text{target}} \). If the quality of the solution in measure of power consumption deteriorates after performing disruptive collection and re-clustering, our algorithm would restore the cluster status back to before decomposing and re-clustering.

4.4 Generating MBFF

Once a window is selected and the set of flip-flop \( F_{\text{target}} \) within the window is obtained, the clustering of \( F_{\text{target}} \) is performed. We propose a clique-based algorithm to decide how to cluster flip-flops in \( F_{\text{target}} \) into multi-bit flip-flops, while satisfying the constraints in Chap. 3. §3.3.

4.4.1 Valid timing slack region (VTSR) computation

In order to meet the timing slack constraint, all flip-flops must be placed on a grid on which all nets connecting to the flip-flop are with a slack larger than or equal to zero. For a flip-flop \( f_i \), let \( P_i \) be the set of pins connected to \( f_i \) by a set of nets \( N_i \). Let \( T_{\text{MAX}} \) be the maximum timing tolerance for net \( n_{\text{MAX}}(p_k, f_i) \in N_i \), \( p_k \in P_i \), as defined in eq. (5). Let VTSR of pin \( p_k \) be the set of grids with the Manhattan distance from the grid to \( p_k \) less than or equal to the \( T_{\text{MAX}} \). To satisfy timing slack constraint for all nets in \( N_i \), \( f_i \) must be placed within the intersection of VTSR of every pin in \( P_i \). The intersection of VTSR of every pin of a flip-flop \( f_i \) is defined as the valid timing slack region of \( f_i \), VTSR\(_{f_i} \). An example is shown in Fig. 10. \( f_i \) is an 1-bit flip-flop with two pins \( p_1 \) and \( p_2 \) connected. VTSR\(_{f_i} \) is the intersection of VTSR of \( p_1 \) and \( p_2 \).

\[
VTSR_{f_i} = \text{intersection of VTSR of every pin connecting to } f_i. \tag{11}
\]

Given two flip-flops \( f_i \) and \( f_j \), let \( P_i \) be the set of pins connected to \( f_i \) by net set \( N_i \), and \( P_j \) be the set of pins connected to \( f_j \) by net set \( N_j \). If \( f_i \) and \( f_j \) are to be merged into a new flip-flop \( f_k \), the timing slack on all nets in \( N_i \) and \( N_j \) must be satisfied after reconnecting to \( f_k \). VTSR\(_{f_i} \) is the intersection of VTSR of \( f_i \) and VTSR of \( f_j \), which are the intersection of VTSR of all pins in \( P_i \) and \( P_j \) respectively as eq. (12).

\[
VTSR_{f_i} = VTSR_{f_i} \cap VTSR_{f_j}. \tag{12}
\]

If \( f_i \) and \( f_j \) are to be merged into a new flip-flop \( f_k \) but VTSR\(_{f_k} \) is null, \( f_i \) and \( f_j \) cannot be merged together since the timing slack constraint cannot be met. Figure 11 shows an example. \( f_1 \) and \( f_2 \) are the flip-flops to be merged, and
VTSR\(_{f_1}\) and VTSR\(_{f_2}\) are the valid timing slack region of \(f_1\) and \(f_2\) respectively. VTSR\(_{f_1}\) and VTSR\(_{f_2}\) does not intersect; VTSR\(_{f_1}\cap f_2\) is a null set. \(f_1\) and \(f_2\) cannot be merged together since there is no grid satisfies the timing slack constraints for every net connecting to \(f_1\) and \(f_2\).

Let \(F\) be the set of flip-flops to cluster together, the VTSR for the merged multi-bit flip-flop is the intersection of VTSR for every \(f_i \in F\). For example, in Fig. 12, \(f_1\), \(f_2\), \(f_3\), and \(f_4\) are the intended flip-flops and VTSR\(_{f_1}\), VTSR\(_{f_2}\), VTSR\(_{f_2}\), and VTSR\(_{f_4}\) are their valid timing slack region respectively. The merged MBFF must be placed with the intersection of VTSR of all the four flip-flops, VTSR\(_{f_1}\cap f_2\cap f_3\cap f_4\).

### 4.4.2 Valid timing slack clique (VTSC) generation

A VTSR intersection graph is a non-directed graph \(G(V,E)\), where each vertex \(v_i \in V\) corresponds to a flip-flop \(f_i\) in the design. \(e_{ij}\) between vertex \(v_i\) and \(v_j\) exists if the intersection of VTSR\(_{f_i}\) \(\cap\) VTSR\(_{f_j}\) \(\neq \emptyset\).

If a set of flip-flop is to be merged, as aforementioned, there must be a non-null intersection of the VTSRs of every flip-flop in the set, which means that there is an edge \(e_{ij}\) between every \(v_i, v_j \in V\). In other words, for all \(v_i\) corresponds to the set of flip-flops, they form a clique.
Let a window $W$ be selected, and the target set of flip-flop to merge $F_{\text{target}}$ is collected. To explore the cluster combination of $F_{\text{target}}$, the VTSR intersection graph of $F$ is first computed. Then we enumerate cliques with degree less than $m$ in the intersection graph, where there is a corresponding cell type in the input library $L$ with bit number $m$.

To enumerate cliques, the flip-flops in the input form the initial cliques in the clique list. Each time two cliques are tested to see whether their VTSR have non-null intersection; if yes, these two cliques form a new clique. This new clique will also be tested with other cliques to derive larger cliques. The process repeats until all combinations to form cliques corresponding to multi-bit flip-flop with the maximum bit number in the library is explored. An example is shown in Fig. 14. Figure 13(a) is the VTSR intersection graph of six flip-flops. Figure 13(b) shows the corresponding enumerated cliques.

However, if the above clique generation method is utilized straight forward, many redundant solutions with large wire length cost may be generated and consumes momentous computation time. A branch-and-bound method is applied to eliminate these undesired cliques and limit their number. If the estimated wire length of the clique and its corresponding flip-flop is too long, this clique is considered undesirable and be pruned. In Fig. 14(b), the grey cliques are cliques pruned based on above pruning mechanism. Cliques with too large wire length are considered undesirable and will not be used to derive more cliques. For instance, $c_{23}$ in Fig. 14(b) is pruned, along with its derivative $c_{234}$.

The pruning procedure is shown as Algorithm 2. In this algorithm, $AVWL$ is the average wire length of each pin in the golden input, computed as eq. (13).

![Fig. 13. (a) Intersection graph of six FFs. (b) Enumeration of corresponding cliques of (a).](image)

![Fig. 14. (a) Intersection graph of six FFs. (b) Enumeration with pruning of corresponding cliques of (a).](image)

**Algorithm 2 Prune Clique ($c_{j}^{m}, f_{j}^{m}$)**

```plaintext
if $\sigma_w < \sigma_{\text{golden}}$ then
    return false;
else if estimated wire length $EWL > \#\text{pins} \times AVWL$ then
    return true;
else
    return false;
end if
```

\[
AVWL = \frac{\sum_{n_i \in N} \text{wire length of } n_i}{\# \text{ of pins}}. \tag{13}
\]
To estimate the wire length of clustering flip-flops, EWL is defined. It is computed as follows. For an $m$-clique $c^m_i$, which corresponds to an $m$-bit flip-flop $f^m_i$, an estimation point $EP_{p_j}$ is the point within VTSR$_{f_j}$ used to estimate wire length for $f^m_i$. The estimated wire length (EWL) of $c^m_i$ and its corresponding flip-flop $f^m_i$ is the aggregation of Manhattan distance between $EP$ and every pin connected to $f^m_i$.

To compute the coordinate of the $EP$, first we need to compute three types of estimation points, inner point (IP), meso-point (MP), and outer point (OP). The estimation point of $f^m_i$ is the weighted center of its inner points, meso-points and outer points.

The shared estimation point for all pins inside VTSR, inner point $IP(x_{inner},y_{inner})$, is the center of those pins. The $x$-coordinate of inner point, $x_{inner}$ is computed as eq. (14).

$$x_{inner} = \frac{\sum_{(p_j \in P_j, p_j \text{ within VTSR})} y_{p_j}}{\# \text{ of component FF}},$$  

$w_{inner}$ is the weight of $x_{inner}$ and is computed as eq. (15).

$$w_{inner} = \sum_{(p_j \in P_j, p_j \text{ within VTSR})} |x_{inner} - x_{p_j}|,$$

$y_{inner}$ and its corresponding weight $w_{inner}$, are computed in similar fashion.

For pin $p_j$, which is outside VTSR$_f$ but within bounding box of VTSR$_{f_i}$, the estimation point for $p_j$, MP$_{p_j}$, is the point on the edge of VTSR$_{f_i}$ with the minimum Manhattan distance to $p_j$. meso-point can be obtained by solving linear equations. The weight of MP$_{p_j}$,$(x_{meso}, y_{meso})$, $w_{meso}$, is the distance between $p_j$ and MP$_{p_j}$.

For pin $p_j$, which is outside the bounding box of VTSR$_f$, the estimation point outer point is the corner of VTSR with the minimum Manhattan distance to $p_j$. The weight of OP$_{p_j}$,$(x_{outer}, y_{outer})$, $w_{outer}$, is the distance between $p_j$ and OP$_{p_j}$.

Figure 15 shows an example of estimation points. $f_1$ is a 2-bit flip-flop with four connected pins, $p_1$, $p_2$, $p_3$, and $p_4$. First, $p_1$ and $p_2$ are inside VTSR of $f_1$. The inner point for $p_1$ and $p_2$ is computed as eq. (14), as point IP marked on the Figure. Next, $p_3$ is outside of VTSR of $f_1$ but within bounding box of VTSR of $f_1$. Thus MP of $p_3$ is the point on the edge of VTSR$_{f_1}$ with the minimum Manhattan distance to $p_3$. Finally, $p_4$ is outside of the bounding box of VTSR$_{f_1}$. The estimation point of $p_4$, OP$_{p_4}$ is the corner of VTSR$_{f_1}$ closest to $p_4$.

The estimation point of $f^m_i$, $EP_{f^m_i}(x_{estimate}, y_{estimate})$ is the weighted center of inner points, meso-points and outer points of all pins connected to $f^m_i$.

$$x_{estimate} = \frac{x_{inner} \times w_{inner} + \sum (x_{meso} \times w_{meso}) + \sum (x_{outer} \times w_{outer})}{w_{inner} + \sum w_{meso} x + \sum w_{outer} x}$$  

$y_{estimate}$ is computed in similar fashion.

### 4.4.3 Clique selection

Let $C$ be the set of all VTSC of the flip-flop set $F$ explored by method in §4.4.2. To select VTSCs to merge into multi-bit flip-flops, we propose an intuitive greedy heuristic.

Let $R_i$ be the cost of an VTSC $c_i \in C$ corresponding to an multi-bit flip-flop $f_i$.

$$R_i = PC_{f_i} + EWL_{f_i}$$

(17)

Each time our algorithm picks the clique $c_{min}$ with the minimum cost to perform clustering; then every other clique with flip-flops in $c_{min}$ are removed from the set.

Fig. 15. An example of inner point (IP), meso-point (MP), and outer point (OP).
Algorithm 3: Selection of VTSC

Sort $C$ in descending order with respect to $R_{i}$ of $c_{i}$

while $C \neq \emptyset$ do

if there is a legal placement grid within $VTSR_{c_{\min}}$ then

Merge $f_{j}$ in $c_{\min}$ into MBFF $f_{c_{\min}}$

$F \leftarrow F - (f_{\text{in}c_{\min}}) + f_{c_{\min}}$

for all $c_{i} \in C$ do

if any $f_{j}$ in $c_{i}$ also in $c_{\min}$ then

$C \leftarrow C - c_{i}$

end if

end for

end if

end while

4.4.4 Decide location of MBFF

When a multi-bit flip-flop $f_{m}$ is generated, the coordinate of $f_{m}$ is decided by searching $VTSR$ of $f_{m}$. The grid within the $VTSR$ that satisfy the density constraint and with the minimum wire length to all pins connected is chosen.

5. Experimental Result

We implement our algorithm in C programming language under Linux operating system. We applied six industrial test cases to corroborate the quality of the solution of our algorithm. The number of flip-flops in these cases ranges from approximately 100 to 170000. The exact composition of the test cases is listed as Table 2. We adopt a library with 1-bit, 2-bit, and 4-bit, three kinds of flip-flops. The specification of the library is shown as Table 3.

Our experiments include two parts. First is to validate the effectiveness of our pruning mechanism and wire length estimation method. Second we compare the power reduction, wire length ratio, and runtime of our algorithm with Chang et al.’s work [2].

<table>
<thead>
<tr>
<th>Case</th>
<th># of 1-bit FFs</th>
<th># of 2-bit FFs</th>
<th># of 4-bit FFs</th>
<th>HPWL</th>
</tr>
</thead>
<tbody>
<tr>
<td>c1</td>
<td>76</td>
<td>22</td>
<td>0</td>
<td>89425</td>
</tr>
<tr>
<td>c2</td>
<td>366</td>
<td>57</td>
<td>0</td>
<td>60132</td>
</tr>
<tr>
<td>c3</td>
<td>1464</td>
<td>228</td>
<td>0</td>
<td>240528</td>
</tr>
<tr>
<td>c4</td>
<td>4378</td>
<td>751</td>
<td>0</td>
<td>772076</td>
</tr>
<tr>
<td>c5</td>
<td>9150</td>
<td>1425</td>
<td>0</td>
<td>1083300</td>
</tr>
<tr>
<td>c6</td>
<td>146400</td>
<td>22800</td>
<td>0</td>
<td>24052800</td>
</tr>
</tbody>
</table>

<table>
<thead>
<tr>
<th>Case</th>
<th>Area</th>
<th>Power</th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>1</td>
<td>172</td>
<td>100</td>
<td></td>
</tr>
<tr>
<td>2</td>
<td>192</td>
<td>172</td>
<td></td>
</tr>
<tr>
<td>4</td>
<td>285</td>
<td>312</td>
<td></td>
</tr>
</tbody>
</table>

5.1 Pruning and wire length estimation

In Chap. 4 we proposed an approach to estimate the wire length of a clique based on its corresponding implementation of multi-bit flip-flop, and used this information to prune inferior cliques. Table 4 shows the number of cliques enumerated during computing each case with and without applying our pruning mechanism. The result demonstrated the effectiveness of our pruning mechanism to reduce the computation time. Table 5 further shows the average wire length estimation error percentage of each test case. Among all cases the maximum average estimation error is less than five percent, manifesting the accuracy of our estimation mechanism, which is capable of predicting the wire length without actually searching every placement grid in $VTSC$ of the clique.

Figure 16 visualizes the estimated wire length and the actual minimum wire length for all cliques generated during computing test case $c1$. The blue line is the estimated wire length, and the red line is the actual minimum wire length.
for the clique. We can see that the two lines are almost identical in their trends, providing demonstration of the accuracy of our estimation.

5.2 Power, wire length and run time

Table 6 shows the results of our clustering algorithm, and Table 7 shows the comparison between Chang et al.’s work [2] and ours based on the given cell library. Our work has competitive, if not slightly finer performance in power consumption of flip-flops, as well as significant improvement in wire length. Our algorithm takes longer time to compute due to a more thorough search in solution space, but the run time is still acceptable in every cases. Even in the largest case with around 170000 flip-flops, our program still takes less than five minutes.

Compared with Chang’s work, our work addresses the importance of wire length reduction more. As the manufacturing technology evolves, the ratio of power consumption caused by metal wires becomes more and more
significant among the whole chip. In addition to power reduction through merging flip-flops, we further model the benefits in power due to reduced wire length into consideration.

Under 65 nm technology node, assume the grid size is equal to the minimum distance between two flip-flops, which is 0.56 micron. The power ratio of an 1-bit flip-flop to 1-micron of metal wire under 65 nm technology node is 250:1, thus the normalized power of 1-bit flip-flop to one unit-length (one grid) of metal wire is 140. We estimate the power consumption of 2-bit and 4-bit flip-flops based on the ratio of their area. Table 8 shows the normalized power of each cell type.

Based on above normalized cell power, we compute the combined power consumption of both flip-flops and metal wires before and after our clustering process. The impact of reduced wire length is shown in Table 9.

6. Conclusion

In this paper, we introduced a problem formulation to synthesize multi-bit flip-flop to optimize the power consumption of clock tree, as well as the wire length consumed. We proposed a window-based algorithm, in which a clique-based clustering is performed. Experimental results based on industrial cases show the effectiveness of our algorithm in merging flip-flops and decreasing wire length. In addition to the benefits of applying multi-bit flip-flops, comparing with previous work, our algorithm has stronger impact on total power dissipation since we are capable of reducing the wire length more significantly.

REFERENCES


