Clustering Based Fast Clock Scheduling for Light Clock-Tree

Makoto Saitoh, Masaaki Azuma, and Atsushi Takahashi

Tokyo Institute of Technology
Department of Communications and Integrated Systems
2-12-1 Ookayama, Meguro-ku, Tokyo 152-8552, Japan
TEL: +81-3-5734-2665, FAX: +81-3-5734-2902
E-mail : {makoto, azuma, atushi}@lab.ss.titech.ac.jp

Abstract

We introduce a clock schedule algorithm to obtain a clock schedule that achieves a shorter clock period and that can be realized by a light clock tree. A shorter clock period can be achieved by controlling the clock input timing of each register, but the required wire length and power consumption of a clock tree tends to be large if clock input timings are determined without considering the locations of registers. To overcome the drawback, our algorithm constructs a cluster that consists of registers with the same clock input timing located in a close area. In our algorithm, first registers are partitioned into clusters by their locations, and clusters are modified to improve the clock period while maintaining the radius of each cluster small. In our experiments for an industrial data of 888 registers, the clock period achieved is 27% shorter than that achieved by a zero-skew clock tree, and 1% longer than the theoretical minimum. The computational time is about 24.9 seconds and the wire length and power consumption of the clock tree is comparable to these of a zero skew tree.

1. Introduction

A *semi-synchronous circuit* is a circuit in which the clock is assumed to be distributed periodically to each individual register, though not necessarily to all registers simultaneously. Among various objectives in the synthesis of high-performance circuits, the clock period minimization is the primal subject. For a given circuit with fixed signal propagation delays between registers, there exists a lower bound of the clock period in semi-synchronous framework which is usually smaller than the maximum signal delay between registers. This lower bound is achieved if the clock is distributed to each register at proper timing [11, 8, 19].

The *clock timing* of register is the difference in clock arrival time between the register and an arbitrary chosen (perhaps hypothetical) reference register. The set of clock timings is called a *clock schedule*.

It is shown that an arbitrary clock schedule can be realized by constructing a clock tree [18]. However the total wire length of a clock tree that realizes a clock schedule depends on the clock schedule. The required wire length for a clock schedule tends to be large if it is determined without considering the locations of registers. It is experimentally shown that the required wire length for a random clock schedule is larger than that for a gentle clock schedule (A clock schedule is said to be gentle if the clock-timings of registers are set to near when their locations are close to each other) [13]. In practice, the allowable wire length and power consumption of a clock tree would be those of a zero-skew clock tree. Thus our problem is to find a clock schedule that achieves a smaller clock period and that can be realized with the wire length at least comparable to or smaller than that of a zero-skew clock tree.

Many clock tree algorithms have been proposed to reduce the wire length and power consumption under the framework of zero skew [2, 3, 4, 9, 10], bounded skew [6, 7, 12], useful skew [21, 22], and associative skew [5]. However, they did not fully utilize the flexibility of clock schedule.

The flexibility is utilized to improve the circuit performance by combining the retiming in [16], and to improve the circuit reliability in [14]. However, the realization of a clock schedule is not considered at all. In [20], a practical clock tree algorithm was introduced in which a discrete clock timing is assigned to each register. It is experimentally shown that the clock period of a circuit is improved about 10% compared against the circuit with a zero skew clock tree, and the wire length of the clock tree is comparable to the zero skew clock tree. By simulations using vender tools, the circuits obtained are proved stable under various practical conditions. However, it takes more than one hour to determine a clock schedule for a problem of about one thousand registers, since the clock schedule algorithm is based on a simulated annealing.

In this paper, we propose a fast clock schedule algorithm that achieves a shorter clock period and that takes the realization cost of a clock tree into account. In the algorithm, a cluster of registers with the same clock input timing located in a close area is constructed. The registers in each cluster are connected by a Steiner tree and driven by a buffer. To make the lengths of the intra-cluster wire and the inter-cluster wire small, the number of clusters and the radius of each cluster should be small in addition to achieve a shorter clock period. In order to get such a desirable clustering, the algorithm first partitions registers into clusters by their locations, and modifies clusters to improve the clock period while maintaining the number of clusters and the radius of each cluster small. In each repetition of modification, a set of critical registers with respect to the clock period is selected. Each register in the set is moved to a near or a new cluster in order to relax the timing constraints.

In experiments, the algorithm is applied to an industrial data of 888 registers. The computational time to obtain the clock schedule is about 24.9 seconds by PentiumII 450Mhz. The clock period achieved is 27% shorter than that achieved by a zero-skew clock tree, and 1% longer than the theoretical minimum without considering the realization of clock schedule. To confirm the realization cost of the obtained clock schedule, a clock tree that realizes the clock schedule is constructed by the algorithm proposed in [1]. The clock tree algorithm consists of two phases, intra-cluster routing and inter-cluster routing. A procedure based on the cost-radius balanced Steiner tree algorithm (CRBST) [17] and that based on the schedule clock tree algorithm (SC) [13] are used in intra-cluster routing and in inter-cluster routing, respectively. To reduce the wire length and power consumption, the flexibility of clock schedule is taken into account in inter-cluster routing. It is shown that the clock tree constructed is comparable to a zero skew clock tree and that the desirable clock schedule is obtained in a short time.

2. Preliminaries

We consider a circuit with a single clock consisting of registers and combinatorial circuits between them. The *clock timing* $s(v)$ of register v is the difference in clock arrival time between v and an arbitrary chosen (perhaps hypothetical) reference register. The set of clock timings is called a *clock schedule*.

We assume the framework that a circuit works correctly if the following two types of constraints are satisfied for every register pair with signal propagation [11]:

No-Double-Clocking (Hold) Constraints :

$$s(v) - s(u) \leq d_{\min}(u,v)$$

No-Zero-Clocking (Setup) Constraints :

$$s(u) - s(v) \leq T - d_{\max}(u,v)$$

where T is the clock period and $d_{\max}(u,v)$ ($d_{\min}(u,v)$) is the maximum (minimum) propagation delay from register u to register v along a combinatorial circuit. These constraints are represented by the *constraint graph*.

The constraint graph $G(V,E)$ is defined as follows: a vertex $v \in V$ corresponds to a register, a directed edge $(u,v) \in E$ corresponds to either type of constraints; an edge (u,v) corresponding to the no-double (no-zero) clocking constraint is called D-edge (Z-edge), and the weight $w(u,v)$ of the edge is $d_{\min}(u,v)$ ($T - d_{\max}(v,u)$). An edge (u,v) is said to be *legal* if $s(v) - s(u) \leq w(u,v)$, *illegal* otherwise. The slack of an edge (u,v) is defined as

$$\Delta(u,v) = s(u) + w(u,v) - s(v).$$

If the slack of an edge is 0, the edge is said to be *critical*. A cycle (path) consisting of critical edges in G is called a critical cycle (path). A clock schedule is called *feasible* if there is no illegal edge in G.

For given the maximum and minimum propagation delays between registers, the minimum feasible clock period,

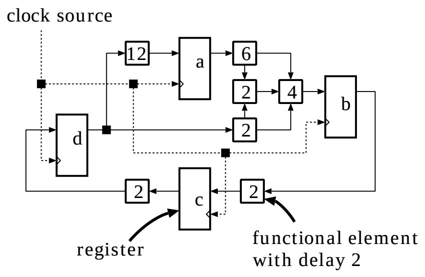

Figure 1. Circuit

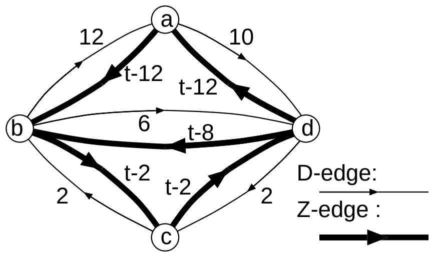

Figure 2. Constraint graph $G_{T=t}$

under the assumption that the clock timing of every register can be controlled, can be determined by using the constraint graph G [19]. Note that the constraints can be satisfied if and only if G contains no negative cycle [15, 19]. The constraint graph G when $T = t$ is denoted by $G_{T=t}$. For example, the constraint graph $G_{T=t}$ of the circuit shown in Fig 1 is shown in Fig 2. The smallest clock period t such that $G_{T=t}$ contains no negative cycle is denoted by $T(G)$. Note that there exists a critical cycle in $G_{T=t}$ if and only if $t = T(G)$.

A feasible clock schedule can be obtained if the constraint graph contains no negative cycle. One way to get a feasible clock schedule is as follows: choosing an arbitrary vertex in the constraint graph, let the clock timing of each register be the weight of a shortest path from the chosen vertex to the vertex corresponding to the register. Note that a feasible clock schedule is not unique in general.

Let $r(v) = [s_{\min}(v), s_{\max}(v)]$ be a range of clock timing of register v. The set r of ranges is called consistent if a feasible clock schedule is obtained whenever the clock timing of every register v is chosen from $r(v)$. One way to get a consistent set of ranges is as follows: find a feasible clock schedule s; let

$$r(v) = [s(v) - \frac{1}{2} \min_{(v,u)\in E} \Delta(v,u), s(v) + \frac{1}{2} \min_{(u,v)\in E} \Delta(u,v)].$$

Note that a consistent set of ranges is not unique in general.

In this paper, the registers are partitioned into several clusters such that the clock timings of registers in each cluster are constrained to be equal. The constraint graph G^C, which is a constraint graph under this constraint, is obtained from G by contracting vertices in each cluster into one vertex. The minimum feasible clock period $T(G^C)$ and a feasible clock schedule s^C under this constraint can be obtained by using G^C. In this case, the clock timing of a register v is denoted by $s^C(v)$.

3. Algorithm

The algorithm partitions registers into several clusters. The registers in each cluster are driven by a buffer. In each cluster, the clock timings of registers are assumed to be equal. This assumption makes sense when the routing delay which is caused by the resistance of the wire connecting from the driving buffer to registers can be ignored compared with the gate delay which is caused by the resistance of a buffer. In order to make the assumption valid, the algorithm bounds the radius of the cluster, the radius of the minimum bounding regular rhombus that covers locations of registers in the cluster, so that the maximum wire length from the driving buffer to each register can be bounded. Note that the required intra-cluster wire length would be small by this bound. Also the reliability of the circuit would be improved since the deviation of clock delay caused by the deviation of routing delay is suppressed [20].

By assuming that the clock delay from a clock source to each driving buffer can be controlled, the algorithm improves clustering to reduce the clock period. As the number of clusters is increased, the minimum feasible clock period becomes shorter, but the required inter-cluster wire length would be larger. This is because not only the number of cluster is increased, but also the resultant clock schedule becomes random. Thus, in order to make the inter-cluster wire length smaller, the number of clusters is controlled as small as possible after taking the driving ability of a buffer into account.

The outline of the proposed algorithm CBCS is shown in Figure 3. In the outline, $c(v)$ denotes the cluster that contains register v, and $h(v)$ dose the rhombus in which v is located.

Algorithm CBCS

Input
- the maximum (minimum) propagation delay between registers
- the location of each register
- the maximum radius of a cluster

Output
- A partition of registers into clusters, and the corresponding minimum feasible clock period and a clock schedule.

1. Partition the chip area into rhombuses with oblique lattice, and let registers in each rhombus be an initial cluster associated with the rhombus.
2. Construct the constraint graph G^C, compute the minimum feasible clock period $T(G^C)$ and find a feasible schedule s^C in $T(G^C)$.
3. Find a feasible clock schedule s' in clock period $T(G^C) - \delta$ such that $\sum_{v \in V} |s'(v) - s^C(v)|$ is minimum where δ is small value. Let S be the set of registers v such that $s'(v) \neq s^C(v)$.
4. For each register v in S, apply the following: find a cluster $c'(v)$ such that $c'(v)$ is associated with nine rhombuses adjacent to $h(v)$ and that the intersection of the consistent range of v and that of $c'(v)$ is maximum; move v from $c(v)$ to $c'(v)$.
5. If there exists a register v in S such that $c(v) \neq c'(v)$ in Step 4, then return to Step 2.
6. If there exists a register v in S such that no new cluster associated with $h(v)$ has been created, then create a new cluster that contains v and return to Step 2.
7. For each register v in each new cluster, and for each register v that is contained in an original cluster not associated with $h(v)$, move to the original cluster in $h(v)$ if the intersection of the consistent range of v and that of the cluster is not empty.
8. Output clusters, minimum clock period, and clock schedule.

Figure 3. Outline of proposed algorithm

In Step 1, the size of rhombus is bounded so that the maximum Manhattan length within the adjacent nine rhombuses is at most twice the maximum radius of a cluster and that a buffer can drive the registers in each cluster. By restricting the registers of a cluster to adjacent nine rhombuses, routing delays could be kept small enough to be ignored. In Step 2, we compute the minimum clock period in $T(G^C)$ and find a feasible schedule s^C by the clock schedule algorithm in [23]. Note that there is a critical cycle in $G^C_{T=T(G^C)}$. If there is a critical cycle in $G_{T=T(G^C)}$, then it is impossible to reduce the clock period by modifying the clustering. Otherwise, the clock period could be reduced by modifying the clustering. Thus, in the following, we assume that the graph obtained from $G_{T=T(G^C)}$ by deleting non-critical edges becomes a directed acyclic graph. In Step 3, S is defined as the set of registers v such that $s'(v) \neq s^C(v)$. The registers on a critical path P in $G_{T=T(G^C)}$, except one register, are contained in S if P contains a Z-edge. A register not incident to a critical edge is not contained in S since δ is chosen small. By assuming that there is no critical cycle consisting only of D-edges, each critical cycle in $G^C_{T=T(G^C)}$ is broken if each register v in S is removed from $c(v)$ and a new cluster consisting only of v is created. However the number of clusters should be small to make the wire length of the clock tree small. Thus we move a register v from $c(v)$ to $c'(v)$ in Step 4 since no new critical cycle is formed if the consistent range of a register v and that of a cluster C is neither empty nor unique. Note that S might be redundant to break all the critical cycles. Thus, we return to Step 2 when at least one register in S is moved in Step 4 by expecting that critical cycles might be broken. The number of new clusters for each rhombus is restricted to at most one to keep the number of clusters small. In Step 6, one new cluster is created, if possible, and return to Step 2. In Step 7, a register moved from the original cluster is returned to it if possible. A register moved from the original cluster might be returned to it if other registers are moved from it.

The consistent set of ranges of clusters are obtained by using feasible schedule s when Step 2 is executed. It is also updated when the cluster accepts a register at Step 4, but postponed when a register is removed from the cluster for the computation time. The consistent set of ranges of registers are obtained by using feasible schedule s' when Step 3 is executed.

Although the number of registers in S might not be important to get an optimal clustering, the size of S is small in most cases since the graph obtained by deleting non-critical edges seldom become complicated one. In case that the setup and hold times of registers are taken into account, the weight of a D-edge might be negative, and a critical cycle

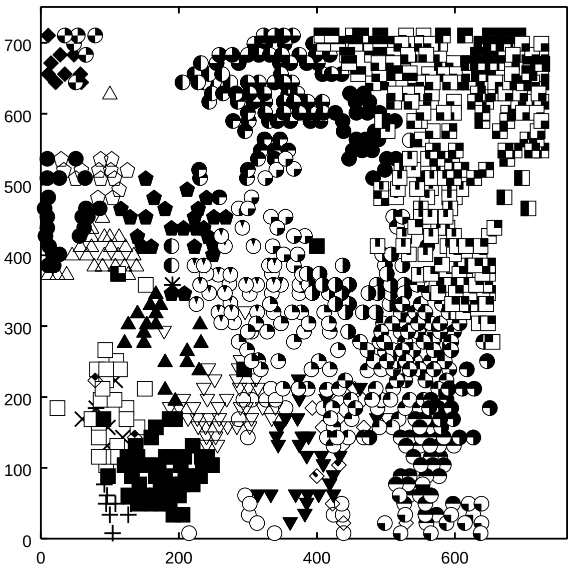

Figure 4. Clustering result (49 clusters)

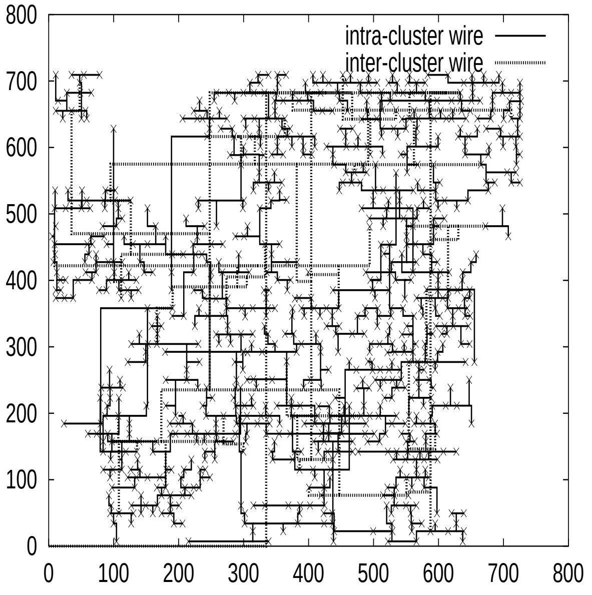

Figure 5. Clock tree layout (8391[*ps*])

consisting only of D-edges might exist. In such case, the clock schedule s' is obtained using the constraint graph obtained by reducing the weight of each edge by δ. However we use the constraint graph obtained by reducing the clock period by δ since it has been implemented.

4. Experiments

The algorithm CBCS is applied to the circuit of 888 registers placed within 728×710 [μm^2] which comes from an industry.

The maximum propagation delay between registers, that is, the minimum clock period in ordinary complete synchronous framework, is 11569 [*ps*], and the minimum clock period in semi-synchronous framework when each cluster consists of one register is 8323 [*ps*]. The maximum radius of a cluster is set 300 [μm].

The minimum feasible clock period of the initial clustering of 37 clusters is 10154 [*ps*]. After Step 2 is executed 52 times, the minimum feasible clock period becomes 8391 [*ps*]. At this point, the number of clusters is 51 and the number of register v not contained in the original cluster associated with $h(v)$ is 103. In Step 7, six registers are moved to their original clusters and the number of cluster becomes 49. The total computational time is about 24.9 seconds by PentiumII 450Mhz. The clustering result is shown in Figure 4. The registers in each cluster are plotted by the same symbol in Figure 4.

In order to confirm the quality of clustering, the clock trees are constructed by the algorithm for low power proposed in [1]. The clock tree algorithm consists of intra-cluster routing and inter-cluster routing. In intra-cluster routing, CRBST [17] is used to obtain a small Steiner tree with radius constraint. In CRBST, the maximum allowable path length from the source to sinks can be set by a parameter. In order to make the routing delay negligible, we set the maximum radius of a cluster as the maximum allowable path length for each inter-cluster routing. In inter-cluster routing, the clock timing of each cluster within the consistent range is achieved by the modified SC [13]. The modified SC is based on Differed-Merge-Embedding strategy with buffer insertion. In the bottom up merging phase, a pair that can be merged by a short interconnection with lower power is selected recursively. The maximum path length from a driving buffer and the maximum driving capacitance of a buffer are also constrained in inter-cluster routing.

In Table 1, the statistics of clock trees are shown. The columns consist of type of clustering, clock period, total wire length, number of inserted buffers, and power consumption. The total wire length is made of lengths of the intra-cluster wire and the inter-cluster wire.

The clock trees in “Flat” obtained without clustering. In the clock tree algorithm, intra-clustering routing is skipped. The zero skew clock trees are obtained by ignoring the consistent range of clock schedule. The clock trees in “Initial(37)” obtained by the initial clustering. The clock trees in “Final(49)” obtained by the final clustering. The layout of the clock tree that achieves the clock period 8391[*ps*] is shown in Figure 5. The clock tree in “Industry” comes from an industry. In the comparison between “Industry” and others, there are several differences about conditions. For example, relatively rough delay model is used in our experiments. Though we believe that the result is not affected significantly.

The clock trees obtained by clustering are better than those in “Flat” with respect to both wire length and power consumption. In constructing a clock tree in “Flat”, the clock schedule is determined without considering the locations of registers. So many buffers are used and the wire length and power consumption are large. With respect to the wire length and the power consumption, the clock trees in “Initial(37)” are better than others. But the minimum clock period is more than 10000[*ps*]. Whenever clustering is modified in order to achieve shorter clock period, the required wire length and power consumption are increased

	clock period	wire length	(intra, inter)	#buf	power
	$[ps]$ (%)	$[\mu m]$ (%)	$[\mu m]$		$[\mu W/MHz]$ (%)
Flat	8391 (73)	39857 (154)	(— , —)	207	76.4 (122)
	9000 (78)	35137 (135)	(— , —)	182	69.0 (110)
	10000 (86)	31067 (120)	(— , —)	156	61.4 (98)
	11569 (100)	29156 (111)	(— , —)	138	57.5 (91)
(zero skew)	11569 (100)	35066 (135)	(— , —)	163	65.6 (104)
Initial(37)	10154 (88)	23540 (91)	(15410, 8130)	56	51.1 (81)
	11569 (100)	22333 (86)	(15410, 7033)	53	49.9 (80)
(zero skew)	11569 (100)	23828 (92)	(15410, 8418)	63	52.5 (84)
Final (49)	8391 (73)	30317 (117)	(19051, 11266)	90	63.4 (101)
	9000 (78)	28706 (111)	(19051, 9655)	73	59.4 (95)
	10000 (86)	28267 (109)	(19051, 9216)	69	58.4 (93)
	11569 (100)	28032 (108)	(19051, 8981)	64	57.4 (91)
Industry	11569 (100)	25947 (100)	(—, —)	84	62.8 (100)

Table 1. Clock tree statistics

since the radius of each cluster becomes large or the number of clusters is increased. By using the clustering "Final(49)", the clock period 8391 $[ps]$ can be achieved. In constructing a clock tree in "Final(49)", the shorter we set the clock period, the larger wire length is and the higher power consumption is. The clock routing becomes harder as the clock period was shorter since the consistent range of each register becomes narrower.

5. Conclusion

In this paper, we proposed a fast clock schedule algorithm that achieves a smaller clock period and that takes the register locations into account. In experiments, the clock period is reduced 27% compared to the complete synchronous framework. The cost of clock tree that realizes the obtained clock schedule is shown to be comparable to the zero skew tree in experiments.

For the future works in order to obtain further smaller clock tree, it is necessary to determine the clock timing of a cluster and its consistent range by taking the characteristics of the clock tree and its construction algorithm into account.

Acknowledgments

The authors are grateful to Dr. Yoji Kajitani, Professor of Tokyo Institute of Technology, Dr. Masahiko Toyonaga, Mr. Keiichi Kurokawa, and Mr. Takuya Yasui, Matsushita Electric Industrial Co., Ltd, for their helpful discussion. This work is part of a project of CAD21 at Tokyo Institute of Technology.

References

[1] M. Azuma, M. Saito, and A. Takahashi. A clock-tree routing algorithm for low power using feasible range of clock schedule. SLDM 2000-SLDM-97 (2000-79), Information Processing Society of Japan, 2000. in Japanese.

[2] K. D. Boese and A. B. Kahng. Zero-skew clock routing trees with minimum wirelength. In *Proc. IEEE 5th ASIC Conf.*, pages 1.1.1–1.1.5, 1992.

[3] T. H. Chao, Y. C. Hsu, and J. M. Ho. Zero skew clock net routing. In *Proc. 29th DAC*, pages 518–523, 1992.

[4] T. H. Chao, Y. C. Hsu, J. M. Ho, K. D. Boese, and A. B. Kahgn. Zero skew clock routing with minimum wirelength. *IEEE Trans. on Circuits and Systems*, 39(11):799–814, 1992.

[5] Y. Chen, A. B. Kahng, G. Qu, and A. Zelikovsky. The associative-skew clock routing problem. In *Proc. 1999 ICCAD*, pages 168–172, 1999.

[6] J. Cong, A. B. Kahng, C. K. Koh, and C. W. A. Tsao. Bounded-skew clock and Steiner routing under Elmore delay. In *Proc. 1995 ICCAD*, pages 66–71, 1995.

[7] J. Cong and C. K. Koh. Minimum-cost bounded-skew clock routing. In *Proc. ISCAS 95*, volume 1, pages 215–218, 1995.

[8] R. B. Deokar and S. S. Sapatnekar. A graph-theoretic approach to clock skew optimization. In *Proc. ISCAS '94*, volume 1, pages 407–410, 1994.

[9] M. Edahiro. A clustering-based optimization algorithm in zero-skew routings. In *Proc. 30th DAC*, pages 612–616, 1993.

[10] M. Edahiro and T. Yoshimura. Minimum path-length equidistant routing. In *Proc. APCCAS 92*, pages 41–46, 1992.

[11] J. P. Fishburn. Clock skew optimization. *IEEE Trans. on Computers*, 39(7):945–951, 1990.

[12] D. J. H. Huang, A. B. Kahng, and C. W. A. Tsao. On the bounded-skew routing tree problem. In *Proc. 32nd DAC*, pages 508–513, 1995.

[13] K. Inoue, W. Takahashi, A. Takahashi, and Y. Kajitani. Schedule-clock-tree routing for semi-synchronous circuits. *IEICE Transactions on Fundamentals*, E82-A(11):2431–2439, 1999.

[14] I. Kourtev and E. Friedman. Clock skew scheduling for improved reliability via quadratic programming. In *Proc. 1999 ICCAD*, pages 239–243, 1999.

[15] E. L. Lawler. *Combinatorial Optimization, Networks and Matroids*. Holt, Rinehart and Winston, New York, 1976.

[16] X. Liu, M. Papaefthymiou, and E. Friedman. Maximizing performance by retiming and clock skew scheduling. In *Proc. 36th DAC*, pages 231–236, 1999.

[17] H. Mitsubayashi, A. Takahashi, and Y. Kajitani. Cost-radius balanced spanning/Steiner trees. *IEICE Transactions on Fundamentals*, E80-A(4):689–694, 1997.

[18] A. Takahashi, K. Inoue, and Y. Kajitani. Clock-tree routing realizing a clock-schedule for semi-synchronous circuits. In *Proc. 1997 ICCAD*, pages 260–265, 1997.

[19] A. Takahashi and Y. Kajitani. Performance and reliability driven clock scheduling of sequential logic circuits. In *Proc. ASP-DAC '97*, pages 37–42, 1997.

[20] M. Toyonaga, K. Kurokawa, T. Yasui, and A. Takahashi. A practical clock tree synthesis for semi-synchronous circuits. In *Proc. ISPD '00*, 2000.

[21] J. G. Xi and W. W. M. Dai. Jitter-tolerant clock routing in two-phase synchronous systems. In *Proc. 1996 ICCAD*, pages 316–320, 1996.

[22] J. G. Xi and W. W. M. Dai. Useful-skew clock routing with gate sizing for low power design. In *Proc. 33rd DAC*, pages 383–388, 1996.

[23] T. Yoda and A. Takahashi. Clock schedule design for minimum realization cost. *IEICE Transactions on Fundamentals*, E83-A(12):to appear, 2000.