Improved Flop Tray-Based Design Implementation for Power Reduction

Andrew B. Kahng[†‡], Jiajia Li[‡] and Lutong Wang[‡]
[†]CSE and [‡]ECE Departments, UC San Diego, La Jolla, CA, USA
{abk, jil150, luw002}@ucsd.edu

ABSTRACT

Clock network power reduction is critical in modern SoC designs. Application of *flop trays* (i.e., multi-bit flip-flops) can significantly reduce the number of sinks in a clock network, and thus reduce the number of clock buffers, clock wirelength, and clock network power. Shared inverters within flop trays also reduce power at the flip-flop level. However, large-size flop trays typically induce placement and routing congestion, and impose additional placement constraints on their fanin/fanout logic cones; this results in power overheads on datapaths. At the same time, to our knowledge, few previous works have studied flop trays with more than four bits. The "chicken-and-egg" loop between flop tray generation and placement optimization is another challenge to flop tray-based design [7]. In this work, we propose an optimization flow to generate and place flop trays from a library of arbitrary given sizes and aspect ratios (ARs), to achieve clock network power reduction. Our optimization starts with an initial placement solution using only single-bit flops. It then performs capacitated K-means clustering to generate solutions with different flop tray sizes and ARs. More specifically, we iteratively use (i) min-cost flow to cluster flops, and (ii) a linear programming-based optimization to determine locations of the generated flop trays. Last, we formulate an integer linear program to select the best combination of flop tray solutions (i.e., sizes and placements) with minimum displacement and number of isolated sinks. Our optimization is aware of flop tray sizes and ARs, as well as timing-critical start-end pairs. Results in foundry 28FDSOI technology show up to 32% total block power reduction as compared to designs using only single-bit flops, up to 16% total block power reduction over designs with flop trays generated by logical clustering during synthesis, and 13% clock power reduction on average compared to the previous work in [10].

1. INTRODUCTION

Clock network optimization is critical in modern SoC designs due to the following reasons: (i) clock network typically has large power due to its high switching activity; (ii) clock skew and latency (with on-chip variation) have significant impact on design performance; and (iii) clock network routing consumes routing resources and can cause routing congestion. In this work, we study design optimization with *flop trays*[1] (i.e., macro cells of multi-bit flip-flops), where the application of flop trays can significantly reduce the number of sinks in (similar to [2]) and thus result in an improved clock network. Further, careful design of the internal routing within a flop tray prevents hold buffer insertion between flops within the tray, especially along scan chains. This reduces the number of hold buffers, DFT (Design for Test) overheads, and potential placement congestion.

Flop tray potential benefits. It is intuitively reasonable that more clock power reduction can be achieved by using larger sizes (i.e., greater number of bits) of flop trays. As a motivating "thought experiment", consider a clock tree with N sinks and fanout of f at each level: the total number of (internal) clock buffers between the clock root and the clock pins of sinks (i.e., flops, flop trays) is $\approx \frac{N-1}{f-1}$. If we could replace all single-bit flops with K-bit flop trays, the number of clock buffers would reduce to only $\approx \frac{N/K-1}{f-1}$ (e.g., using 64-bit flop trays to replace single-bit flops could reduce the number of clock buffers by up to 98.4% ($= \frac{N-N/64}{N-1} \approx \frac{63}{64}$)). Furthermore, Figure 1 illustrates how inverters for clock signals can be shared among flops in a flop tray, resulting in power and area reduction as compared to multiple single-bit flops. These power and area reductions would also increase with flop tray sizes.

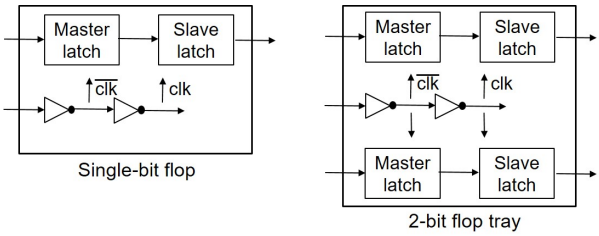


Figure 1: Two inverters for the clock signal are shared between the two flops in a 2-bit flop tray.

Current approaches and their limitations. Flop tray-based implementation is very challenging due to the following reasons. (1) In advanced nodes, flops (including single-bit flops and flop trays) typically occupy a large portion of the entire block area due to their large sizes.[2] Moreover, flop trays can have high aspect ratios (e.g., a 64-bit flop tray may be implemented as a 4 × 16 array of flops, with much greater width than height); flop tray size and shape have been ignored by previous literature on multi-bit flop optimization [14][15][21] and flop clustering [5][18]. Flop trays with large area and high aspect ratio make placement optimization very difficult [6][7]. (2) Clustering of flops imposes additional placement constraints on their fanin and fanout logic cones, which is highly likely to degrade the placement solution quality [7]. (3) Usage of flop trays can easily cause routing congestion. (4) Clustering of single-bit flops into flop trays has large impact on timing and limits the application of useful skew optimization. Most previous works study small-size flop trays, and do not fully address the above challenges in their optimization approaches. Crucially, further achievable benefits of using large-size flop trays are not exploited by previous works. To maximize obtained benefits from flop tray deployment, our present work proposes a flop tray-based optimization that comprehends arbitrary flop tray sizes. (Below, we show results with flop tray size up to 64 bits.)

A common practice for flop tray-based implementation is to cluster flops during the synthesis stage based on logic functions of the design, along with clock domain and clock gating information. We refer to this as *logical clustering* in the following discussion. However, flop tray generation without physical information can result in placement and routing congestion and degrade place-and-route (P&R) solution qualities. Figure 2 shows examples where flop tray-based implementations with logical clustering during synthesis stage can result in 8% – 39% wirelength overhead and 5% – 16%

[1]Terminology: A flop tray is synonymous with a multi-bit flip-flop (MBFF); we use "flop" as a synonym for "flip-flop".

[2]As an example, a minimum-size inverter occupies two placement sites; a single-bit flop occupies 18 sites; and a 64-bit flop tray can occupy 244 sites in width and four cell rows in height. Due to their large sizes, flops and flop trays can consume a substantial fraction of overall cell area (e.g., VGA from OpenCores [28] has 30% of its instances as flops, which accounts for 51% of the total cell area).

ICCAD'16, November 07-10, 2016, Austin, TX, USA.

http://dx.doi.org/10.1145/2966986.2967047.

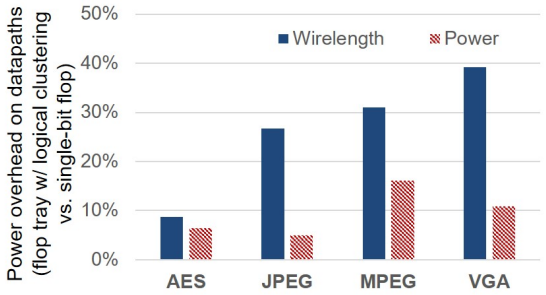


Figure 2: Wirelength and power overheads on datapaths due to flop tray-based implementations compared to implementations using only single-bit flops. Flop trays are generated based on logical clustering during synthesis with a commercial tool. Technology: 28FDSOI. Designs are from *OpenCores* [28]. Numbers of flops and flop trays in flop tray-based implementations, as percentages of flop numbers in implementations with single-bit flops, are 43%, 37%, 41% and 45% for AES, JPEG, MPEG and VGA, respectively.

power overhead on datapaths after detailed routing even at a low conversion ratio from single-bit flops to flop trays. This degrades power benefits from flop tray deployment. Therefore, feedback loops and iterations are required between early-stage flop clustering and P&R optimization, which can significantly increase design time [6]. Furthermore, although splitting large flop trays into smaller trays or single-bit flops during placement and/or routing can mitigate the congestion and power penalty, benefits of applying flop trays then become limited. In addition, the capability of logical clustering to realize flop tray benefits can be limited according to attributes of the given design. Designs with few multi-bit signals may not derive substantial benefits from flop tray deployment. On the other hand, designs with many multi-bit signals might use flop trays aggressively, with large-size flop trays in particular causing placement and routing congestion.

Our approach. In this work, we focus on post-placement flop tray optimization.[3] We first place the design with all single-bit flops, where the placement solution is considered to give *ideal* locations of individual flops and combinational cells (given that there are no additional constraints induced by flop clustering). We then cluster flops based on the placement solution. In this way, we resolve the "chicken-and-egg" loop between early-stage flop tray generation and placement optimization of flop trays. However, post-placement flop tray generation such as ours must carefully comprehend different flop tray sizes and aspect ratios; it must also minimize perturbation on datapath placement and timing degradation (otherwise, the assumption of "ideal" combinational cell placement does not hold).

To maximize the benefits of applying flop trays while minimizing the perturbation on the initial placement solution, we propose a *capacitated K-means optimization* which iteratively executes min-cost flow to cluster single-bit flops into flop trays, and a linear programming-based optimization to place flop trays. Based on the proposed capacitated K-means optimization, we achieve a solution (including flop clustering and flop tray placement) for each given flop tray size and AR. We then formulate an integer linear program (ILP) to select the best combination of flop tray solutions. In addition to minimization of displacement of flops (i.e., from the initial single-bit flop location to the flop location in a flop tray), our optimization is also aware of timing-critical start-end flop pairs. Specifically, we minimize the *relative location* displacement of timing-critical start-end pairs to minimize the timing impact from flop tray insertion.

The contributions of this paper are as follows.

- We propose a capacitated K-means iterative optimization that applies (i) min-cost flow based clustering, and (ii) LP-based placement optimization) to generate flop trays with various sizes (e.g., 4-bit, 16-bit and 64-bit) at the post-placement stage.
- Our optimization is aware of flop tray aspect ratios and *relative location* displacement of timing-critical start-end pairs.
- We apply a new *Silhouette*-based metric in addition to displacement distance to evaluate flop clustering solutions.
- Our optimization is able to convert more single-bit flops into flop trays, but with smaller datapath power overhead, as compared to a logical clustering flow implemented with commercial tools.
- We achieve up to 32% and 90% reductions of total block power and clock power as compared to implementations using only single-bit flops; and up to 16% and 40% reductions of total block power and clock power as compared to a commercial tool-based flow with logical clustering. We also achieve 13% clock power reduction on average compared to the previous work in [10].
- We evaluate the benefit (i.e., leakage reduction) of useful skew optimization on flop tray-based design and propose a useful skew-aware clustering to maximize such benefit.

The remainder of this paper is organized as follows. Section 2 reviews related works on flop tray optimization. Section 3 describes our capacitated K-means optimization flow. In Section 4, we describe our experimental setup and results. Section 5 concludes and gives directions for ongoing work.

2. PREVIOUS WORK

In this section, we review flop clustering and flop tray (multi-bit flop) generation approaches proposed in previous works. We classify these approaches into two categories: (i) early-stage flop tray generation, and (ii) flop tray generation during and/or after placement.

Several early works propose flop tray generation at early design stages. Kretchmer et al. [12] and Chen et al. [4] propose register banking during logic synthesis. They create Liberty models of flop trays, which can be used by logic synthesis tools. But, flop tray generation during synthesis has only logic topology as its main lever, and the lack of physical information can result in a sub-optimal clustering solution, degraded timing and larger power. To address this, Hou et al. [9] further propose register banking removal based on routing congestion and timing information. However, such a "(flop) clustering at early stage and (flop tray) removal at late stage" flow is not able to effectively exploit the benefits of flop tray usage. Thus, many other works propose flop tray generation during and/or after placement.

Yan et al. [23] generate flop trays at the post-placement stage. They first construct an intersection graph based on routing length and congestion constraints derived from an initial placement solution with single-bit flops. They then perform minimum-clique partitioning to reduce the number of flop trays. Lin et al. [13] use progressive window-based optimization to improve the methodology proposed in [23] considering given flop tray sizes. They solve the clustering problem by finding K-cliques and maximum independent sets in a merging graph constructed based on feasible-location regions of flops. Similarly, Wang et al. [21] use clique partitioning to identify a set of non-conflicting cliques. Jiang et al. [10] propose an efficient post-placement flop tray generation technique using interval graphs and a pair of linearized sequences. Liu et al. [15] also propose flop clustering based on an intersection graph. In addition to reducing the number of flop trays, they apply agglomerative clustering to minimize displacements of flops, wirelength and clock power. More recently, Lin et al. [14] develop a clock tree-aware in-placement flop tray generation technique. They build an intersection graph considering clock latency, wirelength and timing, then iteratively perform flop tray generation and timing-driven incremental placement. Xu et al. [22] propose an analytical clustering score for flop tray generation, permitting seamless integration with the traditional wirelength objective. Tsai et al. [20] propose to generate flop trays during placement. During

[3]Other low-power clocking styles and methodologies (e.g., pulsed-latch, register arrays, and rotary clock) are not the focus of this work.

analytical global placement, they guide placement of flops (to enable flop tray generation) with additional bonding force (resembling ionic bonds in chemistry). Other works optimize flop trays with awareness of crosstalk [8], clock gating [16], etc.

In addition to flop tray-based design, flop and/or latch clustering optimizations have been widely applied in previous works for clock tree and latch placement optimization. Mehta et al. [17] propose a clustering algorithm to obtain approximately load-balanced clusters and construct clock trees so as to minimize skew. Papa et al. [18] apply K-means clustering algorithm to minimize latch displacement during a physical synthesis optimization. Deng et al. [5] propose a register clustering methodology in generating the leaf-level topology of the clock tree to reduce clock power consumption.

We summarize our algorithmic and methodological improvements, compared to previous works, as follows.

- None of the previous in-placement and post-placement approaches study flop tray optimization with large-size flop trays (e.g., 64-bit flop trays). The ARs of flop trays are ignored (indeed, many previous works treat flop trays essentially as points in their optimizations). By contrast, our optimization considers arbitrary flop tray sizes and is aware of flop tray ARs.
- Most previous works assume a feasible displacement region for each flop. However, such an assumption does not comprehend the movements of fanin/fanout flops, which can be either pessimistic or optimistic. In addition, such an assumption essentially precludes exploiting benefits of useful skew. By contrast, our approach considers timing path-aware timing impact of flop displacement; specifically, we minimize the *relative location* displacement of timing-critical start-end pairs. We also propose a useful skew-aware optimization flow to maximize such benefit.
- Previous works use local search to cluster flops into flop trays. However, due to capacity constraints of flop trays, such local search can result in outliers with large displacement distances. By contrast, in this work we apply a more globally-aware optimization based on (i) a capacitated K-means formulation (with iterative min-cost flow-based clustering and LP-based placement optimization), and (ii) a practically scalable ILP-based matching and selection of flop tray solutions to globally optimize flop clustering with given capacity constraints (i.e., flop tray sizes).[4]

3. METHODOLOGY

We now describe our optimization methodology for flop tray generation and placement. Figure 3 illustrates our overall optimization flow, where we integrate our flop tray optimization (steps in blue boxes) into a conventional SP&R (synthesis, place, and route) flow. To address the "chicken-and-egg" loop between flop tray generation and placement optimization, we first perform an initial placement with only single-bit flops, where the placement is considered to be "optimal" with no placement constraints induced by flop clustering. We note that since the initial placement is timing- and congestion-aware, minimizing subsequent perturbations can mitigate potential congestion due to flop trays, as well as minimize timing impacts. Further, to comprehend multiple flop tray sizes and ARs, we perform flop tray optimization for each flop tray choice (i.e., a {size, AR} combination). Last, we perform an integer linear programming (ILP)-based optimization to select the optimal combination of flop trays and their placement solutions.[5]

[4]Our ILP runtime (CPLEX 12.6) is less than one minute on the VGA testcase [28] (with 17K flops and 1000 timing-critical paths) with five candidate flop tray sizes studied in Section 3.2 and Section 4 below, using 20 threads on a 2.5GHz Intel Xeon server.

[5]Our separate study shows that due to high runtime complexity, it is practically infeasible for our current approach to optimize flop clustering and flop tray placement considering all possible flop tray candidate sizes simultaneously. We therefore perform a two-step optimization in this work.

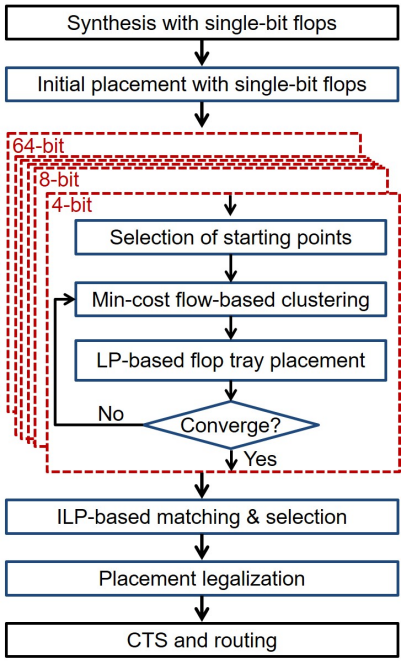


Figure 3: Overall optimization flow of flop tray generation.

We state our post-placement flop tray generation problem as: **Given** an initial placement solution with only single-bit flops, flop tray choices, and timing constraints, **cluster** single-bit flops into flop trays and **determine** the placement location of each flop tray, **such that** total block power (including clock power and power of sequential cells (i.e., flops and flop trays) and combinational cells) is minimized after routing.

The following subsections describe our capacitated K-means clustering and our ILP-based selection of flop tray solutions. Table 1 lists the notations used in our discussion.

Table 1: Description of notations used in our formulation.

Term	Meaning
t_i	i^{th} flop tray
e_i	binary indicator whether t_i is used
w_i	cost of using tray t_i
f_{ij}	j^{th} flop of t_i
h_l	l^{th} single-bit flop
$b_{l,ij}$	binary indicator whether h_l is matched to f_{ij}
(X_i, Y_i)	center location of t_i
(x'_{ij}, y'_{ij})	relative center location of f_{ij} w.r.t. the center of t_i
(x_l, y_l)	optimal location of h_l
$(d_{l,ij}, d_{l,ij})$	Manhattan distance between h_l and f_{ij}

3.1 Capacitated K-Means Clustering

We first address the following, narrower problem: **Given** an initial placement solution with all single-bit flops (i.e., N single-bit flops), and $\lceil N/K \rceil$ K-bit flop trays with fixed AR, **cluster** the single-bit flops into flop trays and **determine** the placement location of each flop tray, **such that** the total displacement of flops is minimized.

To address this problem, we propose a capacitated K-means algorithm [11]. (As noted above, K-means clustering algorithms have also been applied to flop (or latch) clustering in previous works [5][18].) There are two steps in a standard K-means algorithm: (i) clustering, and (ii) updating the center location of each cluster. We associate these two steps with: (i) matching of single-bit flops to flop slots in flop-trays, and (ii) updating the locations of flop trays. We propose a min-cost flow to address (i), and a linear programming (LP)-based optimization to address (ii). We iterate between these two steps until convergence (i.e., no further displacement reduction can be achieved, or a maximum number of iterations (= 35 in our experiments below) is reached).

In our capacitated K-means clustering, we use an algorithm that is similar to K-means++ [3] to select the starting points. Selection of $\lceil N/K \rceil$ starting points for clustering is described in Algorithm 1. In Algorithm 1 we calculate center-to-center distances between single-bit flops. To comprehend the aspect ratio of flop trays, we scale the horizontal distance by (1/AR) (= height/width) of the given flop tray.

Algorithm 1 Selection of starting points.

1: Randomly select one flop among single-bit flops
2: For each flop h_l, calculate the total Manhattan distance (d_l) from h_l to all selected flops
3: Randomly select one new flop with probability d_l
4: Repeat Steps 2 and 3 until $\lceil N/K \rceil$ flops are selected

These selected starting points serve as initial locations of flop trays. We then apply a min-cost flow to achieve *capacitated clustering* of flops. Our min-cost flow is illustrated in Figure 4. To construct the flow instance, we create a node for each single-bit flop h_l. For each flop tray t_i, we further create K nodes for its K slots, $f_{i1} \ldots f_{iK}$. For each edge between a pair of h_l and f_{ij}, we set its capacity as 1 and its cost as the Manhattan distance between h_l and f_{ij}. Here, we directly calculate the Manhattan distance between single-bit flops and flop slots without any scaling. Finally, we create one source and one sink, and assign edges connected to them with capacity as 1 and cost as 0, as illustrated in Figure 4. Notice that by considering the distances between the locations of single-bit flops and flop slots in flop trays, our min-cost flow optimization is explicitly aware of physical information (in particular, dimensions and ARs) of the given flop trays.

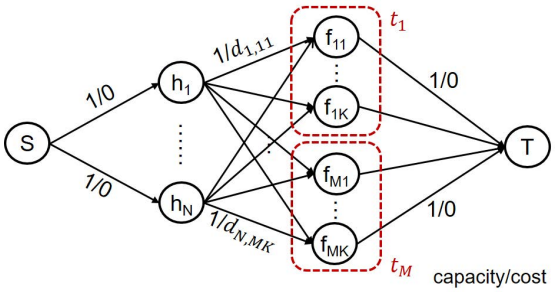


Figure 4: Example of min-cost flow with K-bit flop trays.

Based on the capacitated K-means clustering solution from the min-cost flow, we formulate a linear program (shown as follows) to determine the flop tray locations that achieve minimum total displacement of flops. These placement locations of flop trays will serve as starting points for the next iteration of clustering.

$$Minimize \quad D \tag{1}$$

$$Such\ that\ |X_i + x'_{ij} - x_l| + |Y_i + y'_{ij} - y_l| = d_l \quad \forall h_l \tag{2}$$

$$\sum_l d_l = D \tag{3}$$

Constraint (2) calculates the displacement for each flop (d_l), and the objective seeks to minimize the total displacement over all flops.

We iterate between the min-cost flow-based clustering and the LP-based flop tray placement until no further displacement reduction is achievable (i.e., no flop trays move between two consecutive iterations).

To confirm benefits from awareness of flop tray ARs, we show in Figure 5 representative clustering solutions from (i) the classic K-means approach, which treats each flop tray as a point, and (ii) our min-cost flow-based clustering, which is aware of flop tray ARs. We observe that our clustering solution more closely matches the AR of given flop trays. Further, classic K-means without awareness of flop tray AR can result in 2× increase in average displacement from the "ideal" single-bit flop placement; this is likelier to incur datapath power and timing overheads.

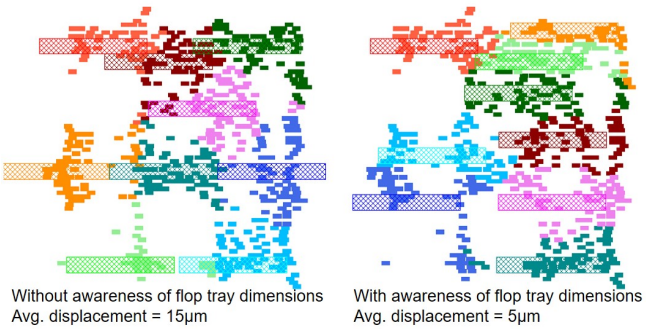


Figure 5: Clustering solutions into 64-bit flop trays (i) without awareness of flop tray aspect ratio and dimensions, and (ii) with awareness of flop tray aspect ratio and dimensions. Design: AES (530 single-bit flops). Technology: 28FDSOI.

In our capacitated K-means algorithm, as with K-means approaches in general, the selection of starting points has a strong impact on the final solution quality. We adapt the Silhouette metric [19] and use Equation (4) to evaluate the solution quality of generated starting points.[6]

$$func(h_l) = \frac{min_{i' \neq i, j'}(d_{l,i'j'}) - d_{l,ij}}{max(d_{l,ij},\ min_{i' \neq i, j'}(d_{l,i'j'}))} \tag{4}$$

where h_l is matched to f_{ij}. The dissimilarity within a cluster is measured by the displacements of each of the cluster's assigned flops h_l. The dissimilarity between a given cluster and other clusters is measured by the distances between assigned flops h_l and the nearest flop-tray slot in another cluster to which h_l is not assigned.

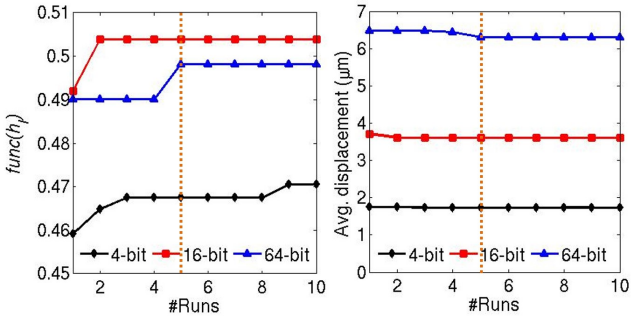


Figure 6: Best clustering solution (i.e., $func(h_l)$ (left) and displacement (right)) with multiple runs (numbers of runs are shown in the x-axis).

We apply a multistart strategy to improve the selection of starting points. Multiple runs (five in our experiments) of the procedure in Algorithm 1 are each followed by a small number (15 in our experiments) of iterations between the min-cost flow and LP-based placement optimization. We then select the solution with the highest average $func(h_l)$ value and proceed with capacitated K-means iterations until convergence. Figure 6 shows a typical improvement of the average value of $func(h_l)$ (left) and the average displacement (right) with increased number of runs. In our studies, the improvement of $func(h_l)$ and displacement typically saturates after five runs. Thus, the experiments reported below apply five multistarts to mitigate the impact of starting point selection.

[6] As presented in [19], the Silhouette value is a measure of how similar an object is to its own cluster, compared to other clusters. A general Silhouette value is defined as $s(i) = \frac{b(i)-a(i)}{max(a(i),b(i))}$, where $a(i)$ is the average dissimilarity (e.g., average distance) of i with all other data within the same cluster, and $b(i)$ is the lowest average dissimilarity (e.g., minimum average distance) of i to the data in any other cluster other than its own. By definition, $-1 \leq s(i) \leq 1$, and a larger Silhouette value indicates a better clustering solution. In this work, data are slots of flop trays, and dissimilarities are measured by distances.

3.2 ILP-Based Matching Optimization

The next step of our optimization approach addresses the following problem: **Given** candidate flop trays with various capacities, each with a fixed placement location, **select** the optimal subset of the candidate flop trays, and **determine** a mapping of single-bit flops into slots of selected candidate flop trays, **such that** (i) every single-bit flop is mapped to a slot of a selected flop tray (including flop trays with one bit, i.e., no clustering), and (ii) a weighted sum of the total displacement of flops, relative displacement of timing-critical start-end pairs, and total flop tray costs is minimized.

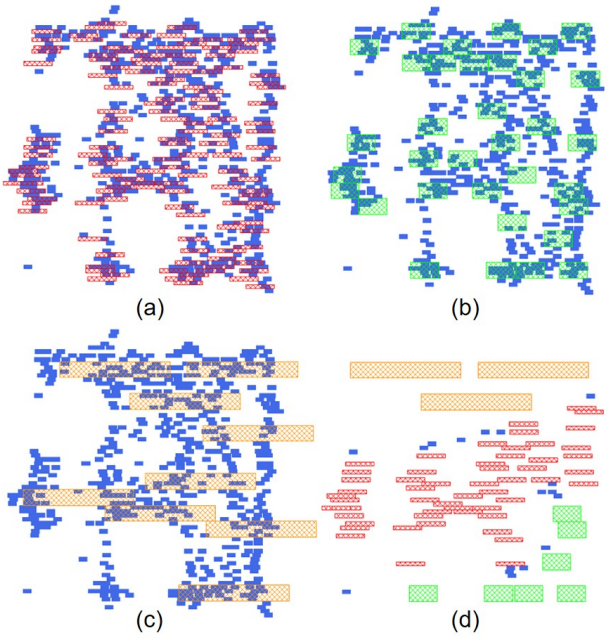


Figure 7: Example of our ILP-based optimization. Inputs: (a) solution with only 4-bit flop trays (flop trays are in red, #flop trays = 133, average displacement = 2μm), (b) solution with only 16-bit flop trays (flop trays are in green, #flop trays = 34, average displacement = 3μm), and (c) solution with only 64-bit flop trays (flop trays are in orange, #flop trays = 9, average displacement = 5μm). Output: (d) solution with a combination of single-bit flops and 4-bit, 16-bit and 64-bit flop trays (#flops + #flop trays = 81, average displacement = 2μm).

As discussed in Section 3.1, we run capacitated K-means clustering with different flop tray sizes and ARs, and use these flop trays together with their optimized placement locations as inputs ("candidates") for an ILP-based matching optimization. Our ILP-based optimization selects an optimal subset of candidate flop trays with various flop tray sizes as our final solution. As an example, Figures 7(a)-(c) show solutions of flop trays with fixed sizes and ARs on the AES testcase. Figure 7(d) shows the final solution. Our objective is to minimize a weighted sum of total displacement of flops, relative displacement of timing-critical start-end flop pairs, and total flop tray cost. Relative displacement of a timing-critical start-end flop pair is illustrated in Figure 8. As an improvement to previous approaches, we comprehend timing impact of flop tray generation considering timing-critical paths (i.e., start-end pairs). Specifically, if the flop tray generation moves two flops towards each other, combinational cells in the logic cone between the flops are forced to be placed in a more compact region, which results in congestion and distortion of the placement and routing. Alternatively, if the flop tray generation moves two flops away from each other, timing paths between the two flops will tend to have longer wirelength, degrading timing. We therefore seek to minimize the *relative displacement* of flops that are timing-critical start-end pairs.

Our ILP to select the optimal combination of flop tray solutions with various sizes and ARs is given below.[7]

$$Minimize \quad \alpha \cdot W + D + \beta \cdot Z \tag{5}$$

$$Such\ that\ |\sum_{ij}(X_i + x'_{ij} - x_l) \cdot b_{l,ij}|$$
$$+|\sum_{ij}(Y_i + y'_{ij} - y_l) \cdot b_{l,ij}| = d_l \quad \forall l \tag{6}$$

$$\sum_{l} d_l = D \tag{7}$$

$$d_l \leq d_{max} \quad \forall l \tag{8}$$

$$|\sum_{ij}(X_i + x'_{ij} - x_l) \cdot b_{l,ij} - \sum_{i'j'}(X_{i'} + x'_{i'j'} - x_{l'}) \cdot b_{l',i'j'}|$$
$$+|\sum_{ij}(Y_i + y'_{ij} - y_l) \cdot b_{l,ij} - \sum_{i'j'}(Y_{i'} + y'_{i'j'} - y_{l'}) \cdot b_{l',i'j'}|$$
$$= z_{ll'} \qquad \forall (h_l,\ h_{l'}) \in \text{timing-critical paths} \tag{9}$$

$$\sum_{(h_l,h_{l'}) \in cri\_paths} z_{ll'} = Z \tag{10}$$

$$z_{ll'} \leq d_{max} \quad \forall (h_l,\ h_{l'}) \in \text{timing-critical paths} \tag{11}$$

$$b_{l,ij} \leq e_i \qquad \forall l,\ j \tag{12}$$

$$e_i \leq \sum_{lj} b_{l,ij} \quad \forall i \tag{13}$$

$$\sum_{i} w_i \cdot e_i = W \tag{14}$$

$$\sum_{l} b_{l,ij} \leq 1 \quad \forall j \tag{15}$$

$$\sum_{i,j} b_{l,ij} = 1 \quad \forall i \tag{16}$$

Here, W is the total cost of selected flop trays, which is determined based on their power consumption and sizes (i.e., number of bits); D is the total displacement over all flops; Z is the total relative displacement over all timing-critical start-end flop pairs; and α and β are weighting parameters. Constraints (6) and (7) calculate the total displacement of all flops. Constraint (8) bounds the maximum displacement of each flop. Constraints (9) and (10) calculate the total relative displacement of timing-critical start-end flop pairs (i.e., $(h_l,\ h_{l'})$). Constraint (11) bounds the maximum relative displacement of each timing-critical start-end flop pair. Constraints (12) and (13) force the binary indicator variable e_i to be 1 if the corresponding flop tray is used, and 0 otherwise. Constraint (14) calculates the total cost of selected flop trays. Constraints (15) and (16) ensure that each flop is matched to exactly one slot, and that each slot is matched to at most one flop. We note that additional mutual exclusion constraints can avoid placement overlaps between pairs of flop trays (e.g., $e_i + e_j \leq 1$ if there is overlap between the i^{th} and j^{th} flop trays). However, such mutual exclusion constraints might limit the solution space and thus degrade the solution quality. We therefore perform placement legalization in the commercial P&R tool to remove overlaps among flop trays.[8] We also note that although an ILP-based optimization typically has large runtime, in our formulation, the number of binary variables is only $O(N \cdot Q)$, where N is the number of flops and Q is the number of candidate flop tray choices (i.e., sizes and dimensions). In practice, our method exhibits practically reasonable runtimes (see Footnote 4 above).

To give an understanding of how the weighting parameters α and β affect solution quality, Figure 9 shows the number of flop trays and the average flop displacement resulting from optimization with various α values. We observe that more large-size flop trays are selected with an increased value of α, so as to minimize the total tray costs. Such selection of large-size flop trays will reduce power of flop trays as well as the clock power. However, the average flop displacement increases with the value of α, and this can incur datapath power

[7]Note that our ILP can be extended to be aware of clock gating, clock domain and useful skew optimization, etc. with additional constraints. Section 4.3 briefly describes a useful skew-aware extension and corresponding benefits.

[8]Our experimental results show no more than three sites displacement on average per flop tray during the placement legalization.

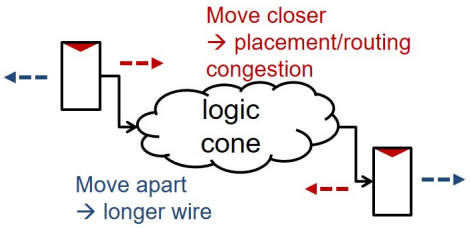


Figure 8: Illustration of the timing impact due to relative displacement between timing-critical start-end flop pairs.

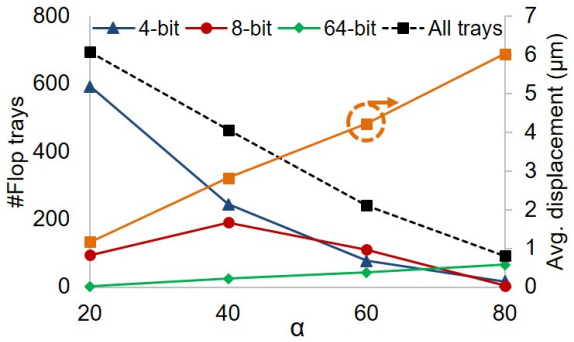


Figure 9: Number of flop trays and average displacement of flops change with different α values. Each column is an implementation with corresponding α. Black-dotted curve indicates the total number of flops and flop trays. Orange curve indicates the average displacement over all flops. (Small) numbers of 16- and 32-bit flop trays omitted for figure clarity. Design: JPEG. Technology: 28FDSOI.

overhead. Therefore, the choice of α determines a tradeoff point between (i) clock power reduction and power reduction of flop trays, versus (ii) the power overhead on datapaths. In our experiments, we empirically set $\alpha = 20$, 40, 60 and 80. We then select the solution with the minimum total block power from these four runs.

To evaluate the impact of β, we uniformly place flop trays within the block area and fix their locations. The number of flop trays is determined according to the number of flops, such that no flop tray can be empty, which eliminates the impact of W in our objective function. We then perform an ILP-based matching optimization to cluster flops into flop trays. Figure 10 shows the total block power of the AES and JPEG testcases implemented with various β values. We observe reduced block power with $\beta > 0$, where our optimization minimizes the relative displacement between timing-critical start-end flop pairs. This confirms the benefits of minimizing the relative displacement between timing-critical start-end flop pairs. We also observe increased block power with a large β value. This is because with a large β value, relative displacements between timing-critical start-end flop pairs dominate our objective function. The resultant large displacements of non-timing critical flops incur datapath power penalty. We empirically use $\beta = 1$ in our experiments.

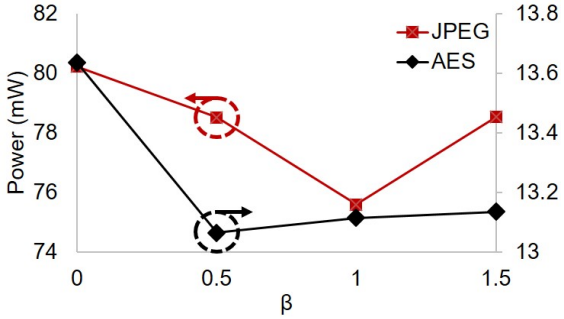


Figure 10: Power change with various β values. Designs: AES, JPEG. Technology: 28FDSOI.

4. EXPERIMENTAL RESULTS

We perform experiments in a 28nm FDSOI foundry technology with dual-VT libraries. We use four design blocks (AES, JPEG, MPEG, VGA) from OpenCores [28] as our testcases. Parameters of these four testcases are shown in Table 2. We scale flop tray power and area based on the ratios shown in Table 3. Layout ARs of flop trays are also shown in Table 3. We synthesize designs using *Synopsys Design Compiler vI-2013.12-SP3* [29] and then place and route using *Cadence Innovus Implementation System v15.2* [24]. We set the placement density at the floorplan stage as 70%. We also perform timing and power analyses using *Cadence Innovus Implementation System v15.2*. We perform vectorless power simulation with a default switching activity of 10% at primary inputs. Our optimization flow is implemented in C++. We use *CPLEX v12.6* [25] as our ILP solver and LEMON [27] as our min-cost flow solver. Functions used in P&R tools are implemented in Tcl. We conduct our experiments on a 2.5GHz Intel Xeon server.

Table 2: Testcase parameters.

design	#inst	#flops	clock period
AES	~12K	530	600ps
JPEG	~47K	4512	600ps
MPEG	~13K	3181	500ps
VGA	~56K	17053	700ps

Table 3: Normalized flop tray area and power, and layout AR.

tray size	4-bit	8-bit	16-bit	32-bit	64-bit
norm. area/power per bit	0.875	0.854	0.854	0.844	0.844
AR (#rows×#columns)	1×4	2×4	4×4	4×8	4×16
AR (#rows×#sites)	1×63	2×62	4×62	4×122	4×244

4.1 Comparison to Other Methods

To evaluate the performance of our proposed methodology, we compare our solutions to three reference flows: (i) the conventional implementation flow with only single-bit flops (*ref_1b*), (ii) a flop tray-based implementation flow which generates flop trays during commercial synthesis based on logical clustering, followed by conventional commercial P&R optimization (*ref_mb1*), and (iii) a flop tray-based implementation flow which generates flop trays at the post-placement stage using the method proposed in [10], followed by clock tree synthesis and routing (*ref_mb2*). No value judgment or "benchmarking" regarding any commercial tool is intended by, or should be inferred from, our present discussion.

Table 4 shows results evaluated at the post-routing stage. Figure 11 shows the layouts of placement solutions with single-bit flops and optimized flop trays. We observe that our proposed optimization (*opt_mb*) is able to significantly reduce the number of sinks with application of flop trays (e.g., we reduce the number of sinks by 98% on the VGA testcase compared to the implementation using only single-bit flops). The reduction in number of sinks results in smaller clock power: our optimization reduces clock power by up to 90% and 40% compared to implementations with single-bit flops and flop trays generated by logical clustering, respectively. Our flop tray generation also results in reduced power on flops. Moreover, we observe that although our optimization has large conversion ratio from single-bit flops to flop trays, the incurred datapath power and wirelength penalties are small as compared to the implementation with logical clustering. This strongly suggests that our approach of optimization with minimum perturbation from a "good" initial placement solution forestalls placement and routing congestion while also minimizing the datapath power penalty from application of flop trays. For the MPEG testcase, our optimization actually results in smaller datapath power as compared to the "ideal" implementation with single-bit flops; we believe this is likely due to reduced placement density (i.e., usage of flop trays reduces total area of flops).

Our optimization (*opt_mb*) also achieves up to 7% total block power reduction compared to the previous work [10] (*ref_mb2*). Since *ref_mb2* only uses up to 8-bit flop trays, we limit the flop

Table 4: Experimental results.

design	flow	power (mW)				#flops						#clk bufs	WNS (ps)	area (μm^2)	WL (μm)	#inst
		comb	seq	clk	sum (norm)	1	4	8	16	32	64					
AES	*ref_1b*	**8.11**	**4.37**	**1.53**	**14.02 (1.00)**	530	0	0	0	0	0	11	-11	10362	140	12002
	ref_mb1	**8.64**	**4.00**	**0.72**	**13.35 (0.95)**	198	5	19	2	2	1	0	9	10606	153	11730
	ref_mb2	**8.14**	**4.05**	**0.43**	**12.62 (0.90)**	34	56	34	0	0	0	3	-4	10122	140	11595
	opt_mb	**8.15**	**3.94**	**0.46**	**12.56 (0.90)**	59	22	46	1	0	0	4	-5	10171	139	11619
	opt_mb'	**8.09**	**3.98**	**0.54**	**12.60 (0.90)**	80	41	36	0	0	0	4	-2	10160	137	11598
JPEG	*ref_1b*	**35.13**	**36.04**	**13.37**	**84.54 (1.00)**	4512	0	0	0	0	0	115	1	47595	420	47567
	ref_mb1	**36.88**	**33.21**	**6.10**	**76.20 (0.90)**	1388	109	84	70	0	14	59	0	46374	531	44246
	ref_mb2	**35.45**	**32.06**	**4.56**	**72.07 (0.85)**	308	457	297	0	0	0	40	-1	45888	437	44094
	opt_mb	**35.68**	**31.28**	**2.28**	**69.24 (0.82)**	274	77	110	2	9	43	25	1	45535	460	43545
	opt_mb'	**35.64**	**31.85**	**3.12**	**70.62 (0.84)**	83	37	537	0	0	0	28	1	45898	428	43607
MPEG	*ref_1b*	**5.88**	**28.93**	**10.72**	**45.53 (1.00)**	3181	0	0	0	0	0	92	-17	18169	149	12291
	ref_mb1	**6.52**	**26.99**	**5.19**	**38.70 (0.85)**	1225	27	17	15	18	14	53	-34	17757	195	10079
	ref_mb2	**6.03**	**25.62**	**3.30**	**34.95 (0.77)**	161	381	187	0	0	0	29	-11	17136	159	9849
	opt_mb	**5.66**	**25.12**	**0.98**	**31.76 (0.70)**	120	9	2	3	1	46	15	-3	16666	176	9183
	opt_mb'	**5.65**	**25.33**	**2.24**	**33.22 (0.73)**	77	16	382	0	0	0	21	-23	16780	149	9531
VGA	*ref_1b*	**14.32**	**108.34**	**42.19**	**164.84 (1.00)**	17053	0	0	0	0	0	361	-5	88015	960	56039
	ref_mb1	**16.63**	**101.63**	**20.73**	**138.99 (0.84)**	7325	42	77	75	50	96	215	-2	84537	1337	45793
	ref_mb2	**14.60**	**94.51**	**10.24**	**119.35 (0.73)**	129	1299	1466	0	0	0	110	-2	80710	1032	41656
	opt_mb	**15.29**	**93.99**	**2.04**	**111.32 (0.68)**	33	1	6	0	2	266	28	3	80083	1132	39129
	opt_mb'	**14.33**	**94.29**	**8.41**	**117.03 (0.71)**	56	51	2114	0	0	0	89	-13	80538	1001	40909

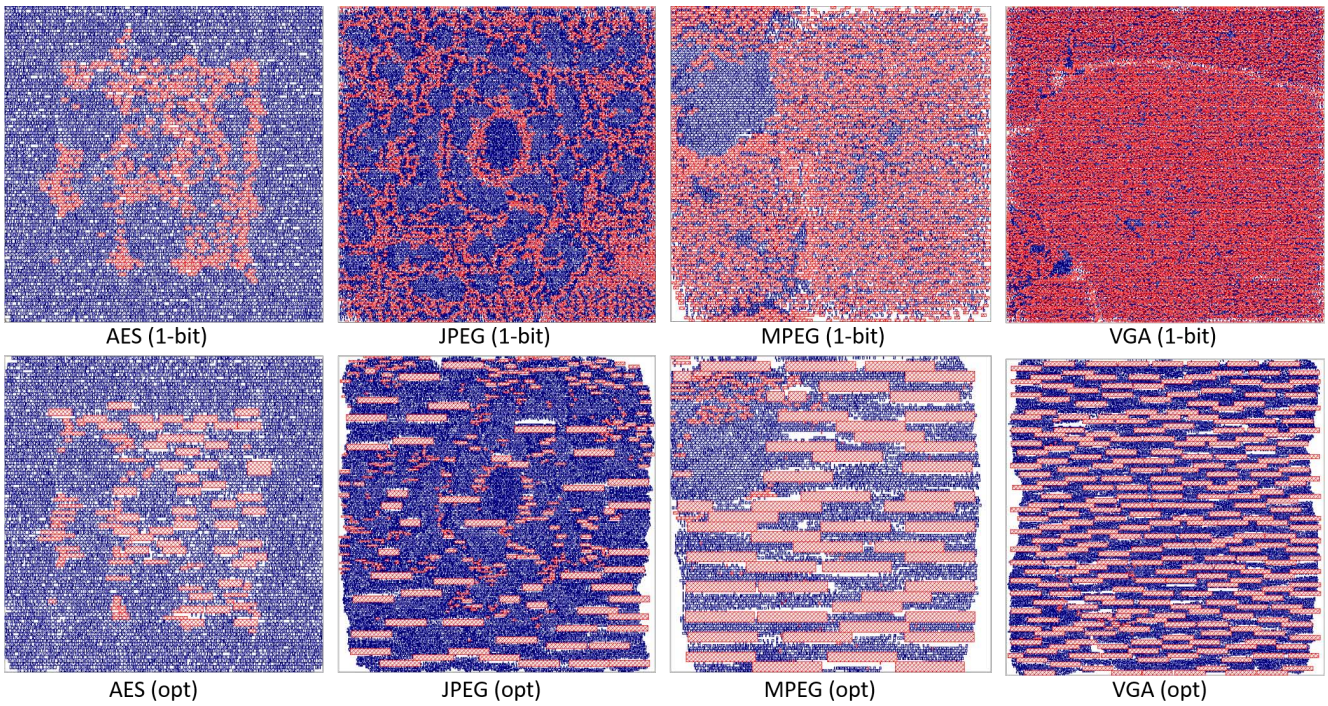


Figure 11: Layout comparison between implementations with only single-bit flops and with optimized flop trays. In the flop tray-based solutions, the candidate flop tray sizes are 4-bit, 8-bit, 16-bit, 32-bit and 64-bit.

tray options to 4-bit and 8-bit flop trays in *opt_mb'* for a fair comparison. Table 4 shows that with the same set of flop tray options our optimization achieves 13% clock power reduction on average compared to *opt_mb'*, along with smaller datapath power for most of the testcases (the exception is the JPEG testcase with <1% power overhead).

4.2 Optimization with Various Flop Tray Sizes

We further perform flop tray optimization with various combinations of flop tray sizes. More specifically, we implement designs with (i) single-bit flops only, (ii) {4-bit} flop trays, (iii) {4-bit, 8-bit} flop trays, (iv) {4-bit, 8-bit, 16-bit} flop trays, and (v) {4-bit, 8-bit, 16-bit, 32-bit, 64-bit} flop trays with various α values (i.e., 20, 40, 60, 80). We note that setups (ii)-(v) can also use single-bit flops. For each setup, we select the minimum total block power solution with <5% power penalty on datapaths as compared to the case with only single-bit flops. Figure 12 shows flop power and clock power, normalized to implementations using only single-bit flops. We observe that with only 4-bit flop trays, our optimization achieves >7% power reduction on flops and flop trays. However, including larger flop trays does not afford much further reduction of flop power. (This may be due to our conservative assumptions regarding power-per-bit in larger flop trays, as shown in Table 3). On the other hand, application of large-size flop trays can effectively reduce clock power. For example, optimizations with {16-bit, 32-bit, 64-bit} flop trays achieve 11% more clock power reduction on average as compared to the cases with only {4-bit, 8-bit} flop trays.

4.3 Study of Useful Skew Optimization with Flop Trays

Last, we evaluate the benefits of useful skew optimization in terms of leakage power reduction on (i) designs with only single-bit flops (*ref_1b*), and (ii) flop tray-based designs (*opt_mb* as shown in Figure 13.[9] Based on the approach proposed in [1], we formulate the useful skew optimization as a maximum mean weight cycle problem and apply iterative shortest path search to maximize the average endpoint slack. We then perform leakage power optimization using a commercial tool [24], i.e., we exploit the increased timing slacks for leakage power reduction. We observe from Figure 13 that due to clustering

[9]In the technology we use, we do not observe significant dynamic power benefits from useful skew optimization. We therefore study leakage power reduction from useful skew optimization in this experiment.

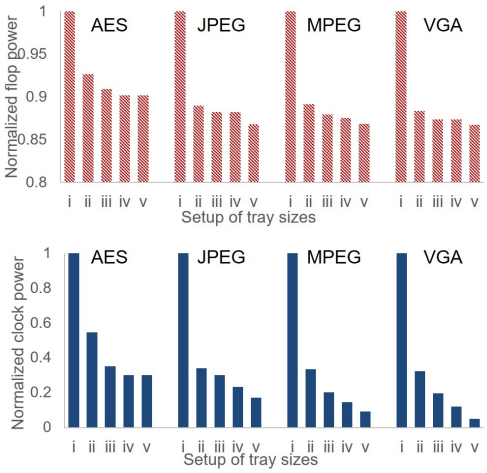
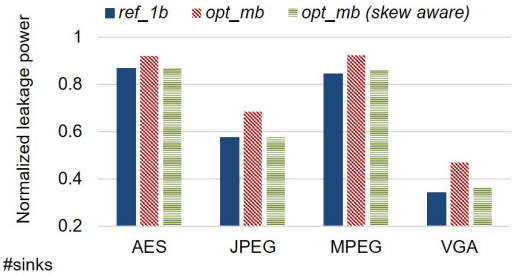


Figure 12: Flop (tray) power and clock power of designs with various flop tray sizes. Candidate tray sizes are 4-bit, 8-bit, 16-bit, 32-bit and 64-bit.

of endpoints, flop tray-based designs have 9% less leakage power reduction on average across four designs as compared to cases with only single-bit flops. To reduce the impact of flop tray generation on benefits from useful skew optimization, we study skew-aware flop tray generation that only allows clustering of flops with desired skew less than θ (we use θ = 20ps in our experiments). Figure 13 shows that the skew-aware clustering (*opt_mb (skew aware)*) can achieve similar leakage power reduction as compared to the cases with only single-bit flops (green vs. blue bars), but at the cost of more sinks.



#sinks	AES	JPEG	MPEG	VGA
ref_1b	530	4512	3181	17053
opt_mb	128	515	181	308
opt_mb (skew aware)	392	1830	205	1245

Figure 13: Datapath leakage power results, normalized to implementations with only single-bit flops. Useful skew-aware flop tray optimization is able to achieve similar leakage power reduction as compared to the optimized design with only single-bit flops (green skew-aware multi-bit vs. blue reference single-bit bars), but with an average of 21% less reduction in number of sinks.

5. CONCLUSION

In this work, we present a novel flop tray-based optimization for improved design power reduction. We propose a capacitated K-means algorithm which iteratively applies a min-cost flow-based clustering and a LP-based flop tray placement. We also propose an ILP-based matching optimization to generate flop trays while minimizing the perturbation to the initial placement solution. Our work achieves several improvements as compared to previous works: (i) awareness of flop tray aspect ratio and (large) size; (ii) explicit minimization of relative displacement of timing-critical start-end flop pairs; and (iii) global optimization instead of local search. The proposed techniques allow us to achieve up to 32% total block power reduction as compared to designs with only single-bit flops, and up to 16% total block power reduction over designs with flop trays generated by logical clustering during synthesis. We also achieve 13% clock power reduction on average compared to the previous work in [10]. We further study the impact of flop tray sizes on optimization solution quality. Finally, we study useful skew optimization in the context of our flop tray-based designs. Our future works include (i) scalable optimization considering all flop tray candidate sizes simultaneously; (ii) awareness of IR-drop in the flop tray clustering and placement; and (iii) floorplan blockage-aware and routing congestion-aware flop tray generation.

Acknowledgments

We are very grateful to the authors of [10] for providing binary of their optimizer for use in our study.

6. REFERENCES

[1] C. Albrecht, B. Korte, J. Schietke and J. Vygen, "Maximum Mean Weight Cycle in a Digraph and Minimizing Cycle Time of a Logic Chip", *Discrete Applied Mathematics* 123(1-3) (2002), pp. 103-127.

[2] C. J. Alpert, Z. Li, G.-J. Nam, S. Ramji, C. N. Sze, P. G. Villarubia and N. Viswanathan, "Structured Placement of Latches/Flip-Flops to Minimize Clock Power in High-Performance Designs", *U.S. Patent* 8,954,912, May 2014.

[3] D. Arthur and S. Vassilvitskii, "K-Means++: The Advantages of Careful Seeding", *Proc. Eighteenth Annual ACM-SIAM Symposium on Discrete Algorithms*, 2007, pp. 1027-1035.

[4] L. Chen, A. Hung, H. M. Chen, E. Tsai, S. H. Chen, M. H. Ku and C. C. Chen, "Using Multi-bit Flip-Flop for Clock Power Saving by DesignCompiler", *Proc. Synopsys User Group*, 2010.

[5] C. Deng, Y.-C. Cai and Q. Zhou, "Register Clustering Methodology for Low Power Clock Tree Synthesis", *J. of Computer Science and Technology* 30(2) (2015), pp. 391-403.

[6] S. Dobre, Qualcomm CDMA Technologies, Inc., *personal communication*, April 2016.

[7] G.-J. Nam, IBM, *personal communication*, March 2016.

[8] C.-C. Hsu, Y.-T. Chang and M. P.-H. Lin, "Crosstalk-Aware Power Optimization with Multi-Bit Flip-Flops", *Proc. ASP-DAC*, 2012, pp. 431-436.

[9] W. Hou, D. Liu and P. H. Ho, "Automatic Register Banking for Low-Power Clock Trees", *Proc. ISQED*, 2009, pp. 647-652.

[10] I. H.-R. Jiang, C. L. Chang and Y. M. Yang, "INTEGRA: Fast Multibit Flip-Flop Clustering for Clock Power Saving", *IEEE TCAD* 31(2) (2012), pp. 192-204.

[11] S. Khuller and Y. J. Sussmann, "The Capacitated K-Center Problem", *SIAM J. Discrete Math.* 13(3) (2000), pp. 403-418.

[12] Y. Kretchmer, "Using Multi-Bit Register Inference to Save Area and Power: The Good, The Bad, and The Ugly", *EE Times Asia*, 2001.

[13] M. P.-H. Lin, C. C. Hsu and Y.-T. Chang, "Post-Placement Power Optimization with Multi-Bit Flip-Flops", *IEEE TCAD* 30(12) (2011), pp. 1870-1882.

[14] M. P. H. Lin, C. C. Hsu and Y. C. Chen, "Clock-Tree Aware Multibit Flip-Flop Generation During Placement for Power Optimization", *IEEE TCAD* 34(2) 2015, pp. 280-292.

[15] S. S. Y. Liu, W. T. Lo, C. J. Lee and H. M. Chen, "Agglomerative-Based Flip-Flop Merging and Relocation for Signal Wirelength and Clock Tree Optimization", *ACM TODAES* 18(3) (2013), pp. 40:1-40:20.

[16] S.-C. Lo, C.-C. Hsu and M. P.-H. Lin, "Power Optimization for Clock Network with Clock Gate Cloning and Flip-Flop Merging", *Proc. ISPD*, 2014, pp. 77-84.

[17] A. D. Mehta, Y.-P. Chen, N. Menezes, D. F. Wong and L. T. Pileggi, "Clustering and Load Balancing for Buffered Clock Tree Synthesis", *Proc. ICCD*, 1997, pp. 217-223.

[18] D. Papa, N. Viswanathan, C. Sze, Z. Li, G.-J. Nam, C. Alpert and I. L. Markov, "Physical Synthesis with Clock-Network Optimization for Large Systems on Chips", *IEEE Micro* 31(4) (2011), pp. 51-62.

[19] P. J. Rousseeuw, "Silhouettes: A Graphical Aid to the Interpretation and Validation of Cluster Analysis", *Journal of Computational and Applied Mathematics* 20 (1987), pp. 53-65.

[20] C. C. Tsai, Y. Shi, G. Luo and I. H.-R. Jiang, "FF-bond: Multi-Bit Flip-Flop Bonding at Placement", *Proc. ISPD*, 2013, pp. 147-153.

[21] S. H. Wang, Y. Y. Liang, T. Y. Kuo and W. K. Mak, "Power-Driven Flip-Flop Merging and Relocation", *IEEE TCAD* 31(2) (2012), pp. 180-191.

[22] C. Xu, P. Li, G. Luo, Y. Shi and I. H.-R. Jiang, "Analytical Clustering Score with Application to Post-Placement Multi-Bit Flip-Flop Merging", *Proc. ISPD*, 2015, pp. 93-100.

[23] J. T. Yan and Z. W. Chen, "Construction of Constrained Multi-Bit Flip-Flops for Clock Power Reduction", *Proc. ICGCS*, 2010, pp. 675-678.

[24] Cadence Innovus User Guide.

[25] IBM ILOG CPLEX. *www.ilog.com/products/cplex/*

[26] CAD/CAM/CAE Wallchart. *http://www.garysmitheda.com/wp-content/uploads/2015/05/All_WC-15.pdf*

[27] LEMON (Library for Efficient Modeling and Optimization in Networks). *http://lemon.cs.elte.hu/trac/lemon*

[28] OpenCores. *http://opencores.org*

[29] Synopsys Design Compiler User's Manual.