# Multisource Clock Tree Synthesis Through Sink Clustering and Fast Clock Latency Prediction

Byungho Choi, Yonghwi Kwon, Umar Afzaal, and Youngsoo Shin
School of Electrical Engineering, KAIST, Daejeon 34141, Korea

***Abstract*—Multisource clock tree consists of a number of local clock trees rooted at respective tap drivers, which are then connected to a clock source through H-tree. We address two key problems for the synthesis of multisource clock tree: clock sink clustering for constructing local clock trees, and the decision of the number of trees. Weight-balanced k-means clustering is applied for the first problem; sinks of the same cluster are localized and the load capacitances of tap drivers are balanced as much as possible. The number of trees can be searched in exhaustive fashion, while clock latency of local trees is estimated with fast CNN-based model. Experiments with a few test circuits demonstrate that clock latency is reduced by 11.8% on average, while synthesis runtime is reduced by 64% thanks to CNN model.**

## I. Introduction

Multisource clock tree synthesis (MSCTS) is used to maximize common clock path, so that on-chip variation (OCV) clock skew is minimized. For MSCTS, tap drivers with large driving strength are inserted, while each tap driver is connected to clock source through balanced tree, e.g. H-tree, and becomes a clock source of a local clock tree.

As shown in Fig. 1, a multisource clock tree consists of a global clock tree, in which tap drivers may be considered as sinks. Each tap driver becomes a source of a subtree, with clock sinks as leaves. The global clock tree may be constructed with either a mesh or a H-tree. A mesh is more effective in reducing the clock skew, but it suffers from large power consumption due to larger load capacitance and short-circuit current (2 to 3 times more power consumption than a tree [1]) and difficulty of timing analysis. We thus focus on a multisource clock tree with H-tree as global clock tree.

An MSCTS method with a goal of reducing both clock latency and clock skew has been proposed [2]. A layout is divided into uniform grids, and tap drivers are introduced at the center of each grid as shown in Fig. 2(a). Subtrees are then built from the tap drivers to the clock sinks within each grid through standard CTS. The load capacitances of tap drivers are not explicitly balanced and the distribution of sinks may vary quite a lot for different grids: subtree latency [3] will not be minimized as a result. In addition, the number of tap drivers is only given.

This work was partly supported by the National Research Foundation of Korea (NRF) of MSIT (No. 2019R1A2C2003402), Institute of Information & communications Technology Planning & Evaluation (IITP) grant funded by the Korea government (MSIT) (No. 2021-0-00754), and Samsung Electronics (No. IO201209-07906-01). The EDA tool was supported by the IC Design Education Center (IDEC), Korea.

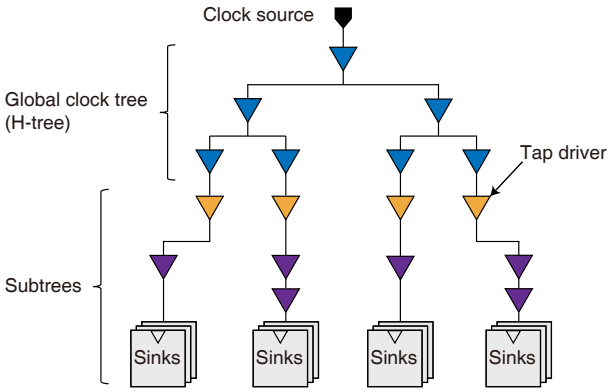


Fig. 1. Structure of H-tree-based multisource clock tree.

In this paper, we take more flexibility and assume irregular grids as shown in Fig. 2(b). We determine the best position of tap drivers within their corresponding grids; we also iterate the algorithm to find the best number of tap drivers. Specifically, clock sinks are clustered in such a way that tap drivers will have similar load capacitances and the variance of distances from each tap driver to associated sinks is minimized. To satisfy the two objectives together, we employ a weight-balanced k-means clustering algorithm.

Iteration of MSCTS with different number of tap drivers is computation intensive. In particular, the runtime to extract the maximum subtree latency is a key. We introduce a CNN model for quick prediction of subtree latency. CNN receives sink distribution map, sink capacitance map, and white space map. In addition, it also receives grid size (see Section II-B) and the variance of sink to tap driver distances.

The remainder of this paper is organized as follows. The details of the proposed method are provided in Section II with two main components: tap driver placement through clustering and subtree latency prediction using CNN model. In Section III, the effectiveness of the proposed method is assessed in terms of runtime and maximum latency. Conclusions are drawn in Section IV.

## II. Proposed MSCTS

The overall flow of the proposed method is shown in Fig. 3. Given a circuit layout with sink placement and a number of tap drivers $n$, we first find the optimal positions of tap drivers using sink clustering. Clustering is performed to balance the sum of sink capacitance of each tap driver, and minimize

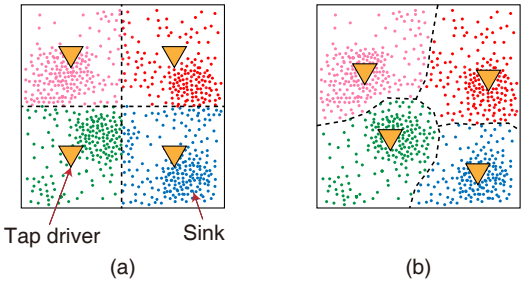


Fig. 2. (a) Regular tap driver placement and (b) proposed tap driver placement by clustering.

the sum of square of the sink-to-tap driver distances in order to minimize the maximum latency. Clustered sinks yield tap drivers with similar load capacitance and minimized sink-to-tap distances. The centroid of each cluster is determined as the location of the tap driver and sinks included in each cluster are assigned to be driven by corresponding tap driver. Then, assigned sinks construct the subtree of the tap driver.

With the positions of tap drivers from the clustering, we extract H-tree latency, which is the latency from the clock source to each tap driver; this is done through standard H-tree CTS. We then predict the maximum subtree latency for each tap driver using the CNN-based subtree latency prediction model. Maximum latency can be obtained as a sum of maximum subtree latency from the prediction model, and H-tree latency extracted by using a commercial tool.

The steps are iterated while we vary the number of tap drivers from 1 to $N$. We choose $n$ with the lowest maximum latency as the best solution. Finally, subtree CTS is performed using the optimal number of tap drivers to obtain the optimal clock tree.

### A. Clock Sink Clustering

We find the tap driver positions using clock sink clustering to minimize the maximum latency. Given a sink placement and a number of tap drivers, weight-balanced k-means clustering [4] is performed to group the sinks. After clustering, the location of tap drivers is determined as the centroid of the clusters, and the sinks in each cluster are assigned to the tap driver. Weight-balanced k-means clustering is capable of handling weighted point sets and prescribed lower and upper bounds on the cluster sizes.

The objective function of the clustering is given by the sum of square of the sink-to-tap driver distances:

$$\text{Objective} = \sum_{i=1}^{n}\sum_{j=1}^{k} y_{ij} w_j \cdot \|x_j - s_i\|^2, \tag{1}$$

where $n$ is the number of clusters (tap drivers), and $k$ is the number of sinks. Note that $y_{ij}$ is an assignment variable of point $x_j$ that belongs to $i$-th cluster $C_i$. $w_j$ denotes the weight of sink $j$, which is the pin capacitance of sink $j$. The constraint

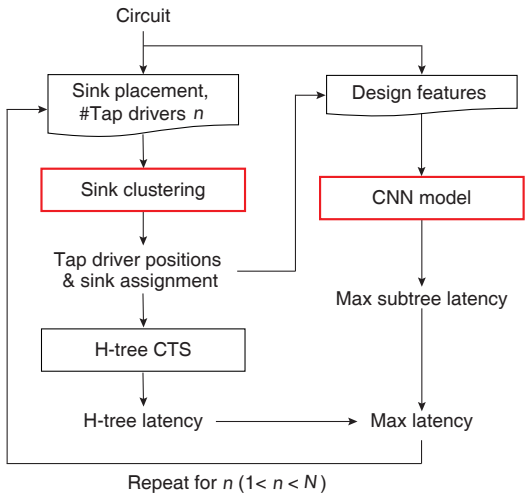


Fig. 3. Overall flow of proposed method.

is the lower bound and upper bound on the cluster size, which is the sum of sink pin capacitance in each cluster:

$$\kappa^- \le |C_i| \le \kappa^+, |C_i| = \sum_{j=1}^{k} y_{ij} w_j. \tag{2}$$

Clustering algorithm works as follows. Sink placement and the number of tap drivers $n$ are given. First, we randomly choose $n$ sink coordinates from the given sink placement and set each position of the sink as an initial centroid of each cluster. The centroids become the positions of the tap drivers. Second, we assign sinks to each cluster by solving a linear program under the objective function (1) and constraint (2). In the third step, the position of each centroid is determined as a mean coordinate of sinks that are placed in each cluster. The second and third steps are repeated until the value of the objective function converges.

Finally, clustering algorithm returns the coordinates of tap drivers and coordinates of sinks which are assigned to each tap driver. After the clustering, the maximum load capacitance of the tap driver can be reduced, and it leads to a reduction of maximum subtree latency. Also, by decreasing the variance of the sink-to-tap driver distances, subtree wirelength and subtree latency are reduced.

### B. Subtree Latency Prediction Model

Subtree latency prediction model is built using CNN. To capture information from 2D feature maps and other numerical features of the design, we adopt CNN to predict the maximum subtree latency from each tap driver.

CNN structure of proposed method is shown in Fig. 4. Two types of inputs are provided to the CNN model. For the first type, three 2D maps are fed into the first convolutional layer and go through 5 convolutional layers and 4 fully connected layers. There are 3×3 kernels for each convolutional layer. Numerical features are also used as CNN inputs. They are fed into the first fully connected layer which is concatenated with

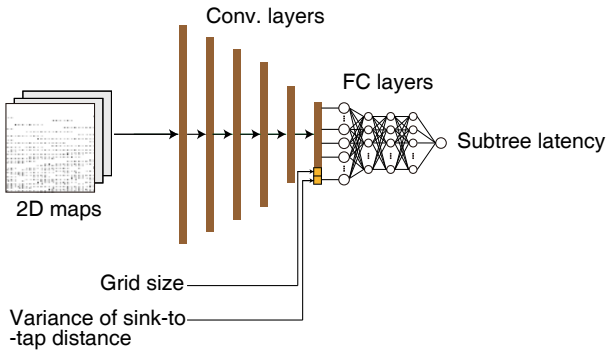


Fig. 4. CNN structure used for subtree latency prediction.

the flatten outputs from the last convolutional layer. Output of the model is a predicted maximum subtree latency from a tap driver to sink. For training of the CNN model, a cost function of mean squared error (MSE) is minimized. Prediction is performed for every tap driver of the design.

We extract three 2D feature maps centered on a tap driver and two numerical features from the placement result, and the tap driver placement and assignment which are obtained by the clustering. Each map is composed of a pixel array with the same number of pixels.

*1) Sink Distribution Map:* Since the sinks are the endpoints of the clock signal, sink distribution directly affects the maximum latency from clock source to sinks. Sink distribution map consists of two channels centered on a tap driver as shown in Fig. 5. For the first channel, each pixel contains its number of sinks which are assigned to the tap driver at the center of the map. For the second channel, each pixel contains the number of sinks which are assigned to other tap drivers. The number of sinks assigned to other tap drivers is used to consider that other surrounding subtrees may interrupt the buffer insertion and routing of subtree of the targeted tap driver. It can reflect an increase in subtree latency.

*2) Sink Capacitance Map:* Subtree latency is a strong function of tap driver's load capacitance, which in turn is affected by sink capacitance [3]. The sum of input capacitance of sinks within each pixel becomes pixel value for this map. For each tap driver, a part of sink capacitance map corresponding to its sinks is provided to CNN.

*3) White Space Map:* Construction of the subtree is influenced by the white (empty) space for clock buffer insertion. If the white space for buffer insertion is not enough due to blockages and standard cells, the optimal subtree may not be constructed, which may lead to larger latency. To consider the amount of white space in each region, we generate a white space map by calculating the ratio of white space area to a single pixel area for each pixel. So, each pixel of the white space map has a value between 0 and 1.

Grid size, which is the unit length of a grid for the 2D maps, is submitted to CNN as a numerical feature. It represents the size of the bounding area where the sinks are placed. The variance of the sink-to-tap driver distances, which is proportional to the subtree wirelength of the tap driver is also used for numerical feature input of CNN.

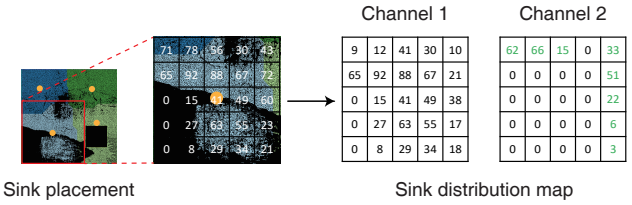


Fig. 5. Extraction of sink distribution map from sink placement.

## III. Experimental Results

Four test circuits are taken from OpenCores [5] for experiments. An initial netlist and its placement of each circuit are obtained through a commercial logic synthesis and physical design tool, respectively, with 28nm standard cell library. Each circuit is placed with 18 different options (6 different core utilization factors from 0.50 to 0.75 with 0.05 step size, and 3 different blockage configurations), which yield a total of 72 layouts for four circuits.

### A. Maximum Latency Prediction

CNN-based subtree latency prediction model is trained with 2250 samples from 54 layouts of 3 circuits; it is tested with 750 samples from 18 layouts of a single remaining circuit. One sample is extracted for each tap driver. The size of input feature maps is set to 128×128 pixels, where the size of a pixel (i.e. grid size) is varied by the bounding area of sinks assigned to a tap driver. CNN model is implemented in Python using PyTorch [6] while Adam optimizer [7] is used to train the model with learning rate of 0.0001.

Reference maximum latency is extracted through CTS by commercial tool, with tap driver positions and tap assignment after the clustering. We obtain the predicted maximum latency as a sum of the maximum subtree latency predicted by CNN model and H-tree latency extracted from commercial tool. Fig. 6 compares the reference maximum latency and the predicted maximum latency of the tate_5 circuit. It shows that predicted latency can correctly find the optimal number of tap drivers, which shows the lowest maximum latency.

We also evaluate the Spearman correlation, a measure of correlation between the ranks of two variables, to analyze how well the rank of predicted latency matches that of reference latency as the number of tap drivers is changed. The average correlation is 0.93 for test datasets, which shows that the predicted latency has a correlated tendency with the reference latency, as shown in Fig. 6. To assess the accuracy of subtree latency prediction model, mean absolute percentage error (MAPE) is evaluated. The proposed method achieves a small error, 4.6%, on average for test datasets.

### B. Search for Optimal Number of Tap Drivers

The goal of MSCTS is to find the optimal number of tap drivers (with their respective local clock trees) such that

TABLE I
RUNTIME (MIN) OF PROPOSED METHOD COMPARED TO PREVIOUS WORK

| Circuits | [2] | Proposed | | | | |
|---|---|---|---|---|---|---|
| | | Clustering | H-tree latency extraction | Prediction | Optimal CTS | Total (speed-up) |
| jpeg_5 | 431 | 124 | 63 | 0.1 | 54 | 241.1 (1.8x) |
| tate_5 | 362 | 105 | 61 | 0.1 | 46 | 212.1 (1.7x) |
| gfx_5 | 557 | 132 | 58 | 0.1 | 71 | 261.1 (2.1x) |
| gfx_20 | 2066 | 370 | 71 | 0.1 | 304 | 745.1 (2.8x) |
| Average | 854 | 183 | 63 | 0.1 | 119 | 365.1 (2.3x) |

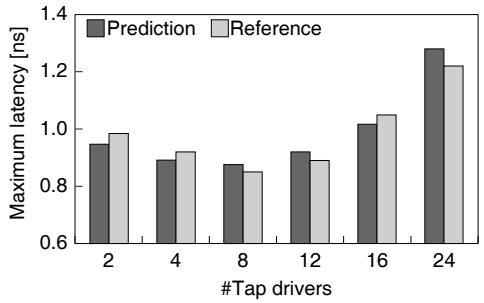


Fig. 6. Maximum latency comparison between prediction and reference for tate_5.

maximum clock latency is minimized. The proposed MSCTS is compared to conventional one. Two methods are repeated for the different numbers of tap drivers to identify the maximum latency at each number of tap drivers. Candidates for the number of tap drivers are set as 2 to 24 as the latency rapidly increased when the number become larger than 24.

The runtime of the proposed method is compared with conventional MSCTS in Table I. The proposed method achieves up to 2.8 times speed-up compared to conventional MSCTS. Runtime of the proposed method consists of four components. Clustering takes the largest portion of the total runtime, 50%, on average. The runtime of H-tree latency extraction and subtree latency prediction corresponds to 17% and 0.03% of the total runtime, respectively. Final CTS, which builds a clock tree for the optimal number of tap drivers, takes 33% of the total runtime. The speed-up of our proposed method will increase as the number of candidates for number of tap drivers increases.

In Table II, for each circuit, the optimal number of tap drivers of proposed method and conventional MSCTS are listed. For tate_5 and gfx_5, the optimal numbers of tap drivers are the same as that of conventional MSCTS. On the other hand, for jpeg_5 and gfx_20, the optimal numbers of tap drivers are observed at different points from conventional MSCTS. This is because clustering results in an irregular placement of the tap drivers, unlike conventional MSCTS. In addition, as a result of clustering, the maximum latency at the optimal number of tap drivers is reduced by an average of 11.8% compared to conventional MSCTS.

TABLE II
OPTIMAL NUMBER OF TAP DRIVERS AND LATENCY (NS)

| Circuits | [2] | | Proposed | |
|---|---|---|---|---|
| | #Tap drivers | Latency | #Tap drivers | Latency |
| jpeg_5 | 2 | 1.287 | 4 | 1.103 |
| tate_5 | 8 | 0.983 | 8 | 0.875 |
| gfx_5 | 8 | 1.261 | 8 | 1.085 |
| gfx_20 | 8 | 2.814 | 12 | 2.597 |
| Average | | 1.000 | | 0.882 |

## IV. CONCLUSION

A method to find the optimal number of tap drivers and their locations has been presented. For a fixed number of tap drivers, the optimal tap driver placement and assignment have been iteratively determined through weight-balanced k-means clustering of sinks. Maximum subtree latency has been predicted by a CNN-based subtree latency prediction model. Total maximum latency has been obtained as a sum of predicted maximum subtree latency and H-tree latency extracted by commercial tool. The clustering and latency prediction have been iterated over to find the optimal number of tap drivers that shows the lowest maximum latency. Experiments have demonstrated up to 2.8 times speed-up and 11.8% reduced maximum latency compared to conventional method.

## REFERENCES

[1] A. L. Sobczyk, A. W. Luczyk, and W. A. Pleskacz, "Power dissipation in basic global clock distribution networks," in *Proc. Design and Diagnostics of Electronic Circuits and Systems*, Apr. 2007, pp. 1–4.
[2] V. Srivatsa, A. P. Chavan, and D. Mourya, "Design of low power & high performance multi source h-tree clock distribution network," in *Proc. VLSI Device Circuit and System*, Jul. 2020, pp. 468–473.
[3] F. Minami and M. Takano, "Clock tree synthesis based on rc delay balancing," in *Proc. Custom Integrated Circuits Conf.*, May 1992, pp. 28–3.
[4] S. Borgwardt, A. Brieden, and P. Gritzmann, "A balanced k-means algorithm for weighted point sets," *arXiv preprint arXiv:1308.4004*, 2013.
[5] Opencores benchmarks. [Online]. Available: http://www.opencores.org
[6] A. Paszke *et al.*, "PyTorch: An imperative style, high-performance deep learning library," in *Proc. Int. Conf. on Neural Information Processing Systems*, Dec. 2019, pp. 8026–8037.
[7] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv:1412.6980 [cs.LG]*, Dec. 2014.