INTEGRA: Fast Multibit Flip-Flop Clustering for Clock Power Saving

Iris Hui-Ru Jiang, *Member, IEEE,* Chih-Long Chang, and Yu-Ming Yang

***Abstract*—Clock power is the major contributor to dynamic power for modern integrated circuit design. A conventional single-bit flip-flop cell uses an inverter chain with a high drive strength to drive the clock signal. Clustering several such cells and forming a multibit flip-flop can share the drive strength, dynamic power, and area of the inverter chain, and can even save the clock network power and facilitate the skew control. Hence, in this paper, we focus on postplacement multibit flip-flop clustering to gain these benefits. Utilizing the properties of Manhattan distance and coordinate transformation, we model the problem instance by two interval graphs and use a pair of linear-sized sequences as our representation. Without enumerating all possible combinations, we identify only partial sequences that are necessary to cluster flip-flops, thus leading to an efficient clustering scheme. Moreover, our fast coordinate transformation also makes the execution of our algorithm very efficient. The experiments are conducted on industrial circuits. Our results show that concise representation delivers superior efficiency and effectiveness. Even under timing and placement density constraints, clock power saving via multibit flip-flop clustering can still be substantial at postplacement.**

***Index Terms*—Clock power, coordinate transformation, interval graph, multibit flip-flops, postplacement optimization.**

I. Introduction

POWER HAS BECOME one of the main circuit implementation bottlenecks for modern integrated circuit design. In particular, high power consumption may prevent a high-speed design from running at its full speed, while low power dissipation is a must for consumer and portable electronic products. Moreover, since the clock signal toggles in each cycle, the total power dissipation in the clock network could be significant. Due to the high switching activity, clock power is the major dynamic power source [1]

$$P_{clk} = C_{clk} V_{dd}^2 f_{clk} \quad (1)$$

where P_{clk} is clock power, f_{clk} is the clock frequency, V_{dd} is the supply voltage, and C_{clk} is the switching capacitance including the gate capacitance of flip-flops (sequential elements) controlled by the clock signal, the interconnect capacitance of the clock network, and the capacitance associated with the buffers/inverters used in the clock network.

To minimize the switching capacitance, one well-known technique is clock gating, stopping the clock for some flip-flops when it is not required [2], [3]. However, the power saving of clock gating heavily depends on logic functions. In this paper, we consider a new type flip-flop cell—multibit flip-flop (MBFF) [4], [5]. Fig. 1(a) illustrates the circuit structure of a single-bit flip-flop, composed of two chained inverters and two cascaded latches. Due to the DFM rules for advanced technology, the inverter chain is oversized and has a high drive strength to shorten the delay from the clock edge to data output. Consequently, it can drive more than two cascaded latches. As shown in Fig. 1(b), clustering several single-bit flip-flops together can share the drive strength of the inverter chain.

The benefits of MBFFs are twofold: 1) clustered flip-flops consume less dynamic power and area; and 2) the clock network can have a simpler topology, easier skew control [5], and lower power due to fewer clock sinks, a shallower depth, and fewer clock buffers (see Fig. 2). In addition, MBFF clustering performs well even if the clock cannot be turned off, and it can easily be combined with clock gating.

Based on these benefits, recent research endeavors have been devoted to MBFF clustering to reduce the switching capacitance C_{clk} [4], [6]–[9]. The single-bit flip-flops can be merged if their timing constraints can be satisfied after being merged.

Chen *et al.* [4] and Hou *et al.* [6] leverage on register banking at logic synthesis and at early physical synthesis, respectively. However, the subsequent timing and routing cost of the clustered result may somewhat deviate from what is expected at such early stages.

On the other hand, Yan and Chen [7], Chang *et al.* [8], and Wang *et al.* [9] postponed this task to postplacement to further consider the timing and even routing issues. Yan and Chen [7] analyzed the timing-safe region for each flip-flop and then constructed an intersection graph to record the pairwise overlapping of these regions. They reduced MBFF clustering to minimum clique partitioning and solved it by iteratively merging flip-flops with fewest compatible flip-flops. However, they assumed the available bit numbers of the given

Manuscript received June 13, 2011; revised August 13, 2011 and October 5, 2011; accepted October 26, 2011. Date of current version January 20, 2012. This work was supported in part by Faraday and the NSC of Taiwan, under Grant NSC 100-2220-E-009-047. An earlier version of this paper was presented at the ACM International Symposium on Physical Design, Santa Barbara, CA, March 2011 [20]. This paper was recommended by Associate Editor J. Hu.

I. H.-R. Jiang is with the Department of Electronics Engineering and the Institute of Electronics, National Chiao Tung University, Hsinchu 30010, Taiwan (e-mail: huiru.jiang@gmail.com).

C.-L. Chang and Y.-M. Yang are with the Institute of Electronics, National Chiao Tung University, Hsinchu 30010, Taiwan (e-mail: paralost.ee96@g2.nctu.edu.tw; yuming.yyang@gmail.com).

Color versions of one or more of the figures in this paper are available online at http://ieeexplore.ieee.org.

Digital Object Identifier 10.1109/TCAD.2011.2177459

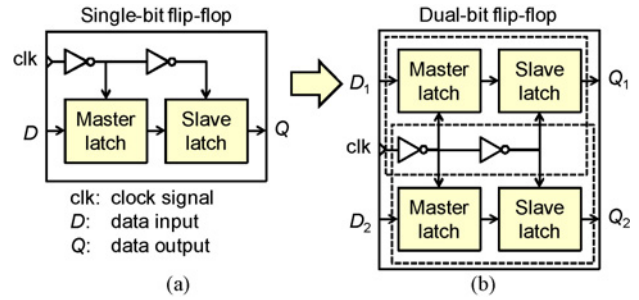

Fig. 1. (a) (Single-bit) flip-flop. (b) Dual-bit flip-flop with two sets of data input and output pins.

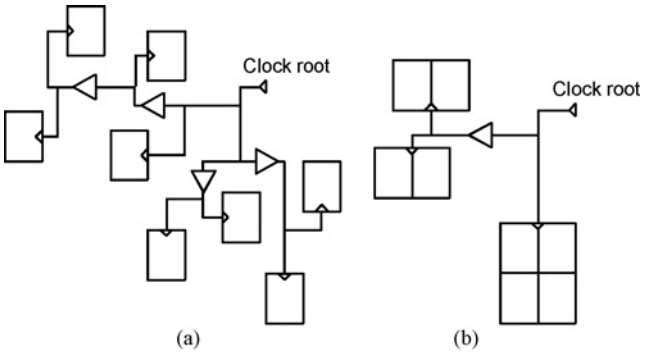

Fig. 2. Clock network. (a) Complicated topology without MBFFs. (b) Simple topology with MBFFs.

MBFF library are contiguous and unlimited. Considering a discrete and finite MBFF library, Chang *et al.* [8] proposed a window-based clustering method. The window iteratively slides, and its size progressively increases. They enumerated maximal cliques and reduced MBFF clustering to maximum independent set; they greedily cluster flip-flops inside the processing window and place MBFFs considering the routing cost and the placement density. However, the clustered flip-flops cannot cross the window boundary, and window sliding repeatedly visits layout bins. Moreover, minimum clique partitioning and maximum independent set are both NP-hard, so recent works [7] and [8] resort them to greedy heuristics. Even so, they may still incur a large storage requirement and a long runtime to solve a large-scale design due to their data structures. Very recently, instead of flip-flop power, Wang *et al.* [9] minimized the number of clock sinks (i.e., the number of MBFFs generated), as well as the influenced wire power of data pins. They enumerated all maximal cliques, generated a small set of MBFF candidates by random sampling, and then greedily clustered MBFFs to minimize their objectives.

In this paper, we tackle the postplacement MBFF clustering problem. Given a (contiguous or discrete) MBFF library and the timing slack of each flip-flop in a design, cluster flip-flops to minimize clock power (as well as the routing cost) subject to timing and placement density constraints.

By moving flip-flops and clustering them into MBFFs, we can have significant clock power reduction over the whole clock network. Meanwhile, the change of wire power on data pins is small compared with large clock power reduction. The postplacement MBFF clustering problem is NP-hard, so our goal is to solve it effectively and efficiently instead of optimally. Utilizing the properties of Manhattan distance and coordinate transformation, we successfully encode the feasible locations of flip-flops into two interval graphs [10], [11]. Particularly, to remedy the time and space complexities, we do not convert the interval graphs into the entire 2-D information and do not enumerate all possible combinations of compatible flip-flops. Instead, our representation is a pair of linear-sized sequences. We identify "decision points" in the sequences, at which there exist some essential flip-flops to be clustered. We cluster flip-flops only at decision points, and the number of decision points is significantly smaller than the number of flip-flops, thus doing so leads to an efficient clustering scheme. We then place MBFFs as close to the optimum location to reduce the routing cost as possible. Our strength results from the concise representation, and our key features are as follows.

1) From the time and space complexities' viewpoint, the representation—a pair of linear-sized sequences—implies an efficient data structure, and our fast coordinate transformation makes the execution of our algorithm very efficient.
2) The number of decision points is significantly smaller than the number of flip-flops. We cluster flip-flops at only decision points thus leading to an efficient clustering scheme.
3) Without enumerating all compatible combinations, we sweep and interleave two sequences to extract the compatibility on the fly and maintain the global view.
4) Our data structure and algorithm are both independent of the number of layout grids and/or bins.
5) We can refine an MBFF-clustered design. The preclustered MBFFs can be collapsed or preserved.
6) We can integrate our algorithm with clock gating. Considering the compatibility of clock enable signals during MBFF clustering can gain dual benefits.
7) We propose a wirelength-oriented clustering extension to handle a design with loose timing constraints.

Three experiments are conducted on industrial circuits. Our results show that the concise representation delivers superior efficiency and effectiveness. Compared with the very recent work [7] and [8], we can deliver the best power saving (only 0.17% away from the power lower bound) and the shortest runtimes (359X and 17X speedups). Our results also show that the number of decision points is significantly smaller than the number of flip-flops indeed (only 12% of flip-flop count). For the designs with loose timing constraints, our wirelength-oriented clustering method can achieve the best power saving and wirelength reduction and further gains 227X and 533X speedups. Compared with MBFF clustering at logic synthesis [4], power of the whole clock network can considerably be improved by postplacement MBFF clustering, even under timing and placement density constraints.

The remainder of this paper is organized as follows. Section II gives the problem formulation and analyzes design information. Section III derives the properties of Manhattan distance and coordinate transformation, gives our representation, and introduces the concept of decision points. Section IV

TABLE I
MBFF LIBRARY: POWER VERSUS AREA

Bit Number	Power	Area	Normalized Power per Bit	Normalized Area per Bit
1	100	100	1.00	1.00
2	172	192	0.86	0.96
4	312	285	0.78	0.71

presents our clustering and placement algorithm—INTEGRA. Section V extends our method to simultaneously consider power saving and wirelength minimization. Section VI shows our experimental results. Finally, Section VII concludes this paper.

II. PROBLEM FORMULATION AND PRELIMINARIES

In this section, we describe the problem formulation, detail the design information and constraints, and give the lower bounds of the objective functions.

A. Problem Formulation

The postplacement MBFF clustering problem is formulated as follows.

1) *Multibit Flip-Flop Clustering Problem:* Given a (contiguous or discrete) MBFF library and the timing slacks of flip-flops in a design, cluster flip-flops to minimize flip-flop power as well as the routing cost subject to timing slack and placement density constraints.

Flip-flop power is the primary objective, while the routing cost is secondary. In the sequel, a flip-flop means a single-bit flip-flop, while an MBFF means a multibit one. The input design may contain preclustered MBFFs.

B. MBFF Library

An MBFF library cell is associated with its bit number, consumed power, and occupied area. The normalized power and area per bit are decreasing as the bit number is increasing (see Table I). An MBFF library cell is redundant if it has greater power and larger area than another cell of the same bit number; the redundant cells can then be pruned.

C. Placement Density

Assume the chip area contains $W \times H$ grids. As shown in Fig. 3, the chip area is divided into $W_c \times H_c$ bins, and a bin is further divided into $W_b \times H_b$ grids. ($W = W_c W_b$, $H = H_c H_b$.) Each gate should be placed on some grid point, and each grid can be occupied by at most one gate. Considering routing congestion, the placement density constraint restricts the area utilization of bin b

$$A_{fb} \leq T_b(W_b H_b A_g - A_{pb}) - A_{cb} \tag{2}$$

where A_{fb} is the area available for flip-flop placement, T_b is the target density, A_g is the grid area, A_{pb} is the area belonging to macros, and A_{cb} is the area occupied by combinational elements within bin b.

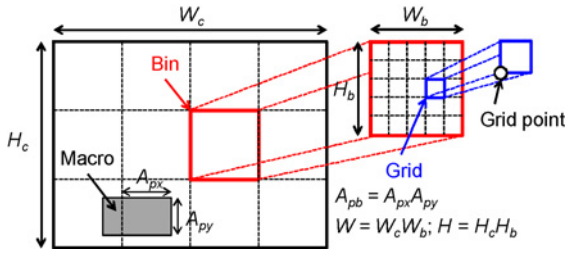

Fig. 3. Layout bins and grids.

D. Routing Cost and Timing Slack

The routing cost $L(i)$ of a flip-flop i is the sum of flip-flop input and output wirelength. For simplicity, the input (respectively, output) wirelength is the Manhattan distance between the flip-flop and its fanin (fanout) gate

$$L(i) = L_{fi}(i) + L_{fo}(i) \quad \forall i \tag{3}$$

where $L_{fi}(i)$ (respectively, $L_{fo}(i)$) is the Manhattan distance to its fanin $f_i(i)$ (fanout $f_o(i)$) gate. The routing cost also reflects the change on the wire power of data input and output pins. The amount of change is quite small compared with the clock power saving.

The timing slack of a flip-flop contains the input and output slacks. The input (respectively, output) slack of a flip-flop is the maximum allowable wire delay between the flip-flop and its fanin (fanout) gate without timing violations. After timing analysis, we budget a half of each path slack for the input (respectively, output) slack of the capture-edge (launch-edge) flip-flop. For a routing congested design, we may preserve a timing margin for routing detour. The margin can be set according to the congestion value of the bin where a flip-flop is located.

According to Synopsys' Liberty Library, the delay of a gate, lumped with its output wire delay, is a function of its output loading and input transition; the calibrated values are stored in 2-D tables. Prior work [13] observed the loading dominance effect, which indicates that the change in the gate delay is dominated by the output loading. The output loading consists of wire loading, the input capacitance of fanout gates, and its output pin capacitance. The wire loading is proportional to the sum of the wirelength between this gate and each of its fanout gates. Since the placement of combinational elements is unchanged during postplacement MBFF clustering, according to timing slacks, we can obtain the maximum allowable wire loading (i.e., wirelength) between a flip-flop and its fanin/fanout gate by table-lookup. Hence, slacks can be converted to equivalent wirelength bounds as follows:

$$\begin{aligned} 0 < L_{fi}(i) \leq S_{fi}(i) \\ 0 < L_{fo}(i) \leq S_{fo}(i) \quad \forall i \end{aligned} \tag{4}$$

where $S_{fi}(i)$ (respectively $S_{fo}(i)$) is the equivalent wirelength of the input (respectively output) slack. If flip-flop i has multiple fanouts, each fanout $f_o(i)$ contributes a routing cost and gives an output slack.

TABLE II
DYNAMIC PROGRAMMING POWER TABLE FOR THE MBFF LIBRARY GIVEN IN TABLE I

Minimum Power $P_{\min}(n)$		n				
		0	1	2	3	4
MBFF library cell options (#bits)	$\emptyset$	0	0	0	0	0
	{1}	0	100	200	300	400
	{1, 2}	0	100	172	272	344
	{1, 2, 4}	0	100	172	272	312

E. Power Analysis for MBFF Library

Given an MBFF library and the number n of flip-flops, we can analyze the best configuration for minimum power without considering the timing slack constraint, i.e., all flip-flops can freely be merged. This analysis can be reduced to the 0–1 Knapsack problem—solvable by dynamic programming [10]. The dynamic programming table lists the lower bound of power for each possible n under different MBFF library cell options. Given m irredundant MBFF cells whose least common multiple of bit numbers is b_{lcm}, the size of the dynamic programming table is only $(m+1) \times (b_{lcm}+1)$ (see Table II, where $m = 3$, $b_{lcm} = 4$). An arbitrary number n can be decomposed into $n = n_0 b_{lcm} + n_1$, where n_0 and n_1 are nonnegative integers; we then have the minimum power $P_{\min}(n)$ for n

$$P_{\min}(n) = n_0 P_{\min}(b_{lcm}) + P_{\min}(n_1) \tag{5}$$

e.g., as listed in Table II, the minimum power for 10 (= 2×4 + 2) bits is 796 (= 2×312 + 172).

F. Lower Bound of the Routing Cost

Based on (3), the routing cost $L(i)$ of flip-flop i is the sum of its input and output wirelengths, which is estimated by total Manhattan distances (see Section II-D). If flip-flop i is located within the bounding box of its fanin and fanout gates, its routing cost can achieve the minimum; we have the lower bound $L_{\min}(i)$ of $L(i)$

$$\begin{aligned} L_{\min}(i) &= \min\{L_{fi}(i) + L_{fo}(i)\} \\ &= |x_{fi}(i) - x_{fo}(i)| + |y_{fi}(i) - y_{fo}(i)| \end{aligned} \tag{6}$$

where $(x_{fi}(i), y_{fi}(i))$ and $(x_{fo}(i), y_{fo}(i))$ are flip-flop i's fanin and fanout gates' coordinates. Hence, the lower bound of the overall routing cost is $\sum_{i=1}^{n} L_{\min}(i)$.

For a specified b-bit MBFF j including flip-flops $j_1, j_2, \ldots, j_b$, its routing cost is as follows:

$$\sum_{i=1}^{b} L(j_i) = \sum_{i=1}^{b} \{L_{fi}(j_i) + L_{fo}(j_i)\}. \tag{7}$$

The minimum routing cost of MBFF j occurs when j is located within the bounding box defined by low/high median x and y coordinates of its all fanin and fanout gates

$$\begin{aligned} &m_l(x_{fi}(j_1), \quad x_{fo}(j_1), \ldots, x_{fi}(j_b), \quad x_{fi}(j_b)), \\ &m_h(x_{fi}(j_1), \quad x_{fo}(j_1), \ldots, x_{fi}(j_b), \quad x_{fi}(j_b)), \\ &m_l(y_{fi}(j_1), \quad y_{fo}(j_1), \ldots, y_{fi}(j_b), \quad y_{fi}(j_b)), \\ &m_h(y_{fi}(j_1), \quad y_{fo}(j_1), \ldots, y_{fi}(j_b), \quad y_{fi}(j_b)) \end{aligned}$$

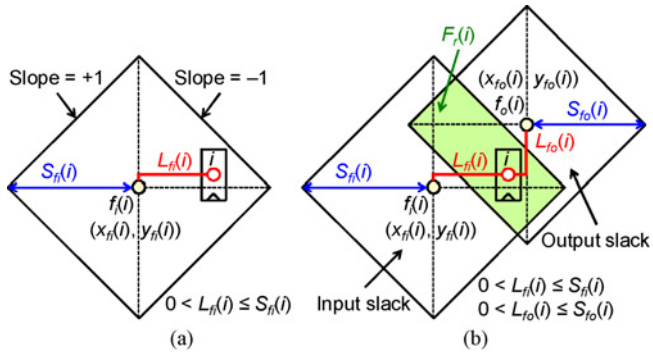

Fig. 4. (a) Input slack of flip-flop i defines a diamond. (b) Feasible region of i is the overlap of two diamonds.

where $m_l(Z)$ and $m_h(Z)$ denote low and high medians of set Z.

III. PROPERTIES AND REPRESENTATION

In this section, we derive the properties and introduce the representation used in our algorithm.

A. Feasible Region

According to (4), each flip-flop i's input (respectively, output) slack constraint defines a diamond whose center is located at i's fanin (fanout) gate and whose half diagonal length is the slack value [see Fig. 4(a)]. Flip-flop i's feasible region $F_r(i)$ is the overlap of these diamonds defined by its input and output slacks [see Fig. 4(b)]. In addition, if flip-flop i has multiple fanouts, $F_r(i)$ is the overlap of the diamonds defined by its input and all output slacks. Flip-flop i is timing-safe if and only if it is placed within its feasible region $F_r(i)$.

B. Coordinate Transformation

Overlapping diamonds and checking if a grid point is located within the feasible region is computationally intensive. Hence, we accelerate the operations by coordinate transformation. Fig. 4 shows, for each flip-flop i, the diamonds of the input and output slacks are determined by line segments with slopes = ±1. After being rotated by 45° clockwise, the diamonds become squares. The feasible region can then be retrieved by overlapping squares, much more computationally efficient than overlapping diamonds.

Let all gates and flip-flops be placed at grids (integer coordinates) in a Cartesian coordinate system C. Consider a new coordinate system C' with origin at (0, 0) in C. The coordinate transformation between the grid point (x, y) in C and its counterpart (x', y') in C' is defined as follows:

$$x' = y + x \quad y' = y - x \tag{8}$$

$$x = \frac{x' - y'}{2} \quad y = \frac{x' + y'}{2}. \tag{9}$$

The unit length in C equals $\sqrt{2}$ unit length in C'. Because of the scaling factor, the transformation from C to C' based on (8) can be done by simple and fast integer addition/subtraction. In

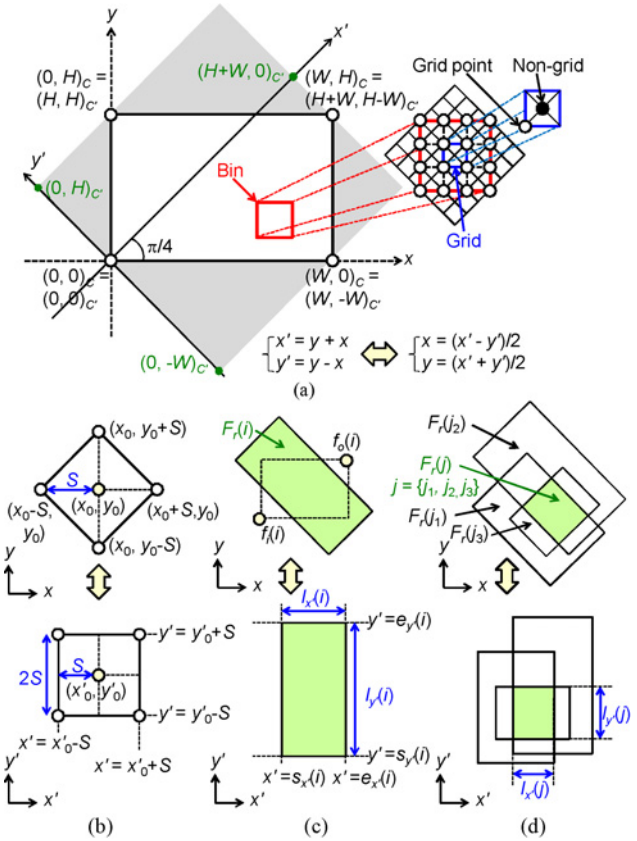

Fig. 5. Coordinate transformation. (a) Chip area. (b) Diamond centered at (x_0, y_0) of diagonal length $2S$ is transformed to a square of side length $2S$. (c) Each feasible region is represented by two intervals on x' and y' axes. (d) Feasible region of a tri-bit flip-flop is the overlap of all rectangles.

addition, each grid point in C is also defined in C'. Fig. 5(a) shows the transformed chip area, where the shaded area is outside the chip, $0 \le x \le W, 0 \le y \le H$, $0 \le x' \le H + W, -W \le y' \le H$. By (9), because a grid point has integer x and y coordinates, the sum and difference of its x' and y' coordinates must be even, i.e., *both x' and y' coordinates of a grid point must be even or odd*

$$x' \bmod 2 = y' \bmod 2. \tag{10}$$

Moreover, for each grid, its left-bottom corner coordinate is defined as the grid point, while its center is a nongrid point. Based on Fig. 4(a), Fig. 5(b) shows an input or output diamond centered at (x_0, y_0) of diagonal length $2S$ is transformed to a square of side length $2S$. Hence, the input and output slack constraints can be described by four x' and four y' coordinates. Based on Fig. 4(b), Fig. 5(c) shows the feasible region of a flip-flop is defined as a rectangle that is the overlap of its input and output squares. Let $s_{x'}(i)$, $e_{x'}(i)$, $s_{y'}(i)$, and $e_{y'}(i)$ denote the left, right, bottom, and top boundaries of flip-flop i's feasible region. For flip-flop i, we have

$$\begin{aligned}
s_{x'}(i) &= \max(y_{fi}(i) + x_{fi}(i) - S_{fi}(i), \quad y_{fo}(i) + x_{fo}(i) - S_{fo}(i)) \\
e_{x'}(i) &= \min(y_{fi}(i) + x_{fi}(i) - S_{fi}(i), \quad y_{fo}(i) + x_{fo}(i) - S_{fo}(i)) \\
s_{y'}(i) &= \max(y_{fi}(i) - x_{fi}(i) - S_{fi}(i), \quad y_{fo}(i) - x_{fo}(i) - S_{fo}(i)) \\
e_{y'}(i) &= \min(y_{fi}(i) - x_{fi}(i) + S_{fi}(i), \quad y_{fo}(i) + x_{fo}(i) - S_{fo}(i)).
\end{aligned} \tag{11}$$

As shown in Fig. 5(d), for a b-bit MBFF j including flip-flops $j_1, j_2, \ldots, j_b$, j's feasible region is the overlap of j_1's, j_2's,..., and j_b's feasible regions, that is

$$\begin{aligned}
s_{x'}(j) &= \max(s_{x'}(j_1), \ldots, s_{x'}(j_b)) \\
e_{x'}(j) &= \min(e_{x'}(j_1), \ldots, e_{x'}(j_b)) \\
s_{y'}(j) &= \max(s_{y'}(j_1), \ldots, s_{y'}(j_b)) \\
e_{y'}(j) &= \min(e_{y'}(j_1), \ldots, e_{y'}(j_b)).
\end{aligned} \tag{12}$$

C. Representation

After coordinate transformation, the feasible region of flip-flop i is a rectangle in the new coordinate system C'. By projecting this rectangle on x' and y' axes, we have two intervals for each flip-flop. Flip-flop i's x' interval $I_{x'}(i)$ and y' interval $I_{y'}(i)$ are specified by their starting and ending points, i.e., boundary coordinates defined by (11)

$$\begin{aligned}
&I_{x'}(i) = [s_{x'}(i), e_{x'}(i)] \text{ and} \\
&I_{y'}(i) = [s_{y'}(i), e_{y'}(i)] \text{ for each flip-flop } i.
\end{aligned} \tag{13}$$

Hence, all feasible regions are encoded by two interval graphs, $G_{x'} = (V_{x'}, E_{x'})$ for x' intervals and $G_{y'} = (V_{y'}, E_{y'})$ for y' intervals

$$\begin{aligned}
&V_{x'} = \{I_{x'}(1), \ldots, I_{x'}(n)\}, \text{ and} \\
&(I_{x'}(i), I_{x'}(j)) \in E_{x'} \Leftrightarrow I_{x'}(i) \bigcap I_{x'}(j) \neq 0, 1 \le i, j \le n \\
&V_{y'} = \{I_{y'}(1), \ldots, I_{y'}(n)\}, \text{ and} \\
&(I_{y'}(i), I_{y'}(j)) \in E_{y'} \Leftrightarrow I_{y'}(i) \bigcap I_{y'}(j) \neq \emptyset, 1 \le i, j \le n.
\end{aligned} \tag{14}$$

A b-bit MBFF j including flip-flops $j_1, j_2, \ldots, j_b$ implies $j_1, j_2, \ldots, j_b$ form a clique[1] not only on x' interval graph but also on y' interval graph. To avoid high-space complexity, we do not construct two interval graphs.

Instead of two graphs, our representation is a pair of linear-sized sequences: (X', Y'). The two sequences store the starting and ending points of x' and y' intervals in ascending order; if several points have the same coordinate, starting points are ordered before ending points. We have sequence X' for x' intervals and sequence Y' for y' intervals, $|X'| = |Y'| = 2n$, where n is the number of flip-flops. Sequences X' and Y' are as follows:

$$\begin{aligned}
&X' =< x'_1, \ldots, x'_{2n} > \text{ where } x'_1 \le x'_2 \le \cdots \le x'_{2n} \\
&x'_j \in \{s_{x'}(i),\ e_{x'}(i) : i = 1, \ldots, n\}, \quad j = 1, \ldots, 2n \\
&Y' =< y'_1, \ldots, y'_{2n} > \text{ where } y'_1 \le y'_2 \le \cdots \le y'_{2n} \\
&y'_j \in \{s_{y'}(i),\ e_{y'}(i) : i = 1, \ldots, n\}, \quad j = 1, \ldots, 2n.
\end{aligned} \tag{15}$$

Please note that in our algorithm, X' is constructed at the beginning, while Y' is dynamically and partially generated by request (see Section IV).

Fig. 6(a) gives a subdesign with eight flip-flops, say $0, 1, \ldots, 7$, where the tilted boxes are feasible regions. Fig. 6(b) shows the transformed feasible regions, each of which is a rectangle aligned with x' and y' axes. Fig. 6(c) and (d) illustrates the corresponding x' and y' intervals. Fig. 6(e) lists the corresponding sequence X'.

[1]A clique in an undirected graph is a subset of its vertices such that every two vertices in the subset are connected by an edge. A maximal clique is a clique that is not contained in any other clique. Among all maximal cliques, the largest one is the maximum clique.

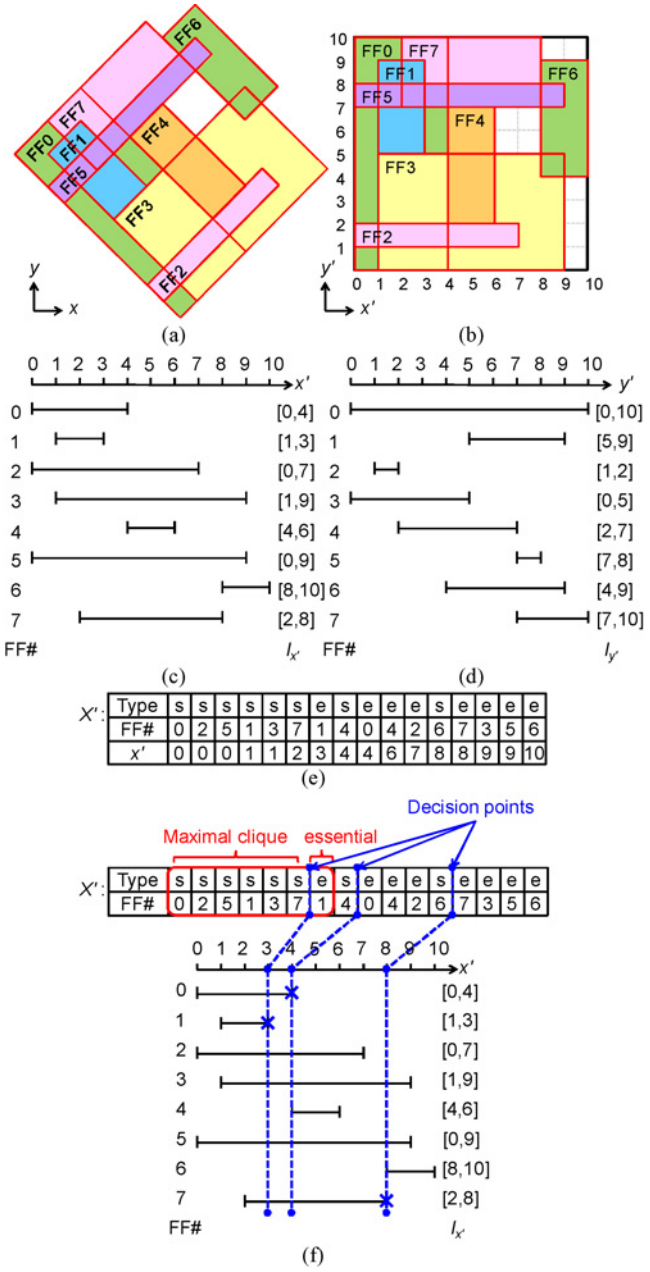

Fig. 6. Subdesign with eight flip-flops. (a) Feasible regions. (b) Transformed feasible regions. (c) x' intervals. (d) y' intervals. (e) X' is of size 16. (f) Initially, there are three decision points at flip-flops 1's, 0's, and 7's ending points. The maximal clique retrieved at the first decision point in X' is $\{0, 1, 2, 3, 5, 7\}$. The essential flip-flop is flip-flop 1, and the related flip-flops include flip-flops 0, 2, 3, 5, 7.

D. Maximal Cliques and Decision Points

We introduce "decision points" and "essential flip-flops" as follows.

Definition 1: If there exist two consecutive points x'_k and x'_{k+1} in X', where $x'_k = s_{x'}(i), x'_{k+1} = e_{x'}(j), 1 \le i, j \le n$, a decision point is the coordinate of x'_{k+1}, i.e., $e_{x'}(j)$.

Definition 2: An essential flip-flop with respect to a decision point is a flip-flop whose ending point x'_j lies between the current decision point x'_i and the next decision point x'_k ($x'_i \le x'_j < x'_k$) or between the current decision point x'_i and the end of X' if x'_i is the last decision point.

Theorem 1: Consider X', a decision point, and the corresponding set of essential flip-flops. The maximal clique containing the set of essential flip-flops in the x' interval graph can be found at this decision point.

Proof: Given an arbitrary point x'_0 in the x'-axis, $K(x'_0)$ is defined as the maximum sized clique that can be retrieved at x'_0. $K(x'_0)$ contains all flip-flops whose intervals cross x'_0, i.e., whose starting points are less than or equal to x'_0, and ending points are greater than or equal to x'_0.

According to the construction of X', the axis x' can be divided into $2n-1$ segments by the starting and ending points of all intervals. There are two situations for the ith segment starting at x'_i and ending at x'_{i+1}.

1) $x'_i < x'_{i+1}$. Consider an arbitrary point x'_0 within this segment, $x'_i < x'_0 < x'_{x+1}$.
   Case "*ss*:" x'_i and x'_{i+1} are both starting points. $K(x'_i) = K(x'_0) \subset K(x'_{i+1})$.
   Case "*se*:" x'_i is a starting point, while x'_{i+1} is an ending point. $K(x'_i) = K(x'_0) = K(x'_{i+1})$.
   Case "*es*:" x'_i is an ending point, while x'_{i+1} is a starting point. $K(x'_0) \subset K(x'_i), K(x'_0) \subset K(x'_{i+1})$.
   Case "*ee*:" x'_i and x'_{i+1} are both ending points. $K(x'_{i+1}) = K(x'_0) \subset K(x'_i)$.
2) $x'_i = x'_{i+1}$. There are only three cases: "*ss*," "*se*," "*ee*." ("*es*" does not happen according to the construction of X'.) $K(x'_i) = K(x'_{i+1})$.

First, each maximal clique must be a maximum sized clique that can be retrieved at some point in the x'-axis.

We define the type of an entry in X' as s for a starting point and as e for an ending point. By definition, X' is interleaved with s strings and e strings as follows:

$$s\,s\,s\,s \ldots s\,e\,e\,e \ldots e\,s\,s\,s \ldots s\,e\,e\,e \ldots e.$$

If we scan X' from left to right segment by segment, we may record the maximum sized cliques found at the boundaries of segments. According to the above analysis, the maximum sized clique retrieved at the first "*e*" after some *s* string is maximal. By definition, such *e* is a decision point. In addition, all of its essential flip-flops cross over this decision point because their starting points must be listed before the corresponding "*e*" of this decision point. The theorem thus follows. ■

Based on the proof of the above theorem, to find *all*[2] maximal cliques, we need to check only decision points rather than all ending points. In the sequel, for a decision point, the *related flip-flops* mean the remaining flip-flops in the maximum sized clique retrieved at this decision point excluding the essential flip-flops.

Corollary 1: A decision point corresponds to at least one essential flip-flop. Hence, the number of decision points is less than or equal to the number of flip-flops.

Considering the instance in Fig. 6(a), Fig. 6(f) shows there exist three decision points at flip-flops 1's, 0's, 7's ending

[2]In [11], Ramalingam and Rangan ordered the intervals in ascending order of their ending points. For a sorted sequence, they proposed a linear-time algorithm to compute *one* maximal clique for each interval. By sorting starting and ending points together, our method can find all maximal cliques for each interval. In addition, Lee proposed a $O(n \log n)$ algorithm to find only one maximum clique in a rectangle intersection graph [12].

points. Flip-flop 1 is the first decision point's essential flip-flop, flip-flops 0, 4, and 2 are the second decision point's, and flip-flops 7, 3, 5, and 6 are the third decision point's. In addition, this instance has eight flip-flops but only three decision points.

A decision point in X' means at which there exist some flip-flops that we should decide how to cluster; otherwise, they cannot be clustered any more. These flip-flops are essential flip-flops with respect to this decision point. For example, Fig. 6(f) shows that at flip-flop 1's ending point (the first decision point), flip-flop 1 has to be clustered, i.e., it is essential. Flip-flops 0, 2, 3, 5, 7 are its related flip-flops, meaning {0, 1, 2, 3, 5, 7} forms a maximal clique on x' interval graph. Moreover, considering the interval from the second decision point (flip-flop 0's ending point) to the third decision point (flip-flop 7's ending point), $[4, 8)$, the maximal clique containing flip-flops 0, 4, 2 can be found at flip-flop 0's ending point, {0, 2, 3, 4, 5, 7}.

IV. MBFF Clustering

In this section, we detail our MBFF clustering algorithm, INTEGRA, based on our representation and properties developed in Section III. We utilize the concept of decision points and essential flip-flops to reduce the number of times that flip-flop clustering is applied. INTEGRA contains three phases: initialization, flip-flop clustering, and flip-flop placement. Fig. 7 lists the procedure of INTEGRA, and the following is a brief summary.

1) INTEGRA preprocesses the design intent [see Fig. 6(e)].
2) INTEGRA iteratively finds a decision point in X' and extracts the essential flip-flops and their related flip-flops [see Fig. 8(a), (b)].
3) INTEGRA finds the maximal clique in the partial Y' for each essential flip-flop [see Fig. 8(c)].
4) INTEGRA clusters each essential flip-flop [see Fig. 8(c)].
5) INTEGRA places the clustered flip-flop at a legal location with routing cost and density consideration [see Fig. 8(e)].
6) INTEGRA repeats steps 2–5 until all flip-flops are investigated.

The number of decision points is smaller than or equal to the number of flip-flops, thus leading to an efficient clustering scheme. In addition to efficiency, INTEGRA can refine a preclustered design.

A. Phase 1: Initialization

At the initialization phase, INTEGRA preprocesses the design intent as follows.

In line 1 in Fig. 7(a), the MBFF library cells are sorted to facilitate the selection of an appropriate MBFF cell for a maximal clique. Considering power and area together, the MBFF library cells are thus lexicographically sorted in ascending bit number, descending power, and descending area order, e.g., the corresponding ordered list of Table I is as follows:

$$< (1, 100, 100), (2, 172, 192), (4, 312, 285) > .$$

```
Algorithm INTEGRA
// Initialization
1.  lexicographically sort the MBFF library
2.  collapse MBFFs
3.  X' ← sort {s_x(i), e_x(i): i = 1..n}, j ← 1, Q ← ∅
// Main body
4.  while (X' is not empty) do
5.     find a decision point
6.     Q ← Q + essential flip-flops and unclustered related flip-flops
7.     Y' ← sort {s_y(i), e_y(i): i ∈ Q}
8.     foreach essential flip-flop k do
          // Flip-flop clustering
9.        K_max ← max_clique(Y', k)
10.       find the appropriate MBFF cell of bit number B for |K_max|
11.       sort K_max - {k} in the ascending order of e_x(i) for i ∈ K_max - {k}
12.       K_j ← flip-flop k and the first (B-1) flip-flops in K_max
          // Flip-flop placement
13.       find bounding box B_b for K_j
14.       find the points in F_A(K_j) nearest to the 4 corners and center of B_b
15.       select the point with minimum distance
16.       legalize this point and assign it to MBFF K_j
17.       if legalization fails, then go to line 12 for other options
18.       Q ← Q - K_j, X' ← X' - K_j, Y' ← Y' - K_j
19.       j++
```

(a)

```
Subroutine max_clique(Y', k)
// scan Y'
// find the largest partial sum within [s_y(k) , e_y(k)]
// check = 1 within [s_y(k) , e_y(k)]; otherwise, 0.
1.  K ← K_max ← ∅, check←size ← max_size ← 0
2.  for i = 1 to |Y'| do
3.    if Y'[i].Type = 's' then // Y'[i] is a starting point
4.       K ←K + Y'[i].FF
5.       size++
6.       if (Y'[i].FF=k) then
7.             check ← 1, max_size ← size, K_max ← K
8.       if (check=1 and size > max_size) then
9.             max_size ← size, K_max← K
10.   else // Y'[i] is an end point
11.      K ←K - Y'[i].FF, size--
12.      if (Y'[i].FF=k) then check ← 0
13.return K_max
```

(b)

Fig. 7. Our MBFF clustering algorithm: INTEGRA. (a) Overall procedure. (b) Maximum clique subroutine.

In line 2, each preclustered b-bit MBFF is collapsed into b flip-flops; the collapsed flip-flops are temporarily placed at the MBFF's location. Because the initial MBFFs in the input design may be clustered by some heuristics or other tools, we recluster them for further power reduction. If they are preclustered based on designers' intentions, they are preserved. In line 3, the feasible region of each flip-flop is computed according to its timing slack. X' is also created (see Section III-C).

B. Phase 2: Flip-Flop Clustering

We cluster flip-flops based on the representation and properties described in Section III. As mentioned earlier, an MBFF can be formed if the corresponding flip-flops form a clique not only on x' interval graph but also on y' interval graph. In Section III-D, we introduced the concept of decision points to retrieve maximal cliques in X'. To form an MBFF, the maximal clique found in X' should be further verified by the y' intervals of its members. For the flip-flops in the maximal clique in X', the starting and ending points of their y' intervals are sorted in ascending order and form the working sequence Y'. Similarly,

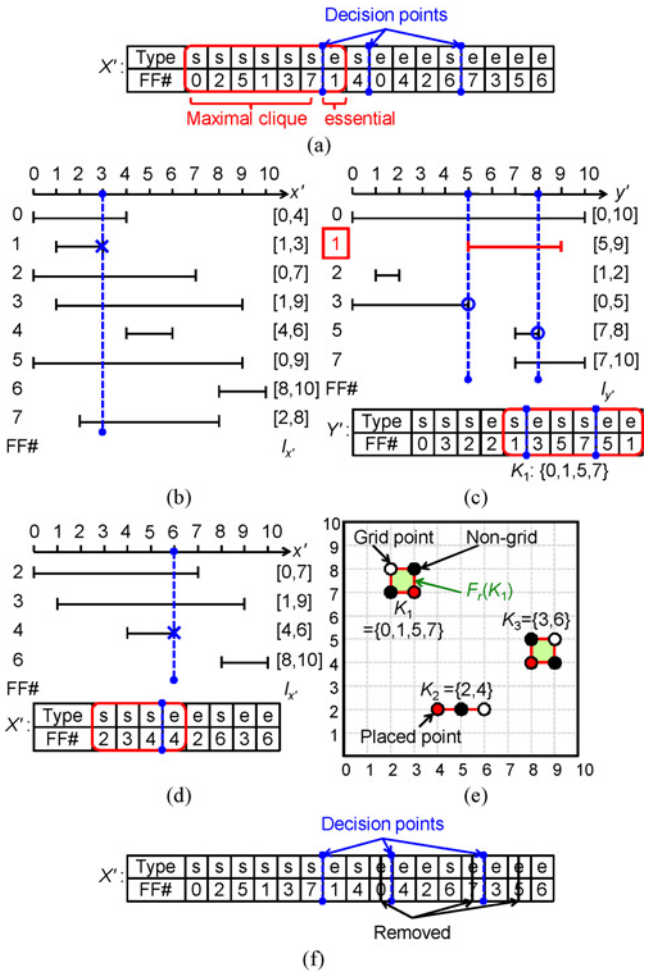

Fig. 8. Flip-flop clustering. (a) Based on the example given in Fig. 6, initially, there are three decision points at flip-flops 1's, 0's, and 7's ending points. (b) At the first decision point in X', the essential flip-flop is flip-flop 1, and the related flip-flops include flip-flops 0, 2, 3, 5, 7. (c) Corresponding Y'. Flip-flop 1's maximal clique is $\{0, 1, 5, 7\}$. The appropriate MBFF cell is 4-bit flip-flop, so $K_1 = \{0, 1, 5, 7\}$. (d) After removing K_1 from X', we find the next decision point, at flip-flop 4's end point. We then cluster flip-flops 2 and 4. Finally, we have MBFFs: $K_1 = \{0, 1, 5, 7\}$, $K_2 = \{2, 4\}$, $K_3 = \{3, 6\}$. (e) Feasible regions of MBFFs, where circles indicate grid points, while dots indicate nongrid points. The shaded points indicate the final placement of each MBFF. (f) Runtime decision points are at flip-flops 1's, 4's, and 3's end points.

the maximal clique in Y' of each essential flip-flop can be found by checking the decision points in Y' within the y' interval of the essential flip-flop [see Fig. 7(b)]. Fig. 8(c) shows that the decision points in Y' within flip-flop 1's y' interval are flip-flops 3's and 5's ending points. Flip-flop 3 belongs to a maximal clique $\{0, 1, 3\}$, while flip-flop 5 belongs to $\{0, 1, 5, 7\}$. We choose $\{0, 1, 5, 7\}$ and check the sorted MBFF library to find an appropriate MBFF cell to cluster them. An appropriate MBFF cell means its bit number is the largest among all cells of bit number smaller than or equal to the clique size. Assume the MBFF library is specified by Table I. Flip-flops 0, 1, 5, 7, hence, form a 4-bit MBFF; flip-flops 0, 1, 5, 7 are then removed from X'.

If the clique size does not fit the bit number of the selected MBFF cell (it usually happens for a discrete MBFF library), we need more than one MBFF cell to cover this clique. Although Table II lists the best configuration of MBFF cells to cluster this found clique, at this time we just cluster the biggest (most power-efficient) MBFF cell in this configuration. We defer the decision making for the remaining flip-flops since they may have other options to form a larger/better MBFF at later iterations. We greedily cluster from a flip-flop with the smallest x' ending point because a larger ending point implies the flip-flop may have more chances to be clustered.

The created MBFF is then placed at a legal grid point with routing cost and placement density consideration (see Section IV-C). Fig. 8(d) shows the updated X' after flip-flops 0, 1, 5, 7 are clustered, where the next decision point is at flip-flop 4's ending point. The same process is repeated until all flip-flops are investigated, i.e., X' is empty.

Since flip-flops are iteratively clustered and removed from X', the size of X' gradually shrinks, and the runtime decision points might be deferred or even disappear. For example, Fig. 8(f) indicates that runtime decision points are at flip-flops 1's, 4's, and 3's ending points [see Fig. 8(a)].

INTEGRA iteratively finds a decision point in line 5, records the essential and unclustered related flip-flops by queue Q in line 6, and creates the corresponding Y' in line 7. From line 8 to line 12, for each essential flip-flop, INTEGRA iteratively extracts the maximal clique in Y' and selects an appropriate MBFF cell to cluster it. After placing it (see Section IV-C), INTEGRA removes the clustered flip-flops from X', Y', and Q in line 18.

C. Phase 3: Flip-Flop Placement

Each MBFF generated by flip-flop clustering (see Section IV-B) is placed and legalized to a legal grid point as close to the optimum location to reduce the routing cost as possible. Fig. 8(e) depicts the clustered flip-flops, their feasible regions, and their final positions. A legal grid point satisfies the following conditions.

1) It is a grid point [checked by (10)].
2) It is not occupied by other fanin/fanout gates or flip-flops.
3) It is density-safe.

As mentioned in Section II-F, the minimum routing cost of MBFF j occurs when j is located within the bounding box B_b defined by low/high median x and y coordinates of its all fanin and fanout gates. To find the nearest point in the feasible region to the bounding box, [8] iteratively enlarges the bounding box until it reaches any point in the feasible region. Unlike [8], we do it efficiently. First, INTEGRA finds the x' and y' coordinates of the corner and center points of the bounding box in line 13. INTEGRA then finds the points in MBFF j's feasible region, which are nearest to the four corners and center of B_b and selects the best one in lines 14 and 15. The nearest points and their distances can efficiently be computed by checking their x' and y' coordinates. Fig. 9(a) and (b) shows two general cases that the bounding box may or may not overlap with the feasible region.

We consider the center point here to avoid unnecessary movement for some trivial cases. Some corners of the bounding box may be occupied by some fanin/fanout gates.

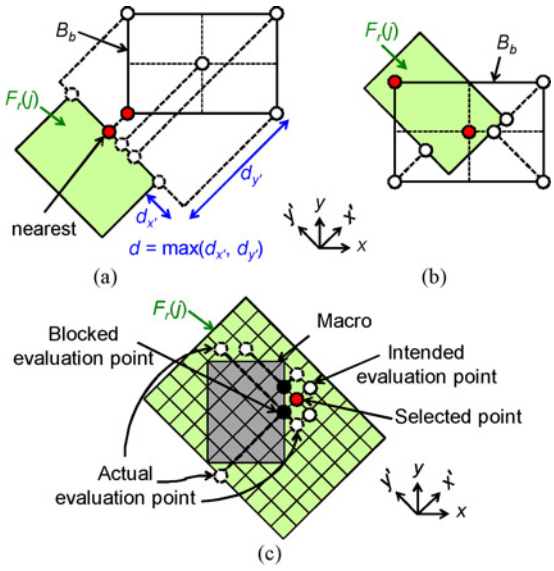

Fig. 9. Flip-flop placement: the optimal location with minimum routing cost. (a) MBFF j's bounding box B_b and the feasible region $F_r(j)$ do not overlap. (b) B_b and $F_r(j)$ overlap. If some corner and the center are located inside $F_r(j)$, the center point will be selected. (c) Tunneling through a preplaced macro or occupied grid point to find a legal point.

If we place the MBFF at a corner that happens to be some fanin/fanout gate, we need to legalize it. To prevent fanin/fanout gates from being directly selected, as shown in Fig. 9(b), if some corner and the center are both located inside $F_r(j)$, the center point will be selected.

In line 16, INTEGRA moves the selected point within the feasible region until a legal point is found. Considering preplaced macros and occupied grid points, we adopt the idea of tunneling during movement [19]. If a neighboring grid in x' and y' is occupied, then we tunnel through it [see Fig. 9(c)]. Please note that the maximum number of grids that need to be considered is limited to 13. (In case the selected point is occupied and it is surrounded by macros or by occupied points on all sides.) INTEGRA then places the MBFF at the found point. If there exist no legal points in the feasible region, it goes back to line 12 to try another clique or keeps k as a single-bit FF.

Table I indicates that the larger bit number, the smaller normalized area per bit. Once some flip-flops form an MBFF, the total occupied area is smaller, and the placement density constraint becomes easier to meet. Because of the extra released space, if MBFFs overlap with preplaced cells (combinational elements) or macros, legalization can be applied to adjust these preplaced cells; the displacement should be small, and the impact on timing is negligible.

D. Time and Space Complexities

Given n flip-flops, INTEGRA takes $O(n \log n)$ time to create X', $O(n)$ time to find decision points in X', totally $O(n \log n)$ time to create the working Y' and to find all maximal cliques in Y'. Hence, INTEGRA eventually uses $O(n \log n)$ time and $O(n)$ space.

We detail the analysis of time and space complexities of INTEGRA as follows. Initialization takes $O(n \log n)$ time in lines 1–3 in Fig. 7(a). Compared to the total flip-flop count, the number of MBFF library cells is relatively small, which can be considered as a constant; the lexicographical sorting in line 1 can be done in $O(1)$ time. In line 2, the number of preclustered MBFFs must be smaller than the total number of flip-flops, so the collapsing is done in $O(n)$. Line 3 takes $O(n \log n)$ time to create X'.

Flip-flop clustering and placement also takes $O(n \log n)$ time in lines 4–19. In lines 4–5, the while loop scans X' once and finds decision points, thus taking $O(n)$ time in total. In line 6, each flip-flop is pushed into Q at most once. In line 7, Y' is created by incrementally sorting Q. The overall sorting during the entire while loop can be done in $O(n \log n)$ time when Q is implemented by a balanced binary search tree. To analyze lines 8–11, we assume there are m decision points during the entire while loop, and for the ith decision point, there are n_i essential and related flip-flops found in line 6; $n_1 + \cdots + n_m = O(n)$. Hence, for the ith decision point, line 9 takes $O(n_i)$ time to scan Y' to find the maximal clique, while line 11 at most takes $O(n_i \log n)$ time to sort the found maximal clique. Hence, line 9 takes $O(n)$ time in total, while line 11 takes $O(n \log n)$ time in total. Lines 12–16 take $O(1)$ time for each formed MBFF and thus totally take $O(n)$ time. In line 18, every flip-flop is popped out Q at most once, thus totally $O(n)$ time.

V. Extension

In this section, we present two extensions.

A. Wirelength-Oriented Flip-Flop Clustering

When the timing slack constraint is not tight, the feasible region of a flip-flop greatly overlaps others' feasible regions or even fully covers the bounding box of its fanin and fanout gates. In this case, wirelength becomes the primary objective, and thus we propose a wirelength-oriented flip-flop clustering method.

We modify the criterion to select flip-flops from a given clique. After lines 9–10, a maximal clique $K_{\max}$ is extracted, and an adequate MBFF cell of bit number B is selected. Lines 11–12 in Fig. 7(a) are modified as follows. Each flip-flop in $K_{\max}$ is represented by the center of its bounding box. We iteratively cluster the closest pair of the representative points until we reach the appropriate size B. The first closest pair can be found by divide-and-conquer, while the subsequent pairs can be solved by the dynamic closest-pair algorithm [14].

Finally, the representative point of each clustered flip-flop is then moved as near to the center of the bounding box resulting from the low/high median fanin/fanout coordinates as possible.

Moreover, for a congested design, flip-flops may not be allowed to be moved far away. In this case, the MBFF clustering with a small amount of flip-flop displacement is desired. To achieve this goal, we may adopt a different setting of representative points. The representative point of each flip-flop is set as its initial location, while that of an intermediate clustered result is set as the median coordinate of their initial locations. The wirelength-oriented approach thus tends to cluster flip-flops with small displacement.

TABLE III
STATISTICS OF THE BENCHMARK USED IN [8]

Circuit	#FFs	Chip Size (#Grids)	Initial		Lower Bound	
			Power	WL	Power Ratio	WL Ratio
C1	120	600 × 600	11 384	89 425	82.2%	48.7%
C2	480	1200 × 1200	46 404	348 920	80.7%	49.9%
C3	1920	2400 × 2400	185 616	1 395 680	80.7%	49.9%
C4	5880	4200 × 4200	566 972	4 290 655	80.9%	49.7%
C5	12 000	6000 × 6000	1 160 100	8 723 000	80.7%	49.9%
C6	192 000	24 000 × 24 000	18 561 600	139 568 000	80.7%	49.9%

B. Integration with Clock Gating

INTEGRA can readily be extended to integrate with clock gating. Line 6 of INTEGRA should be modified as follows. The related flip-flops found in X' should have compatible clock enable signals [15]. After lines 7–12, the clock enable of the created MBFF is obtained by performing logic OR operation on the clock enable signals of its constituent flip-flops.

VI. EXPERIMENTAL RESULTS

We implemented INTEGRA in the C programming language and executed the program on a platform with an Intel Xeon 3.8 GHz CPU and with 16 GB memory under Ubuntu 10.04 OS. Three experiments are conducted to show our superior efficiency and effectiveness. The first two experiments focus on the flip-flop power saving as well as the wirelength impact on data pins before clock tree synthesis, while the third one completes clock tree synthesis and then shows the entire clock power saving.

A. Power-Oriented Flip-Flop Clustering

Table III lists six preclustered benchmark circuits provided by [8]. The effective number of single-bit flip-flops (#FFs) ranges from 120 to 192 000. The bin size is 100 × 100 grids. The MBFF library is specified in Table I. "Initial" lists the status of each input circuit. The lower bound ("Lower bound") of power is computed by the dynamic programming table and (5) given in Section II-E. The lower bound of wirelength is computed by (6).

Table IV(A) compares flip-flop power, wirelength (WL) on data pins of flip-flops, and runtime (Time) among [7], [8] and INTEGRA. "Power ratio" (respectively, "WL ratio") means the power (wirelength) after MBFF clustering over that of the input design; "#Decision" represents the number of runtime decision points; "Avg. ratio" means the average runtime speedup and the ratio of #Decision and #FFs; "woc" and "wc" mean without and with collapsing initial MBFFs [line 2 in Fig. 7(a)], respectively. For fair comparison, we modify [7] to handle discrete MBFF libraries. When a clique in the intersection graph is found, we allocate an MBFF cell of the appropriate bit number and cluster flip-flops starting with the smallest degrees. Considering power saving, INTEGRA generates solutions with only 0.17% away from the lower bounds, while the modified versions of [7] and [8] are 0.38% and 2.36% away, respectively. Keeping wirelength almost unchanged, INTEGRA can outperform modified [7] and [8] with 359X and 17X speedups. In particular, [7] and [8] suffer from long runtimes for C6, which is a large-scale design with numerous bins/grids. Moreover, the number of runtime decision points is significantly smaller than the number of flip-flops, on average 12% of flip-flop count. On the other hand, Table IV(B) compares the number of clock sinks. Interestingly, our power-oriented approach can also generate much fewer sinks even though our main objective is flip-flop power reduction. In addition, the initial MBFFs in these designs are generated by some heuristic considering only 1-bit and 2-bit flip-flops. They are preserved in [8]. It can be seen that collapsing these initial MBFFs and reclustering them [ours (wc)] leads to better results than preserving them [ours (woc)].

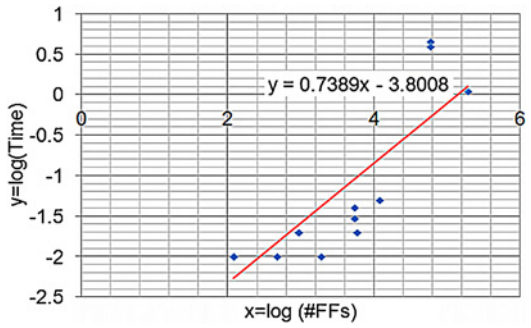

Fig. 10. Log–log graph of the runtime (y-axis) versus the number of flip-flops (x-axis). Empirically, INTEGRA can be done in $O(n^{0.7})$ time.

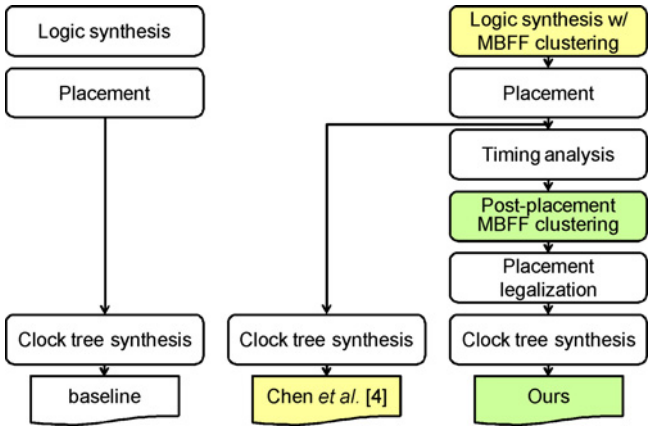

Fig. 11. Experimental flow. The baseline is a design with only single-bit flip-flops. [4] clusters flip-flops using RTL clock gating by Design Compiler [17].

After regression analysis, Fig. 10 depicts the log–log graph of the runtime (y-axis) versus the number of flip-flops (x-axis). Empirically, INTEGRA can be done in $O(n^{0.7})$ time.

B. Wirelength-Oriented Flip-Flop Clustering

When timing constraints are not tight, the wirelength becomes the primary objective. Unlike Table III, the circuits listed in Table V(A) reflect other possible difficulties for MBFF clustering [16]. The number of flip-flops (#FFs) ranges from 120 to 60 000, while the chip size ranges from 3.6×10^5 to 6×10^6 grids. Each grid is 5×5 (unit length)^2. The MBFF library of S1 and S2 is specified in Table I; the MBFF library of T2 includes a 13-bit MBFF cell; the MBFF library of T3

TABLE IV

POWER-ORIENTED FLIP-FLOP CLUSTERING COMPARISON WITH [7] AND [8] BASED ON THE BENCHMARK USED IN [8]

(A) Power, Wirelength, and Runtime

Circuit	#FFs	Modified [7] (wc)			[8] (woc)			Ours (woc)			Ours (wc)			
		Power ratio	WL ratio	Time (s)	Power ratio	WL ratio	Time (s)	Power ratio	WL ratio	Time (s)	Power ratio	WL ratio	#Decision	Time (s)
C1	120	82.8%	123.0%	0.03	85.2%	91.7%	<0.01	83.3%	103.6%	<0.01	82.8%	96.2%	28	<0.01
C2	480	81.2%	124.8%	0.11	83.1%	94.7%	0.02	81.2%	110.3%	<0.01	80.9%	101.7%	90	<0.01
C3	1920	81.3%	125.2%	0.53	82.9%	94.8%	0.07	81.1%	113.4%	0.03	80.8%	103.3%	229	<0.01
C4	5880	81.2%	124.7%	2.55	83.2%	94.5%	0.23	81.3%	112.9%	0.10	81.0%	103.8%	458	0.02
C5	12 000	81.0%	124.2%	8.01	82.9%	94.9%	0.52	81.1%	113.8%	0.28	80.7%	104.5%	690	0.05
C6	192 000	81.0%	124.4%	1994.61	82.8%	94.9%	76.94	81.1%	113.7%	5.58	80.7%	105.0%	3007	1.11
Avg. ratio	1.00	–	–	358.61	–	–	16.87	–	–	5.07	–	–	0.12	1.00

1. #FFs: number of single-bit flip-flops, bin size: 100×100 grids, grid size 5×5 unit-length^2, Max. density (FF area per bin): 19 000.
2. Wirelength (WL): the total wirelength between flip-flops and fanin/fanout gates.
3. woc: without collapsing initial MBFFs, wc: with collapsing initial MBFFs.

(B) Clock Sinks

	Initial		Modified [7] (wc)		[8] (woc)		Ours (woc)		Ours (wc)	
	Total #Sinks	#MBFFs 1-/2-/4-bit	Total #Sinks	#MBFFs 1-/2-/4-bit	Total #Sinks	#MBFFs 1-/2-/4-bit	Total #Sinks	#MBFFs 1-/2-/4-bit	Total #Sinks	#MBFFs 1-/2-/4-bit
C1	98	76/22/0	32	2/1/29	41	8/10/23	34	2/5/27	32	0/4/28
C2	423	366/57/0	127	4/8/115	156	24/36/96	128	0/16/112	123	0/6/117
C3	1692	1464/228/0	500	10/25/465	616	84/146/386	505	4/44/457	487	0/14/473
C4	5128	4378/751/0	1533	38/69/1426	1886	242/402/1211	1557	12/156/1389	1482	2/21/1459
C5	10 575	9150/1425/0	3119	70/133/2916	3820	480/920/2420	3148	6/287/2855	3018	2/33/2983
C6	169 200	146 400/22 800/0	49 951	1126/2213/46 612	60 880	7320/14 780/38 780	50 401	58/4715/45 628	48 070	18/113/47 939
Avg. ratio	100.00%	–	29.53%	–	36.02%	–	29.81%	–	28.44%	–

TABLE V

WIRELENGTH-ORIENTED FLIP-FLOP CLUSTERING COMPARISON

(A) Statistics of Loose Timing Circuits Provided by [16]

Circuit	#FFs	Chip Size (#Grids)	Bin Size (#Grids)	Grid Size Unit-Length^2	Max. Density (FF Area per Bin)	FF Cell Library (Bit Numbers)	FF Library Cells (Bit-Number, Power, Area)	Lower Bound	
								Power	Wirelength
S1	120	600 × 600	100 × 100	5 × 5	19 000	{1, 2, 4}	(1, 100, 100), (2, 172, 192), (4, 312, 285)	9360	43 545
S2	120	600 × 600	100 × 100	5 × 5	25 000	{1, 2, 4}	(1, 100, 100), (2, 172, 192), (4, 312, 285)	9360	43 260
T1	60 000	2000 × 3000	20 × 10	5 × 5	200 000	{1, 2, 4, 8}	(1, 100, 100), (2, 172, 192), (4, 312, 385), (8, 560, 725)	4 200 000	24 775 695
T2	5524	2000 × 2000	20 × 10	5 × 5	70 000	{1, 2, 4, 6, 13}	(1, 100, 100), (2, 172, 192), (4, 312, 385), (6, 450, 550), (13, 900, 1205)	382 427	1 489 635
T3	953	600×1600	20 × 10	5 × 5	70 000	{1, 2, 4, 4, 8}	(1, 100, 1000), (2, 172, 1920), (4, 312, 3850), (4, 299, 3980), (8, 560, 7250)	66 710	258 140

(B) Comparison on Power, Wirelength, and Runtime with [7] and [8] Based on the Benchmark Listed in (A)

Circuit	Initial		Modified [7]			[8]			Ours: Power-Oriented			Ours: WL-Oriented		
	Power	WL	Power Ratio	WL Ratio	Time (s)	Power Ratio	WL Ratio	Time (s)	Power Ratio	WL Ratio	Time (s)	Power Ratio	WL Ratio	Time (s)
S1	12 000	83 285	78.8%	119.2%	0.03	80.0%	85.1%	<0.01	78.5%	103.3%	<0.01	78.8%	85.8%	<0.01
S2	12 000	82 220	78.8%	117.4%	0.03	80.3%	97.1%	<0.01	78.3%	102.2%	<0.01	78.5%	84.1%	<0.01
T1	6 000 000	53 624 875	70.0%	137.5%	2128.93	70.2%	66.8%	785.69	70.0%	113.0%	4.49	70.0%	52.9%	3.92
T2	552 400	3 562 985	74.4%	84.8%	2.59	74.5%	74.4%	121.00	74.1%	74.2%	0.03	74.2%	72.6%	0.04
T3	95 300	576 710	71.0%	135.1%	0.27	71.6%	92.3%	0.40	70.2%	108.5%	0.02	70.3%	92.8%	0.02
Ratio					532.96			226.78			1.14			1.00
	Delta_P	Delta_WL	Delta_P	Delta_WL		Delta_P	Delta_WL		Delta_P	Delta_WL		Delta_P	Delta_WL	
S1	+2640	+39 740	+92	+55 730		+244	+27 370		+64	+42 520		+96	+27 930	
S2	+2640	+38 960	+92	+53 300		+272	+36 610		+32	+40 790		+64	+25 880	
T1	+1 800 000	+28 849 180	+1384	+48 979 780		+9184	+11 029 520		+0	+35 805 830		+0	+3 566 730	
T2	+169 973	+2 073 350	+28 463	+1 532 810		+29 251	+1 162 820		+26 923	+1 154 380		+27 391	+1 096 410	
T3	+28 590	+318 570	+955	+521 130		+1500	+274 040		+151	+367 760		+293	+277 170	
Ratio	71.97	6.27	1.11	10.24		1.45	2.51		0.98	7.49		1.00	1.00	

1. Delta_P (Delta_WL) means the amount of power (wirelength) exceeding the lower bound. Initial lists the result of the input circuit. Total power equals power lower bound plus Delta_P, while total wirelength equals wirelength lower bound plus Delta_WL. #Decision means the number of decision points.
2. Modified [7] is the revision of [7] to handle the discrete and finite MBFF library. Modified [7] and [8] are run on our platform.

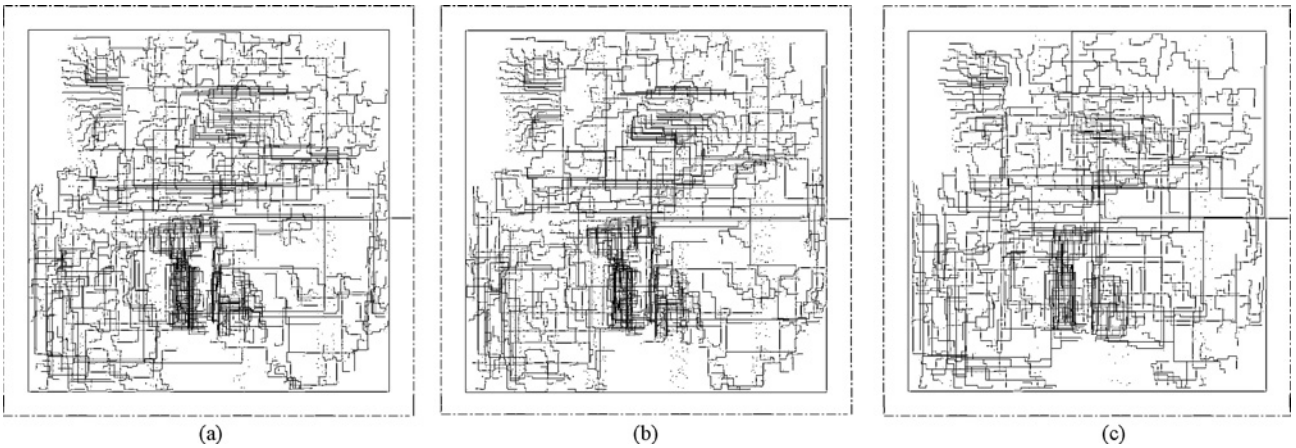

Fig. 12. RISC32: the clock tree is synthesized based on (a) pure single-bit flip-flops, (b) multibit flip-flops clustered by [4], and (c) multibit flip-flops clustered by INTEGRA.

includes two irredundant 4-bit MBFF cells. Since the timing slack is large, the intersection graph is very dense, especially for T1. The placement density constraint of T3 is relatively tight.

Table V(B) lists the comparison with [7] and [8] on flip-flop power (P), wirelength (WL) on data pins, and runtime (Time) by circuits listed in Table V(A). "Delta_P" (respectively, "Delta_WL") means the amount of power (wirelength) exceeding the lower bound, "Initial" lists the result of the input circuit, "Ratio" means the total extra power, extra wirelength, or total runtimes compared with our wirelength-oriented extension. It can be seen that our wirelength-oriented extension delivers the best wirelength, maintains almost the same flip-flop power as the power-oriented method does and achieves 533X and 227X speedups over modified [7] and [8], respectively. Hence, our wirelength-oriented extension is suitable for designs with loose timing slacks. In addition to larger power and/or wirelength, modified [7] and [8] also suffer from long runtimes because of numerous bins/grids.

C. Clock Tree Synthesis

The third experiment is conducted on an industrial design—a 32-bit RISC CPU (RISC32). To demonstrate the effectiveness of our postplacement MBFF clustering method (INTEGRA), we apply the experimental flow shown in Fig. 11. Logic synthesis is done by Synopsys Design Compiler [17]. Static timing analysis is reported by Synopsys Prime Time [17] and placement (including legalization) and clock tree synthesis are generated by Cadence SoC Encounter [18]. As listed in Table VI, the design without multibit flip-flop clustering is viewed as the baseline. The MBFF library used in [4] contains only single-bit and dual-bit flip-flop cells. Chen *et al.* [4] used register banking at logic synthesis to cluster flip-flops. We apply INTEGRA to the preclustered design at postplacement. 55% of the single-bit flip-flops in the initial design can be replaced by [4], while 95% can be replaced by INTEGRA. The lower bound of flip-flop power ratio is 94.75%, and INTEGRA reaches 95.02%. The MBFF clustering results pass the static timing check, so our timing model is reasonable, and the degradation by legalization is acceptable. Although the flip-flop power reduction is not large due to the limit of the used MBFF library, the total clock power reduction is still promising. According to the clock tree synthesis report, compared with [4], we gain 15.15% more clock power reduction (14.94% versus 30.09%). Moreover, the number of clock sinks is greatly reduced, the number of clock buffers is thus decreased to maintain the same clock skew, 300 ps. The induced wirelength on data pins almost does not affect the combinational power (only 1%–3% difference). Compared with MBFF clustering at logic synthesis [4], it can

TABLE VI
COMPARISON ON CLOCK TREE POWER

RISC32 CPU	Baseline	[4]	Ours
#Single-bit FFs	7279	3269	379
#Dual-bit FFs	0	2005	3450
FF replacement rate	0.00%	55.09%	94.79%
FF power ratio	100.00%	97.11%	95.02%
#Clock sinks	7279	5274	3829
Clock tree synthesis and power report			
Internal clock power (mw) A	0.3768	0.3469	0.3125
Leakage clock power (mw) B	0.013	0.011	0.008
Switching clock power (mw) C	1.893	1.584	1.275
Total clock power (mw) D = A + B + C	2.283	1.942	1.596
Total clock power reduction	0.00%	14.94%	30.09%
Total combinational power (mw) E	5.750	5.608	5.572
Whole chip power (mw) D + E	8.033	7.55	7.168
Whole chip power reduction	0.00%	6.01%	10.77%
#Clock subtrees	157	157	153
#Clock buffers	262	224	123
#Clock depth	7	9	7
Routed wirelength of clock tree (μm)	66643.0	60299.6	50469.3
Clock tree wirelength reduction	0.00%	9.52%	24.27%
Routed wirelength of data pins (μm)	295487.0	296491.0	301087.0
Routed wirelength ratio of data pins	100.00%	100.34%	101.90%

1. RISC32 CPU is a newer release than [4], gate count 120k.

2. 7279 single-bit flip-flops, chip size: 50 000×50 000 grids, bin size: 50×50 grids, grid size: 10×10 unit-length^2, Max. density (FF area per bin): 19 000, MBFF library: <(1, 36789.3, 6.4), (2, 69713.6, 12.16)>.

3. In clock tree synthesis report, total power includes dynamic (internal and switching) and static power. Clock power contains the flip-flop power and clock tree power. Power supply voltage is 0.9 V, the target clock skew is 300 ps. Routed wirelength is reported by SoC Encounter [18].

be seen that the power overhead on data pins is negligible, while power of the whole clock network can considerably be improved by postplacement MBFF clustering, even under timing and placement density constraints. Fig. 12 shows the clock tree synthesis results; it can be seen that our method generates a much simpler topology. Hence, MBFF clustering can not only save the power and area of flip-flops but also reduce the power and area of the whole clock network.

VII. Conclusion

In this paper, we presented a fast multibit flip-flop clustering algorithm for clock power saving. To resolve the time and space deficiencies encountered by recent works, we adopted coordinate transformation and expressed the feasible regions by two interval graphs. Our representation is a pair of linear-sized sequences. We directly manipulated intervals, clustering/placing flip-flops on the sequences. Utilizing the properties of interval graphs, we introduced the concept of decision points and further reduced the number of times of clustering applied. Our results show the concise representation brings an efficient data structure and an effective algorithm. Even under timing and placement density constraints, clock power saving still can be substantial at the postplacement stage using multibit flip-flops.

Acknowledgment

The authors would like to thank Dr. E. Y.-W. Tsai, Faraday Technology Corporation, Hsinchu, Taiwan, for his abundant support and providing the benchmarks. They would also like to thank Prof. M. P.-H. Lin, National Chung Cheng University, Chiayi, Taiwan, for providing them with the benchmarks and the binary code used in [8].

References

[1] L.-T. Wang, Y.-W. Chang, and K.-T. Cheng, Eds., *Electronic Design Automation: Synthesis, Verification, and Test*. Burlington, MA: Elsevier/ Morgan Kaufmann, 2009.

[2] J. M. Rabaey, A. Chandrakasan, and B. Nikolic, *Digital Integrated Circuits: A Design Perspective*, 2nd ed. Upper Saddle River, NJ: Prentice-Hall, 2003.

[3] M. Keating, D. Flynn, R. Aitken, A. Gibbons, and K. Shi, *Low Power Methodology Manual*. Berlin, Germany: Springer, 2007.

[4] L. Chen, A. Hung, H.-M. Chen, E. Y.-W. Tsai, S.-H. Chen, M.-H. Ku, and C.-C. Chen, "Using multi-bit flip-flop for clock power saving by DesignCompiler," in *Proc. Synopsys User Group (SNUG)*, 2010 [Online]. Available: http://www.synopsys.com.cn/information/snug/2010/using-multi-bit-flip-flop-for-clock-power-saving-by-designcompiler

[5] R. R. Pokala, R. A. Feretich, and R. W. McGuffin, "Physical synthesis for performance optimization," in *Proc. ASIC Conf.*, 1992, pp. 34–37.

[6] W. Hou, D. Liu, and P.-H. Ho, "Automatic register banking for low-power clock trees," in *Proc. ISQED*, 2009, pp. 647–652.

[7] J.-T. Yan and Z.-W. Chen, "Construction of constrained multi-bit flip-flops for clock power reduction," in *Proc. ICGCS*, 2010, pp. 675–678.

[8] Y.-T. Chang, C.-C. Hsu, M. P.-H. Lin, Y.-W. Tsai, and S.-F. Chen, "Postplacement power optimization with multi-bit flip-flops," in *Proc. ICCAD*, 2010, pp. 218–223.

[9] S.-H. Wang, Y.-Y. Liang, T.-Y. Kuo, and W.-K. Mak, "Power-driven flip-flop merging and relocation," in *Proc. ISPD*, 2011, pp. 107–114.

[10] J. Kleinberg and E. Tardos, *Algorithm Design*. Reading, MA: Addison-Wesley, 2006.

[11] G. Ramalingam and C. P. Rangan, "A unified approach to domination problems on interval graphs," *Inform. Process. Lett.*, vol. 27, no. 5, pp. 271–274, 1998.

[12] D. T. Lee, "Maximum clique problem of rectangle graphs," in *Advances in Computing Research*, vol. 1, F. P. Preparata, Ed. Greenwich, CT: JAI Press, 1983, pp. 91–107.

[13] Y.-P. Chen, J.-W. Fang, and Y.-W. Chang, "ECO timing optimization using spare cells," in *Proc. ICCAD*, 2007, pp. 530–535.

[14] M. Golin, R. Raman, C. Schwarz, and M. Smid, "Randomized data structures for the dynamic closest-pair problem," *SIAM J. Comput.*, vol. 27, no. 4, pp. 1036–1072, 1998.

[15] I. H.-R. Jiang and M.-H. Wu, "Power-state-aware buffered tree construction," in *Proc. IEEE ICCD*, Oct. 2008, pp. 21–26.

[16] *2010 CAD Contest of Taiwan* [Online]. Available: http://cadcontest.ee.ntu.edu.tw/cad10/Problems/B1_Faraday_091223_MultiBitFF.pdf.

[17] *Design Compiler/Prime Time*, Synopsys, Inc.

[18] *SoC Encounter*, Cadence Design Systems, Inc.

[19] N. Viswanathan, G.-J. Nam, J. A. Roy, Z. Li, C. J. Alpert, S. Ramji, and C. Chu, "ITOP: Integrating timing optimization within placement," in *Proc. ISPD*, 2010, pp. 83–90.

[20] C.-L. Chang, I. H.-R. Jiang, Y.-M. Yang, E. Y.-W. Tsai, and L. S.-F. Chen, "INTEGRA: Fast multi-bit flip-flop clustering for clock power saving based on interval graphs," in *Proc. ISPD*, 2011, pp. 115–121.

Iris Hui-Ru Jiang (M'07) received the B.S. and Ph.D. degrees in electronics engineering from National Chiao Tung University (NCTU), Hsinchu, Taiwan, in 1995 and 2002, respectively.

She was with VIA Technologies, Inc., West Seneca, NY, from 2002 to 2005. She is currently an Associate Professor with the Department of Electronics Engineering and the Institute of Electronics, NCTU. Her current research interests include very large scale integration physical design optimization and interaction between logic design and physical synthesis.

Dr. Jiang is a member of the ACM and the Phi Tau Phi Scholastic Honor Society.

Chih-Long Chang received the B.S. degree in electronics engineering from National Chiao Tung University (NCTU), Hsinchu, Taiwan, in 2011. He is currently working toward the M.S. degree in electronics from the Institute of Electronics, NCTU.

His current research interests include physical design optimization.

Yu-Ming Yang received the B.S. degree in electronics engineering from National Chiao Tung University (NCTU), Hsinchu, Taiwan, in 2008. He is currently pursuing the Ph.D. degree with the Institute of Electronics, NCTU.

His current research interests include very large scale integration physical design.