Robust Chip-Level Clock Tree Synthesis for SOC Designs

Anand Rajaram
Department of ECE
University of Texas at Austin, Texas
anandr@mail.utexas.edu

David Z. Pan
Department of ECE
University of Texas at Austin, Texas
dpan@ece.utexas.edu

ABSTRACT

A key problem that arises in System-on-a-Chip (SOC) designs of today is the Chip-level Clock Tree Synthesis (CCTS). CCTS is done by merging all the clock trees belonging to different IPs per chip specifications. A primary requirement of CCTS is to balance the *sub-clock-trees* belonging to different IPs such that the entire tree has a small skew across all process corners. This helps in timing closure across all the design corners. Another important requirement of CCTS is to reduce clock divergence between IPs that have critical timing paths between them, thereby reducing maximum possible clock skew in the critical paths and thus improves yield. In this work, we propose effective CCTS algorithms to simultaneously reduce multi-corner skew and clock divergence. To the best of our knowledge, this is the first work that attempts to solve this practically important problem. Experimental results on several testcases indicate that our methods achieve 10%-31%(20% on average) clock divergence reduction and between 16-64ps skew reduction (1.6%-6.4% of cycle time for a 1GHz clock) with less than 0.5% increase in buffer area/wirelength compared to existing CTS algorithms.

Categories and Subject Descriptors

B.7.2 [**Hardware**]: Integrated Circuits

General Terms

Algorithms

Keywords

Clock Network, Chip-level CTS, Physical Design

1. INTRODUCTION

A System-on-a-Chip (SOC) design can be defined as "an IC, designed by stitching together multiple stand-alone VLSI designs to provide full functionality for an application" [1]. In today's 65nm/45nm VLSI technologies, SOC designs have become increasingly common and the trend is expected to continue in the future [2]. Most SOC physical design closure is done in a hierarchical fashion [1]. In such a methodology, different logical and physical partitions of the chip are timing closed independently [1–4] followed by a chip-level timing closure step. This chip-level timing closure includes CCTS in which a chip-level clock tree is synthesized to drive all the block-level clock trees. The primary objective of CCTS is that the full clock tree, which includes the chip-level and all the block-level clock trees, should be balanced and have less skew across all design corners. Satisfying this requirement is relatively easy when considering only the nominal delay corner. However, timing closure in most practical chips involve verifying timing across several corners that represent several global variation effects. This implies that the clock trees should have small skews across all the design corners. This is a very challenging task primarily because of the possible difference in the way the delays of the different sub-clock-trees scale, either because of difference in the clock structures or the relative significance of cell and interconnect delays. Another objective of CCTS is to minimize the clock divergence for the IPs with critical path between them. This helps to minimize skew variation between the critical timing paths between the IPs and thus improves the overall yield. In this work, we propose effective algorithms with the objective of addressing the above two objectives.

2. MOTIVATION

2.1 Significance of Clock Divergence Reduction

The significance of reducing clock divergence between registers in *timing-critical* paths is well known. For a given overall delay, the lesser the divergent delay between the such register-pairs, the lesser is the value of maximum skew (and skew variation) that can be seen between them. The same principle is also applicable at the chip-level where different sub-blocks interact with each other instead of register pairs.

2.2 Impact of Sub-block Clock Pin Location

Unlike hard IPs, the clock pins of the soft-IPs can be changed specific to a given chip and floorplan. This flexibility can be used towards clock divergence reduction between critical IPs. Figure 1 shows a simple example where the clock pin assignment might make a difference in clock divergence reducing.

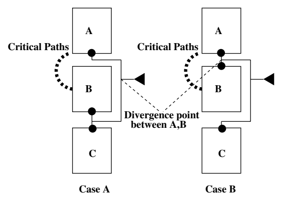

Figure 1: Pin location in Case B will result in reduced clock divergence between A and B.

2.3 Multi-corner skew reduction problem

Consider Figure 2 where only two sub-blocks are present. The squares in the sub-blocks represent clock sinks. The left-side block has bigger buffers with longer interconnects and the right-side block has smaller buffers with shorter interconnect. Let us assume that both sub-clock-trees have identical delays in the nominal corner. However, their delays across other corners will be different, mainly because of the difference in the interconnect lengths and buffer sizes. To balance these two sub-clock-trees across all corners, the chip-level clock tree should be built such that the differences in the delays, *across all corners*, between the sub-clock-trees gets compensated at the chip-level. In most SOC designs, there will be several sub-blocks having clock trees with significant differences in their size, structure, buffer sizes used and interconnect lengths. Thus, synthesizing a chip-level clock tree that can simultaneously reduce the skew across all corners by accounting for these differences while not significantly increasing the overall delay is a challenging problem.

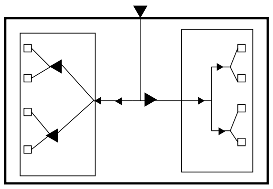

Figure 2: Delays of the two blocks will scale differently across corners due to different buffer sizes and interconnect lengths.

Problem Formulation: The CCTS problem is formulated using the following two sub-problems:
Given: Chip-level floorplan and criticality of clock divergence between all block pairs.
Problem: Select clock pin locations of all soft-IPs to reduce clock divergence of critical IP pairs.
Given: All information from previous step and information on sub-clock-tree delays/skews across corners.
Problem: Obtain a chip-level clock tree such that skews/delays across all corners are reduced, while simultaneously reducing the weighted sum of clock divergence between all IP pairs. The value of weight for a given IP pair is directly obtained from the number and timing criticality of paths between them.

***Tradeoff between divergence reduction and delay reduction*:** In some cases, we might be able to achieve lesser clock divergence by increasing the overall delay of the clock tree and vice-versa. One simple way to quantify this tradeoff is to use a scaling factor that will determine the percentage of delay increase that can be tolerated for a given reduction in clock divergence. Using this scaling factor, we can define the overall cost as follows:

$$Cost = x * Max\_Delay + (1-x) * DIV\_COST, \qquad (1)$$

where $DIV\_COST = \sum_{\forall i,j} W_{i,j} * \left({D_i}^F + {D_j}^F - 2 * {D_{i,j}}^C\right)$, x is $\in$ 01 quantifies delay Vs divergence tradeoff, Max_Delay is maximum delay to any sink in the tree, DIV_COST is the divergence cost between all IPs pairs, i,j are block numbers with $1 \leq i,j \leq N, i \neq j$;, $W_{i,j}$ is criticality of clock divergence between blocks i,j, ${D_i}^F$ is average root to flop delay in block i, ${D_{i,j}}^C$ is the max. *common* delay between two block pair i,j. All delay information are w.r.t. the nominal corner.

3. CLOCK PIN ASSIGNMENT ALGORITHM

Under the assumption that the clock pin locations for the sub-blocks are restricted to the mid points of the four sizes of the block, clock pin assignment problem can be formulated as follows. Let B_i denote the sub-block number i where $1 \leq i \leq N$. Let $W_{i,j}$ denote the criticality of the paths between blocks i and j. Also, let all the four possible clock pin assignments for a given block be denoted by ${B_i}^1$, ${B_i}^2$, ${B_i}^3$, ${B_i}^4$ where the pin locations are numbered starting from bottom point and proceeds in the clockwise direction. Let the pin selection for a given block be denoted by a set of four variables: ${x_i}^1$, ${x_i}^2$, ${x_i}^3$, ${x_i}^4$. Each of these variables can be either 0 or a 1 and the sum of the four will always be 1 to make sure exactly one of the four locations are selected. Using these definitions, the problem of clock pin assignment for clock divergence reduction can be formulated as:

$$Minimize : \sum {x_i}^p * {x_j}^q * W_{i,j} * Top\_Level\_Dist({B_i}^p, {B_j}^q) \quad (2)$$

$$s.t : \sum {x_i}^p = 1, \quad {x_i}^p \in \{0,1\}$$

$$1 \leq i,j \leq N, i \neq j; \quad 1 \leq p \leq 4; \quad 1 \leq q \leq 4;$$

where, i, j are block numbers, p, q denote the pin locations on a given block. $Top\_Level\_Dist({B_i}^p, {B_j}^q)$ represents the Manhattan distance between pin location p of B_i and q of B_j. The conditions that each of the variables ${x_i}^p$ should be either 0 or 1 and that the sum of all the variables for a given block should exactly be 1 makes sure that exactly one pin location is selected for each block. The cost function being minimized is the weighted sum of distances between all the clock pins of all block pairs where the weight is the criticality of the paths between a given block pair. The only variables in the above optimization problem are ${x_i}^p$ and since they can only take values of either 0 or 1, the above problem is a 0-1 Quadratic Programming problem and can be solved using efficient heuristics.

4. MULTI-CORNER SKEW REDUCTION

The multi-corner skew balancing problem for a given pair of sub-trees can be divided into two categories. In the first, the clock pins are located very close to each other *and* their delays across corners are very similar. Hence, the multi-corner skew balancing is trivial since it is possible to merge the clock pins with just interconnect without adding an extra buffer level. In the second case, the clock pins are far apart *and/or* they have significantly different delays across the corners. In such situations, we need to add one or more single-fanout buffer stages (with appropriate buffer sizes/interconnect lengths) to the root of the sub-trees to reduce their skews across corners. In future discussions, we call the selection of appropriate buffer size/interconnect length as selection of an appropriate *buffer configuration.* Figure 3 shows an example of a *buffer configuration.*

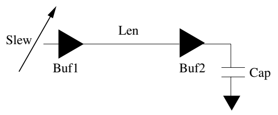

Figure 3: Buffer configuration used for multi-corner delay characterization.

To pick the right buffer configurations for multi-corner skew reduction, following properties of CCTS problem can be used:

- The CCTS problem will have just a hand full of end points (clock pins of sub-blocks) that are much more spread apart in distance than typical registers. This is because the number of sub-blocks in a typical SOC will be orders of magnitude lesser than the number of flops in the whole design.
- As a result, the average fanout for a buffer in the chip-level clock tree will be considerably less compared to the block-level clock trees. In most practical cases, this can be as low as 1 or 2.

In order to distinguish between the different buffer configurations and select the right set of configurations to achieve multi-corner skew reduction, we can follow the following steps:

- Restrict the maximum fanout for any chip-level clock buffer to just 1 or 2. The clock power/area penalty due to this restriction will be negligible because the fanout of most buffers is expected to be small anyway.
- The fanout restriction reduces the number of possible buffer configurations, enabling us to do the multi-corner delay characterization for each configuration quite easily. For example, for the setup shown in Figure 3, it is trivial to complete multi-corner characterization w.r.t. all variables shown.

- Next, we obtain *cross-corner delay ratios* (CCDR) for all buffer configuration by dividing the delays across all N corners with the nom-corner delay. After this, each configuration will have a *vector* of N numbers called its CCDR. This normalization helps us to compare the *relative* cross-corner scaling of different buffer configurations and choose the appropriate one during sub-tree mergers.

```
Procedure: $Multi\_Corner\_Subtree\_Balance(S_A, S_B)$
Input: Location and delay information for both sub-trees $S_A$ & $S_B$.
Output: New sub-tree $S_C$ combining $S_A$ & $S_B$.
1. Get CCDRs of $S_A$, $S_B$ w.r.t. nominal corner.
2. Set $Sub\_trees\_not\_close = 1$
3. While ( $Sub\_trees\_not\_close == 1$ )
    (i) Pick sub-tree with min nom-corner delay, denoted by $S_P$.
        Let $S_Q$ be the sub-tree with max nom-corner delay.
    (ii) Select best buffer config. to add to $S_P$ such that $CCDR$ of
        $S_P$ after adding the buffer config. moves closest to
        CCDR value of max delay sub-tree without exceeding its delay
        significantly(i.e. > biggest buffer delay). Since CCDRs are
        vectors, the closeness is defined by $L^2$ norm between them.
    (iii) Update delay and CCDR information for $S_P$.
    (iv) If skew across corners between sub-trees less than limit
            $Sub\_trees\_not\_close = 0$
4. Using selected buffer configurations, identify the manhattan ring
   within which each sub-tree's new root can be located.
5. If (Two manhattan rings intersect)
       Any point within intersecting area can be the new root.
       Do wire-snaking to ensure preservation of skews/delays.
   Else
       Select closest points on the rings to reduce wirelength.
       Merge them by constructing a simple symmetric (0 skew) tree.
6. Name the new sub-tree as $S_C$ and return $S_C$.
```

Figure 4: Multi-corner skew balancing heuristic.

The concept of CCDR described above is used in our multi-corner sub-tree merging heuristic shown in Figure 4. In the above procedure, we first pick the sub-tree, denoted by S_P, with lesser nominal corner delay and recursively add buffer configurations at its root such that the CCDR of the two sub-tree moves closer. This is repeated till the delays of both the sub-trees are fairly close across all corners. At this point, the exact configurations to be added at the roots of both sub-trees A and B are available. However, the merging point location of the two sub-trees is still not yet fixed. For a given sub-tree, the total lengths of all the interconnects added with buffer configuration gives the radius of the Manhattan ring within which its root pin is to be located. If the Manhattan rings of both sub-trees intersect, then any point within the intersection can be selected as the root with appropriate wire-snaking. If the Manhattan rings do not overlap, it means that though the two sub-trees have similar delays, we need to add more buffer levels to physically merge them. To achieve this, we identify the closest points/segments on the two Manhattan rings and merge them with a perfectly symmetric tree. This will preserve the multi-corner skew balancing between the two sub-trees.

5. CHIP-LEVEL CTS ALGORITHMS

In this section, we briefly describe three modifications of existing CTS algorithms (used for comparison purposes) and discuss our dynamic programming based algorithm in detail.

5.1 Existing Algorithms

Single-Corner DME based Approach: This algorithm is a direct application of existing CTS algorithm of [5] using only nominal corner delays.

Multi-Corner DME based Approach: This approach is identical to the single-corner approach with the difference that multi-corner sub-tree merging method described in Figure 4 is used instead of using only the nominal corner delay.

Greedy CCTS Algorithm: This algorithm is a simple modification of [6] in which the sub-trees to be merged are selected to minimize the cost defined by Equation(1). The node pair merger is done using the multi-corner skew reduction method of Figure 4.

```
Procedure: $Dynamic\_Programming\_Top\_Level\_CTS$
Input: Location and delay information for all blocks.
Output: Chip-level clock tree with min delay & clock divergence.
1. Initialize
   a. Mergers_Completed = 0
   b. Active_SubTrees = Clock Pins of all blocks
   c. For each subtree ∈ Active_SubTrees
        Status(subtree) = new.
2. While ( Mergers_Completed == 0 )
   a. Valid_Pairs = Pick_Valid_Pairs(Active_SubTrees)
   b. Eliminated_Pairs = Pre_Eliminate(Valid_Pairs)
   c. Valid_Pairs = Valid_Pairs - Eliminated_Pairs
   d. Generate_Cost for merger of each Valid_Pairs using Eq.(1)
   e. Potential_SubTrees = Active_SubTrees + Valid_Pairs
   f. Eliminated_SubTrees = Post_Eliminate(Potential_SubTrees)
   g. New_Additions = Potential_SubTrees - Eliminated_SubTrees
                      - Active_SubTrees
   h. For each subtree ∈ Active_SubTrees
        Status(subtree) = old.
   i. For each subtree ∈ New_Additions
        Status(subtree) = new.
   j. Active_SubTrees = Potential_SubTrees - Eliminated_SubTrees
   k. if (No. of New_Additions == 0)
        Mergers_Completed = 1
3. Pick sub-tree in Active_SubTrees with all
blocks and min delay and clock divergence.
```

Figure 5: Dynamic Programming CCTS. The sub-steps defined in Figures 6, 7, 8.

5.2 Dynamic programming CCTS Algorithm

Our dynamic programming based CCTS algorithm is shown in Figure 5. As with any dynamic programming based approach, two key aspects of our algorithm are optimal solution of sub-problems and effective pruning of inferior sub-solutions to avoid exponential run-time. For subsequent discussions, we use the following terminologies. An *active sub-tree* is one that has not yet been eliminated/pruned from subsequent merging operations. The list of active sub-trees represent the current list of sub-solutions to CCTS problem. A *new* sub-tree in the list of active sub-trees is one that has not gone through even a single round of mergers with other active sub-trees.

After initialization in step 1 of Figure 5, the existing sub-trees are iteratively combined in step 2 to get one or more solutions that drive all target clock pins. In each iteration , the set of valid sub-tree pairs that can be merged to create new sub-trees are picked. The steps to consider only the non-eliminated sub-tree pairs are outlined in Figure 6. The sub-trees that have gone through one round of merging with each other are marked as *old*. Any merger between two *old* nodes is invalid as it would have happened in one of the previous iterations. Also, any merger between two solutions that have overlapping list of target clock pins is also invalid as it is physically infeasible.

```
Procedure: $Pick\_Valid\_Pairs(Active\_SubTrees)$
Input: All active sub-trees.
Output: All pairs of sub-trees that are valid for merger.
1. Valid_Pairs = {}; Number sub-trees from 1 to $N$.
2. For $i = 1$ to $N$
     For $j = i$ to $N$
       Sub-tree pair considered: $S_i$ and $S_j$
       If (Status($S_i$) == new OR Status($S_j$) == new ) AND
       If (No overlap between $S_i$ and $S_j$ on block clock-pins driven)
         Valid_Pairs → Valid_Pairs + ($S_i$, $S_j$)
3. Return Valid_Pairs
```

Figure 6: Procedure to pick valid pairs for merger.

The pre-elimination procedure of Figure 7 weeds out sub-trees that can be safely removed from consideration even before actual merger. For example, if two sub-trees are located very far away from each other, then it is a good choice to eliminate the pair because merging them will cause a significant increase in overall clock tree delay. After pre-elimination, the procedure in Figure 5 updates the list of valid pairs. Next, each of the eligible sub-tree pairs are merged using the multi-corner sub-tree balancing algorithm of Figure 4 at the end of which, each sub-tree will have a specific cost as defined by Equation(1). After this, the set *Potential_SubTrees* is created by merging the

TC	PAM	CCTS Meth.	Delay(ps)			Skew(ps)			Divergence(ps)			BA	WL	CPU
			Nom	Slow	Fast	Nom	Slow	Fast	Nom	Slow	Fast	nm^2	um	(s)
Avg (20 TCs)	RND	1C-DME	2314	2909	1853	27	119	94	237103	297218	190267	3231469	16326298	0.5
		MC-DME	2313	2859	1883	74	99	69	235085	290498	191429	3234319	16332297	1
		MC-GRD	2295	2840	1863	76	97	83	237314	268882	212592	3245459	16365661	7
		MC-DyP	2305	2848	1879	63	82	63	193900	239022	157705	3239869	16350275	32
	QP	1C-DME	2315	2912	1856	37	113	104	231896	290863	185877	3225499	16308305	1
		MC-DME	2346	2900	1911	69	97	64	231445	286111	188578	3229589	16314602	2
		MC-GRD	2298	2840	1866	66	88	78	229345	259050	206075	3235339	16338849	9
		MC-DyP	2331	2888	1904	43	57	40	185953	229559	151434	3234539	16329956	49

Table 1: Results demonstrating the effectiveness of the proposed pin assignment and CCTS algorithms.

```
Procedure: Pre_Eliminate(Valid_Pairs)
Input: All Valid Pairs of sub-trees.
Output: All very bad merging choices.
1. Eliminated_Pairs = {}. Number Valid_Pairs 1 to $V$.
2. For $i = 1$ to $V$
     Let $S_A$ and $S_B$ be sub-trees of the pair $i$.
     If (dist($S_A, S_B$) > $Dist\_Threshold$) OR
     If (delay_diff($S_A, S_B$) > $Delay\_Threshold$)
       Mark Pair $i$ for potential elimination.
3. For each Pair $i$ marked for potential elimination
     Let $S_A$ and $S_B$ be sub-trees of the pair $i$.
     If ($S_A$ and $S_B$ have merging pairs not marked for elimination)
       Eliminated_Pairs → Eliminated_Pairs + Pair $i$
3. Return Eliminated_Pairs
```

Figure 7: The pre-eliminate procedure.

existing active sub-trees and the valid pairs for which merging cost is available.

```
Procedure: Post_Eliminate(Potential_SubTrees)
Input: Full list of potentially valid subtrees.
Output: Sub-treesPairs that can be pruned because of existence of
        other dominating sub-trees.

1. Eliminated_SubTrees = {};
   Number Potential_SubTrees sub-trees from 1 to $N$.
2. For $i = 1$ to $N$
     For $j = i$ to $N$
       P,Q → List of all sub-block clock pins driven by sub-tree $i$,$j$.
       If (P $\subseteq$ Q AND cost(P) $\geq$ cost(Q))
         Eliminated_SubTrees → Eliminated_SubTrees + P;
       If (Q $\subset$ P AND cost(Q) > cost(P))
         Eliminated_SubTrees → Eliminated_SubTrees + Q;
3. Return Eliminated_SubTrees
```

Figure 8: The post-eliminate procedure.

In the post elimination (Figure 8), all inferior solutions are eliminated. A sub-tree P is inferior if there exist another sub-tree Q that covers the same set (or a super-set) of clock pins covered by sub-tree P with a lower merging cost. After post elimination, all existing active sub-trees are marked as *old* as all possible valid sub-tree pairs from these have been evaluated already. However, the newly created sub-trees should be allowed to merge, with each other and also with old sub-trees. So the newly created sub-trees are marked as *new*. The termination of the iterations in Figure 5 occurs when there are no new sub-trees that were added in any given loop.

6. EXPERIMENTAL RESULTS

Using the data available on SOC chips in the literature [1–4], we define reasonable *ranges* for a few key parameters such as the size of the chip, aspect ratio, number of sub-blocks etc. and randomly generate the testcases with information on chip-level floorplan, timing criticality information and block-level CTS delays across corners. We use the 65nm model cards from [7] for generation of delays across three device corners (Nom, FF, SS). The four algorithms described in Section 5 are run on two sets of testcases: one with random pin placement and the other with Quadratic Programming pin placement. Since the two testcase sets are identical except in pin locations, comparing their results will indicate the effectiveness of clock pin placement algorithm. We also run each of the four CCTS algorithms on all testcases to compare the effectiveness of the four CCTS algorithms. We generate 20 random testcases with unique floorplans. Due to page limit, we only present the average results from all 20 testcases in Table 1. The acronyms used in Table 1 and 2 are: TC is Test Case, PAM is the Pin Assignment Method used (either Quadratic-Programming (QP) or random pin assignment (RND)), 1C-DME, MC-DME, MC-GRD, MC-DyP denote the single corner DME, multi-corner DME, multi-corner greedy and multi-corner dynamic programming respectively. The divergence values given are weighted sum of clock divergence between all IP pairs. All the results in Table 1 have been normalized and summarized in Table 2 *with the exception of skew*. Since skew is a difference, skew ratios do not mean anything physically. So we measure the effect of skew reduction in absolute pico-seconds. The actual significance of the skew reduction depends on the clock frequency of the chip. Based on Table 1,2 , we can observe the following:

- Our algorithms significantly reduce skew across corners (1.6 % to 6.4% of cycle time even for a 1GHz clock) than the single corner approach.
- Our dynamic programming approach achieves an 18% reduction in average divergent delay across all three corners with 0.5% increase in buffer area/wirelength.
- Compared to 1-corner DME with random pin placement, the dynamic programming approach with QP pin placement achieves 20% average clock divergence reduction.

PAM	CCTS Meth	Delay(Ratio)			Divergence(Ratio)			BA Ratio	WL Ratio
		Nom	Slow	Fast	Nom	Slow	Fast		
RND	1C-DME	1.00	1.00	1.00	1.00	1.00	1.00	1.000	1.000
	MC-DME	1.00	0.98	1.02	0.99	0.98	1.01	1.001	1.001
	MC-GRD	0.99	0.98	1.01	1.00	0.90	1.12	1.004	1.002
	MC-DyP	1.00	0.98	1.01	0.82	0.80	0.83	1.002	1.001
QP	1C-DME	1.00	1.00	1.00	0.98	0.98	0.98	0.998	0.998
	MC-DME	1.01	1.00	1.03	0.98	0.96	0.99	0.999	0.999
	MC-GRD	0.99	0.98	1.01	0.97	0.87	1.08	1.001	1.001
	MC-DyP	1.01	0.99	1.03	0.78	0.77	0.80	1.001	1.002

Table 2: Summary of normalized results from Table 1.

7. CONCLUSION

We have proposed efficient algorithms for soft-IP clock-pin assignment, multi-corner skew reduction and chip-level CTS problems. Experimental results on several testcases show that these algorithms are effective in simultaneous reduction of multi-corner skew and clock divergence between critical IP pairs.

8. ACKNOWLEDGEMENTS

This work is supported in part by NSF, SRC, IBM Faculty Award, Fujitsu, Qualcomm, Sun, and equipment donations from Intel.

9. REFERENCES

[1] R. Rajsuman, "System-on-a-chip: Design and Test", 2000 Artech House Inc. Publishers.

[2] M. Keating, P. Bricaud "Reuse Methodology Manual for System-on-a-Chip Designs", Third Edition, 2002, Kluwer Academic Publishers.

[3] S. Agarwala et. al "A 800 MHz System-on-Chip for Wireless Infrastructure Applications," in *VLSI Design 2004*.

[4] S. Agarwala et. al "A 65nm C64x+ Multi-Core DSP Platform for Communications Infrastructure," in *IEEE ISSCC*, 2007, pp.262–264

[5] Masato. Edahiro, "A clustering-based optimization algorithm in zero-skew routings," in *Proc. of DAC*, 1993, pp. 612–616.

[6] R. Chaturvedi, and J. Hu, "An efficient merging scheme for prescribed skew clock routing" in *IEEE TVLSI*, vol.13, no.6, pp. 750–754, Jun'05.

[7] "http://www.eas.asu.edu/p̃tm/"