Symmetrical Buffer Placement in Clock Trees for Minimal Skew Immune to Global On-chip Variations

Renshen Wang
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0404
Email: rewang@cs.ucsd.edu

Takumi Okamoto
System IP Core Research Lab
NEC Corporation
Kawasaki, Kanagawa 211-8666
Japan
Email: okamoto@ct.jp.nec.com

Chung-Kuan Cheng
Department of Computer Science and Engineering
University of California, San Diego
La Jolla, CA 92093-0404
Email: ckcheng@ucsd.edu

***Abstract*—As the feature size of VLSI circuits scales down and clock rates increases, circuit performance is becoming more sensitive to process variations. This paper proposes an algorithm of symmetrical buffer placement in symmetrical clock trees to achieve zero-skew in theory, as well as robust low skew under process or environment variations. With the completely symmetrical structure, we can eliminate many factors of clock skew such as model inaccuracy, environment temperature and intra-die process variations. We devise a new dynamic programming scheme to handle buffer placement and wire sizing under the constraint of symmetry. By classifying the wires by tree levels and defining the level-dependent blockages, the potential candidate points in the gaps of circuit blocks can be fully explored. The algorithm is efficient for minimizing source-sink delay as well as other linear cost functions. Experiments show that our method helps to obtain a balanced design of clock tree with low delay, skew and power.**

I. Introduction

Clock distribution is a critical part of today's synchronous integrated circuits. As clock frequencies scale high, lower clock skew is desired to guarantee correct functioning of high performance ICs. On the other hand, as feature sizes continue to scale down, process variations inevitably become a larger factor to clock skews. Other factors such as environment temperature many also affect circuit components and introduce extra skews. Clock skew is typically defined as the maximum difference between the arrival time (d_i) of logically connected pairs of sinks in the clock distribution network, as in [7]

$$\text{Skew} = \max_{i,j \in Sinks} (d_i - d_j) = \max_{i \in Sinks} d_i - \min_{j \in Sinks} d_j$$

Skew is generally harmful because it may limit the clock speed or cause functional errors between synchronous elements. Clock tree synthesis algorithms which minimize skews are proposed in [8], [9], and zero-skew under Elmore delay model is achieved in [1], [3]. Thermal variation is considered in [4] where the worst case skew under different temperature distributions is minimized. These works mainly try to minimize the skew in the routing process, by selecting proper merging points. The result of these approaches may be affected by the accuracy of timing model, and also affected by process and environment variations since the R, C parameters are sensitive to many factors such as wire width, wire thickness and temperature.

Adding buffer in the clock tree is another technique for reducing skews. Buffer sizing and/or wire sizing techniques are proposed in [12] and [7], where sequential programming techniques are adopted to minimize both power consumption and clock skew. These non-symmetrical approaches also have the drawback of being affected by modeling errors and on-chip variations mentioned above.

Modern high-performance ICs are using more complex clock distribution networks. For example in [10], a single global clock signal is first distributed over the whole chip, and then local clock blocks are used to buffer and amplify the global clock. The primary global clock distribution network is designed to have minimum skew in order to simplify the design, reduce timing complexity and reduce clock uncertainty. A typical global clock network structure is a symmetric H-tree with buffers, because symmetrical tree structures are naturally zero-skew. In gated clock trees, a type-matching technique [2] by symmetry is proposed also for the purpose of zero skew.

In our approach, we devise an algorithm for symmetrical buffer placement algorithm in a global clock tree. By making the global clock distribution network completely symmetrical, we make it not only zero-skew in theory, but also be robust against many skew inducing factors. Minimal delay under the constraint of symmetry minimizes the effect of local on-chip variations, because the skew margin for on-chip variation (OCV) is computed as *delay*×*derating factor*. The constraint on buffer placement is often very strong, because to avoid placing buffer a circuit block we may need to give up other areas on the chip to maintain the symmetrical structure. Using the proposed level-dependent blockages, we adopt a revised dynamic programming approach based on [6] which can efficiently obtain optimal solutions under the constraint of symmetry. We are also able to minimize resource or power consumption induced by total capacitive load. As long as the objective function is linear such as (*delay* + *#buffers*×x), where x is a coefficient set by user, optimal solution can be computed in very short time. The results can be used with little extra effort in practical clock network designs.

The following section will introduce the problem statement.

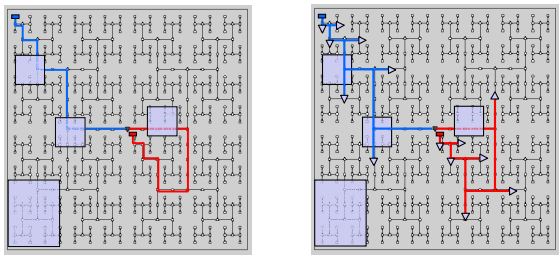
978-1-4244-5028-2/09/$25.00 ©2009 IEEE

Section 3 discusses the characteristics of the problem with the symmetry constraint. Section 4 describes the algorithm in details and section 5 shows the experimental results of the algorithm on industrial cases. Finally section 6 concludes the paper.

II. Problem Statement

We are given a symmetrical binary clock distribution tree with a source at root and a set of sinks all at leaves. The tree is symmetrical which means all the paths from root to leaf have an identical sequence of wire segments and branching points. We define the wire segment at the root to be on level 0 and the children of branches on level i are on level $i+1$. The wire segments on the same level are identical, so are the counts of there children. Also given is a set of circuit blocks which are blockages of buffer placement. The task is to symmetrically insert buffers in the wires avoiding the blockages, so that while keeping all the source-to-sink paths identical, the delay from the source to the set of sinks is minimized.

Fig.1(a) shows a simple example of a symmetrical clock tree with two sinks. Although the sinks have different distances from the source, the H-tree structure ensures that all the leaves have identical paths from the source by every point. Note that our problem includes clock trees with general symmetrical structures not limited in H-trees. Tree leaves do not necessarily have a sink on it, and the overall distribution of the clock signal sinks can be uneven.



(a) Identical paths (b) Same load by every point

Fig. 1. An H-tree clock network with two sinks

Elmore delay is used as our timing model. For tree structures, the delay from root to leaf is the sum of resistance times downstream capacitance over all segments.

$$D_{Elmore}(i) = \sum_{k \in \pi(i)} R_k L_k \qquad (1)$$

where $\pi(i)$ is the set of nodes on the path from the root to node i, R_i is the resistance of branch i and L_i is the downstream capacitive load of node i.

When buffers are inserted in the wires, the delay on a path is the sum of the delay on each wire segment and the internal delay on each buffer. As in Fig.2, a buffer has three parameters, input capacitance C_{buf}, internal delay D_{buf} and output resistance R_{buf}. The delay on a wire can be calculated using the buffer parameters at both ends in (1).

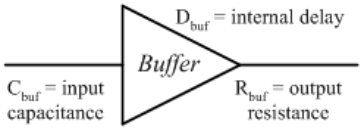


Fig. 2. Buffer model

By the Elmore delay model and our problem formulation, we need not only identical segments along the paths, but also exactly same capacitive load on every point along the paths.

In Fig.1(b), the two paths are still symmetrical because the additional wire branches attached to them have same lengths and loads. In fact, the delays will always be identical in this case no matter what the timing model we use as long as mutual inductance between segments can be ignored (which is the case for global clock distribution). With the definition of symmetrical paths, the symmetrical buffer placement problem can be formulated.

Definition 1: The constraint of symmetry on buffers:

1) *A path from the source to a sink is an effective path;*
2) *A path from a node to a non-sink leaf of the tree is a don't-care path;*
3) *All effective paths must have identical buffer placement and identical wire sizing;*
4) *Buffers at the same path length points from the source must have identical downstream load.*

Under the constraint of symmetry, we place buffers to minimize the delay from source to sink. We use Elmore delay in the delay minimization. Even with presence of modeling error, the signal skew between the sinks are always zero under ideal conditions. With local process variations, the effect of Elmore delay model error, ($min\_delay_{Elmore} - min\_delay_{real}$)$\times$*OCV derating factor*, is negligible.

III. Properties on Symmetric Trees

Buffer placement has been well studied since [6]. In our problem formulation, there is an additional strong constraint of symmetry on the positions of buffers. We need to discuss the effect of this constraint on the solutions.

A. Path combining

First, we observe that in a tree with perfect symmetry, the paths can be combined into a single path with increasing width by levels of the tree. In Fig.3, the H-tree is topologically equivalent to a binary tree. And since all the paths from source to sink are identical, the tree branches can be combined into a single path, where at tree level i the path has 2^i times wire width. The delay on this path is same as the delay on any path in the original H-tree.

When we insert buffers in the tree under the constraint of symmetry, buffers at a certain path length can also be combined to one big buffer. At tree level i, the combined buffer has 2^i times capacitance and $1/2^i$ resistance. Thus, the tree structure in our problem is simplified into a linear structure.

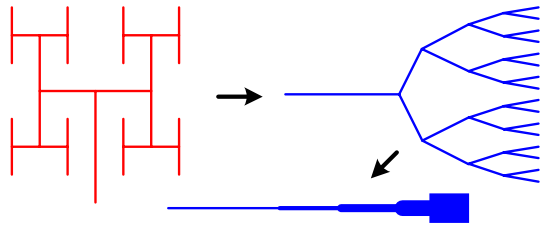


Fig. 3. Combining identical paths

B. Level-dependent blockages

Circuit blocks on the chip occupy silicon resources and act as blockages of buffer candidate points. With the constraint of symmetry, a blockage on one of the branches may also block other branches in that level, depending on what path the blockage is on and the level of upstream buffers.

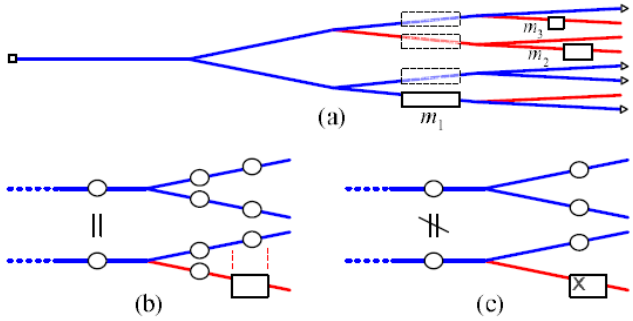


Fig. 4. Blockages on tree branches

The detailed cases of blockages are shown in Fig.4 and explained as follows.

(i) When part of an effective path is blocked (m_1 in Fig.4(a)), then the same part on other paths are also blocked.

(ii) When part of a don't-care path is blocked(m_2 and m_3 in Fig.4(a)), we assume the don't-care path starts at level l_p and the blockage is on level l_b,

(ii.a) If its nearest upstream buffer is on the same level $l = l_b$ as the blockage (as in Fig.4(b)), or the upstream buffer is on a lower level $l < l_b$ but $l \geq l_p$, we can ignore the blockage. Because the don't-care path needs no buffer while other paths can be placed with buffers unaffected;

(ii.b) If the upstream buffer is on a lower level $l < l_b$ and $l < l_p$, the blockage is effective. Because if we place buffers on other branches like in Fig.4(c), while we cannot place a buffer on the same position of the don't-care path, the result is that the upstream buffers have different load capacitances which violates the constraint of symmetry.

From the two cases of blockage on a don't-care path, we observe that blockages on the clock tree under the constraint of symmetry is level-dependent. We must consider the position of upstream buffers when we mark the blockages for buffer placement algorithms: In the combined path (Fig.3) consisting of buffer candidate points, we mark the blocked points with an extra dimension of "level on tree".

Definition 2: Level-dependent blockage:

1) *The buffer candidate points on each path are numbered as 1, 2, 3, ..., MAX starting at the root and ending at the leaves;*
2) *Blocked*$[p][l]$ *indicates the candidate point* p *is blocked if the upstream buffer is at level* l.

By the definition above, blockage array *Blocked*$[p][l]$ can be pre-computed by checking each tree branch covered by circuit blocks. If a blocked point p on level l is on

(i) an effective path, *Blocked*$[p][l] \leftarrow 1$;

(ii) a don't-care path, find the starting level l_p of the don't-care path. *Blocked*$[p][0] \leftarrow 1$, ..., *Blocked*$[p][l_p - 1] \leftarrow 1$.

The two operations map the circuit blocks of the chip into the combined path as level-dependent blockages, where the buffer placement optimization can be carried out. In the example of Fig.4(a), the buffer candidate points covered by m_2 have $Blocked[p][i] = 1$ for $i = 0, 1$; and the points covered by m_3 have $Blocked[p][i] = 1$ for $i = 0, 1, 2$.

C. Level merging in trees

To exploit the full potential of buffer placement solutions, we also need to merge some continuous levels in definition 2 in order to remove unnecessary false blockages. As the example in Fig.5, the branches of level 1 both have only one child branch in level 2. *Blocked*$[p][1]$ does not need to be 1 even there is a block on point p, because we can delete the don't-care paths starting from level 2 while preserving the symmetry of the tree.

Definition 3: Mergeable tree levels: If there is no pair of effective paths sharing a branch at level k*, then level* k *and level* $k + 1$ *are mergeable.*

The mergeable levels are easily recognizable by definition 3. We merge all the consecutive mergeable levels to further remove unnecessary constraints. Although to merge any levels we need at least half of non-sink leaves, this is still likely to appear in the cases with largely concentrated clock loads.

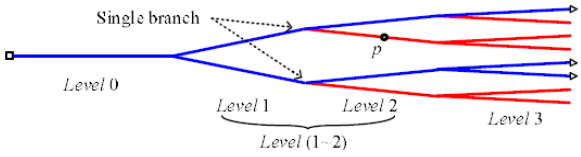


Fig. 5. Merging level 1 & level 2

D. Feasibility of solutions

By our basic formulation of the problem, the given clock tree is symmetrical. Even if the entire path is covered by blockages, we have least one feasible solution which is not placing buffers at all.

However, in practice the driver and buffers of the clock distribution network have limited driving capacities, which means a certain number of buffers are necessary. In some cases with large portion of area occupied by circuit blocks, it is possible that no feasible solution exists. Resolving this issue will involve circuit floorplan and relaxation on the

constraint of symmetry. In the following sections we focus on the algorithms for the ideal formulation which guarantees symmetrical buffer placement solutions.

IV. Algorithms for Symmetrical Buffer Placement

A. Previous dynamic programming approaches

To place buffers in an RC network for minimal delay, a dynamic programming [5] algorithm in O(n^2) time is proposed in [6], where the state after a partial solution can be described as a pair (D, L). D is the delay at the point and L is the load capacitance. For each value of D we only need the minimal L and vice versa. Fig.6 is a possible curve of states at a buffer candidate point.

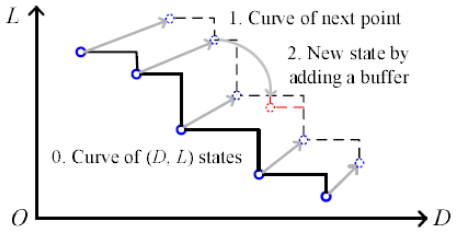


Fig. 6. Curve of states in dynamic programming

At each point, we keep a curve of the (D, L) states, which contains the minimal delay at each load capacitance. The curves can be derived point by point, starting from the sinks and ending at the source point. A step of deriving is shown in Fig.6. With an added segment of wire, the load is increased by the capacitance of the segment, and the delay is increased by the Elmore delay. Assume the wire has resistance rate r and capacitance rate c,

$$D'_k = D_k + r\Delta l L_k + \tfrac{1}{2}(c\Delta l)(r\Delta l)$$
$$L'_k = L_k + c\Delta l$$

Besides adding wire segments, we may also add buffers which isolate load capacitance. Using a buffer with output resistance R_{buf}, input capacitance C_{buf} and intrinsic delay D_{buf}, we have

$$D''_k = D'_k + D_{buf} + R_{buf} L'_k$$
$$L''_k = C_{buf}$$

If the new state (D''_k, L''_k) is under the curve, we add the state into the set and the overall curve is improved by the buffer.

The flow above returns optimal solution for usual buffered interconnections.

However, in our symmetrical buffer placement, the (D, L) states are no longer sufficient to propagate all the optimal partial solutions, because there is no information on the position of the nearest buffer(s). Specifically, we cannot decide whether we are allowed to add buffer on a point with the curve of (D, L) pairs. We adopt an alternative form of dynamic programming using "delay at buffer" to solve this problem.

B. Revised dynamic programming with level-dependent blockages

To handle the constraint of symmetry and level-dependent blockages, we reduce the states from a curve of delay-load pairs into a single value D, which denotes the minimal delay when buffers are placed at the point. Following is a simple form of the revised dynamic programming without considering different buffer types or wire sizing.

$$D[k] = \min\{D[i]+delay(k,i) : i > k \;\&\; \neg Blocked[i][level(k)]\} \quad (2)$$

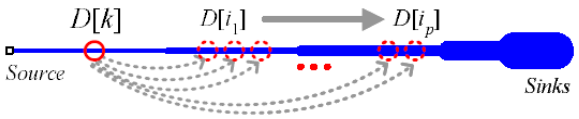


Fig. 7. Dynamic programming by path sweeping

Instead of sweeping a curve of states at each buffer candidate point, we sweep a series of points with partial solutions of "minimal delay on buffer". Starting from the end points, we compute $D[k]$ for each point by the recursive formula. As in definition 2, "*Blocked*$[i][level(k)]$ = 0" means when the upstream buffers of point i are placed at point k, the symmetry on paths can be preserved. In this case, the configuration of "buffer at k" followed by "buffer at i" is feasible and $D[i] + delay(k, i)$ can be considered as a candidate of $D[k]$; otherwise, point i must be ignored.

Regarding the number of points on a path as n, the time complexity of this algorithm is O(n^2), same as the original dynamic programming in [6]. In recursive equation (2), $delay(k, i)$ can be calculated using Elmore delay model by equation (1). The delay on the wire from point k to i is the sum of a series of RC terms and can be added in O(n) time. When we sweep the wire from $i = k+1$ to *MAX*, the total time is still O(n) since the values of $delay(k, i)$ are accumulative. Therefore the overall time is O(n^2).

The size of space our algorithm uses is O(n), better than the O(n^2) in [6]. Because instead of storing a curve of O(n) states at each point for tracing optimal solution, we need only one state each point.

The essential amount of computation needed by the two forms of dynamic programming is same. Both need to sweep a series of states at each step. The original form in [6] sweeps the curve of states while the revised form sweeps a series of points. But in our zero-skew problem with with constraint of symmetry and level dependent blockages, the latter one has significant advantages.

C. With buffer and wire sizing

In clock tree planning, we have additional choices of different buffer types and wire sizing. We may also have inverters with a constraint that the total number of inverters along an effective path must be even. Wire sizing is limited to the whole net in our case, i.e. the wires in a same net always have the same width. To handle these options, we add some modifications on (2).

For buffer types, since they have different capacitive load which will decide the delay after the partial solution, we add a dimension of *"buffer type"* in the dynamic programming

space. $D[k][T]$ denotes the minimal delay at point k when buffer type T is placed at k.

When inverters are considered, we allow inverted signals in some segments of the path as long as the final signal at sinks is uninverted. Since buffers are actually two inverters packed together, inverters usually have less internal delay than buffers and are preferred in many types of designs. We double the solution space by adding another dimension $[0, 1]$. $D[k][T][v]$ denotes the minimal delay point k when buffer/inverter type T is placed and the signal is inverted for $v = 1$, uninverted for $v = 0$. The size of solution space is increased by $2T$ times for buffers/inverters. T is usually a small number, so the overall space complexity is unchanged.

Wire sizing on each net can be optimized by directly adding a loop of choosing wire width $w_1, w_2, w_3, ..., w_U$. The overall flow is as follows.

Zero-skew clock tree buffer placement algorithm

INPUT: A symmetric tree, buffer/inverter and wire types
OUTPUT: Minimum clock delay with symmetrical buffer placement
1. Merge all the consecutive mergeable tree levels
2. Precompute array *Blocked*$[p][l]$
3. FOR $k =$ *MAX*, *MAX*$-1, \cdots, 1$
  FOR $i = k + 1, \cdots, MAX$
    IF not *Blocked*$[i]$[*level*(k)]
      FOR $t_1 = 1 \cdots T$ and $t_2 = 1 \cdots T$
        FOR $v_1 = 0, 1$
          if (buffer type t_1 is an inverter), $v_2 = 1 - v_1$;
            else $v_2 = v_1$;
          FOR *size* $= 1 \cdots U$ /* *wire sizing* */
            Compute $\Delta d =$ *delay*$(k, t_1, i, t_2, w_{size})$;
            if $(D[k][t_1][v_1] > D[i][t_2][v_2] + \Delta d)$
              Update $D[k][t_1][v_1]$;
        end_FOR *size*, v_1, t_2, t_1;
      end_IF
  end_FOR i, k;
4. Find $\min_{t \in \{1, \cdots, T\}} D[1][t][0]$;
5. Trace the solution from point 1 with buffer type t;

In summary, just by changing the dynamic programming process from "curve sweeping" to "path sweeping", our algorithm can efficiently handle the special level-dependent blockages induced by the constraint of symmetry. The "curve of states" method in [6] is most efficient when we have strong wire sizing options where the wire width can change from point to point along a wire, which is impractical in manufacturing. Usually the wires in a net have the same size, where our revised form of dynamic programming has no drawbacks.

The optimization flow is also applicable to other cost functions besides minimal delay. In clock distribution network designs, a linear cost function is common, such as for minimizing delay, number of buffers or capacitive loads.

V. Experimental Results

We implement the algorithm in C++ and tested on several industrial cases from NEC corporation. The clock tree area used in the cases has a size approximately 2.3mm×2.6mm. We pick the distance between buffer candidate points to be 10μm, so by a 7-level H-tree structure, the number of points on the combined path is about 250. The running time is less than 1 second, and we can expect the time to be in seconds for larger chips at larger scale of e.g. 10mm×10mm.

A. Minimizing delay

The basic objective is to minimize the delay from source to sinks. We tested the minimum delay on three basic cases, T1, T2 and T3. In these cases, some leaves of the H-tree are covered by circuit modules and are not effective sinks. We consider these leaves still having load capacitance of a normal sink, since we can add wires to match the capacitance.

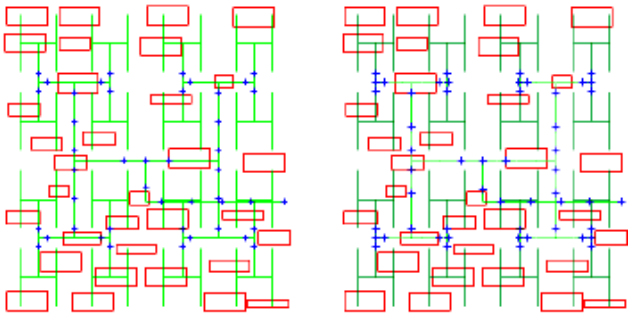


Fig. 8. Case T2 and T2$_w$

Table 1 lists the summary of our experiments, including number of tree levels, ideal minimum delay when there are no blockages, minimum delay with blockages, number of buffers on each path and total number of buffers used. There are 7 types of buffers available and different types are used over the clock tree. The wire width is 90nm in the first 3 cases (Ti) and can be chosen among (45nm, 90nm, 135nm) in the last 3 cases (Ti_w). We find the blockages have a large impact on the minimum delay we can achieve. The effect of wire sizing varies from negligible to significant among the test cases. We show the results of T2 and T2$_w$ in Fig.8 where the solutions have greater difference. The differences on wire width between nets are also shown in the T2$_w$ case.

TABLE I
Minimum delay buffer placement summary

Case	Levels	Ideal delay (*ns*)	Delay (*ns*)	Buffers /path	Buffers total
T1	7	1.121	1.894	13	33
T2	7	1.159	1.867	13	45
T3	9	1.198	1.573	16	97
T1$_w$	7	1.117	1.890	13	33
T2$_w$	7	1.153	1.807	14	53
T3$_w$	9	1.191	1.543	16	97

We also integrate the program in a design system and the solutions of two test cases are shown in Fig.9. The left one is an H-tree, and the right one has a more complex but still symmetrical "~~H~~H" basic structure.

B. Tradeoff between delay and resource

Since the same flow is applicable to other objectives, we also test our program using a cost function with both delay

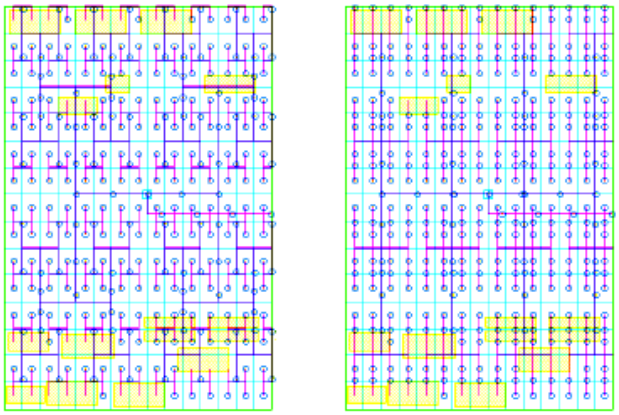


Fig. 9. Buffer placement in NEC CAD system

and number of buffers. T3$_w$ is a test case with combined cost function (delay + #buffers $\times x$), where the delay is measured in pico-second and x is a coefficient which can be tuned. We can increase x for reducing buffers, or decrease x for reducing delay, depending on the overall chip design. The program can generate optimal solutions in same amount of time as described in the previous subsection.

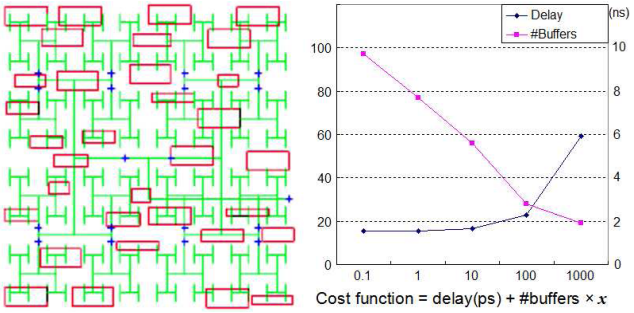


Fig. 10. Tradeoff between delay and #buffers

As the coefficient x in the cost function varies from 0.1 to 1000, the results of "delay" and "#buffers" form two curves, one increasing and the other decreasing. Fig.10 shows the result with minimum number of buffers and the two curves with $x = 0.1, 1, 10, 100, 1000$. These curves reveal the tradeoff between two conflicting objectives and possible design compromises. Starting from the minimum delay, when we slightly loosen the delay constraint, the number of buffers decreases fast. This tendency continues until x is between 10 and 100, where the delay penalty becomes larger for reduced buffers. In practical design, we may choose a solution in this area for a good compromise of delay and resource.

Other objective functions can also be applied in the same way. For example, the total capacitive load, including the input capacitance of buffers and area/fringe capacitance of wires, determines the power consumption of the network. By replacing the number of buffers with total capacitive load, we can tune the compromise between clock skew and power for the design of global clock distribution network.

VI. Conclusions and Future Works

We propose a method to reduce clock skew using a completely symmetrical clock tree. The symmetric structure plays a critical role in our problem formulation and solutions. We completely explore all possibilities of symmetrical buffer placement by defining level-dependent blockages and revising the traditional dynamic programming flow. The flow has been integrated into the clock tree planning stage of NEC in-house CAD system.

The techniques in this work are mainly for the higher level global clock distribution. They can be combined with other techniques to minimize the final clock skews, such as link insertion [11] in the tree structure which reduces skews by local variations. With all these optimizations applied, the final design of clock distribution networks will have strong immunity to both global and local variations.

The limitation of this methodology is from the strong constraint of symmetry. We may have some distributions of circuit blocks with no possible solutions of symmetrical buffer placement in the clock tree. A relaxation on the constraint of symmetry can help to obtain feasible solutions while allowing deviation of buffer locations from perfect symmetry. For instance, we can use a cost function of (delay + $\alpha\times$max deviation + $\beta\times$#buffers) where the deviation value of each point can be stored in the $Blocked[p][l]$ array. Future works will address such formulations, which have larger and more flexible solution space, where we can find more feasible zero-skew or near zero-skew solutions. Eventually, a combined flow of floorplanning and clock-planning may provide best results on clock skew minimization.

References

[1] K. D. Boese and A. B. Kahng. Zero-skew clock routing trees with minimum wirelength. *IEEE Int. Conf. ASIC*, pages 1.1.1–1.1.5, 1992.

[2] C.-M. Chang, S.-H. Huang, Y.-K. Ho, J.-Z. Lin, H.-P. Wang, and Y.-S. Lu. Type-matching clock tree for zero skew clock gating. *ACM/IEEE Design Automation Conf.*, pages 714–719, 2008.

[3] T.-H. Chao, Y.-C. Hsu, and J.-M. Ho. Zero skew clock net routing. *ACM/IEEE Design Automation Conf.*, pages 518–523, 1992.

[4] M. Cho, S. Ahmed, and D. Z. Pan. Taco: temperature aware clock-tree optimization. *IEEE/ACM Int'l Conf. on Computer Aided Design*, pages 582–587, 2005.

[5] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein. *Introduction to Algorithms, Second Edition*, pages 323–369, 2001.

[6] L. P. V. Ginneken. Buffer placement in distributed rc-tree networks for minimal elmore delay. *IEEE Int'l Symp. on Circuits and Systems*, pages 865–868, 1990.

[7] M. Guthaus, D. Sylvester, and R. Brown. Clock buffer and wire sizing using sequential programming. *ACM/IEEE Design Automation Conf.*, pages 1041–1046, Jul. 2006.

[8] M. Jackson, A. Srinivasan, and E. S. Kuh. Clock routing for high-performance ics. *ACM/IEEE Design Automation Conf.*, pages 573–579, 1990.

[9] A. B. Kahng, J. Cong, and G. Robins. High-performance clock routing based on recursive geometric matching. *ACM/IEEE Design Automation Conf.*, pages 322–327, 1991.

[10] P. J. Restle et al. A clock distribution network for microprocessors. *IEEE Journal of Solid-State Circuits*, 36(5):792–799, May 2001.

[11] G. Venkataraman, N. Jayakumar, J. Hu, P. Li, and S. Khatri. Practical techniques to reduce skew and its variations in buffered clock networks. *IEEE/ACM Int. Conf. on Computer-Aided Design*, pages 592–596, 2005.

[12] K. Wang and M. Marek-Sadowska. Buffer sizing for clock power minimization subject to general skew constraints. *ACM/IEEE Design Automation Conf.*, pages 159–164, 2004.