IEICE TRANS. FUNDAMENTALS, VOL.E85–A, NO.12 DECEMBER 2002

PAPER *Special Section on VLSI Design and CAD Algorithms*

A Clustering Based Fast Clock Schedule Algorithm for Light Clock-Trees***

Makoto SAITOH†*, **Masaaki AZUMA**†**, ***Nonmembers,***
and **Atsushi TAKAHASHI**†a), ***Regular Member***

SUMMARY 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. The registers in each cluster are driven by a buffer and a shorter wire length can be achieved. 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, the clock period achieved in average is about 13% shorter than that achieved by a zero-skew clock tree, and about 4% longer than the theoretical minimum. The wire length and power consumption of a clock tree according to an obtained clock schedule is comparable to these of a zero skew tree.

key words: *semi-synchronous circuit, clustering, clock-scheduling, clock 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 [8], [11], [21].

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.*

Manuscript received March 19, 2002.
Manuscript revised June 17, 2002.
Final manuscript received August 5, 2002.

†The authors are with the Graduate School of Science and Technology, Tokyo Institute of Technology, Tokyo, 152-8552 Japan.
*Presently, with Fujitsu Corporation.
**Presently, with Japan Patent Office.

***The preliminary version was presented at [19].

a) E-mail: atushi@lab.ss.titech.ac.jp

It is shown that an arbitrary clock schedule can be realized by constructing a clock tree [20]. 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 framework called zero skew [2]–[4], [9], [10], bounded skew [6], [7], [12], useful skew [23], [24], 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 [17], and to improve the circuit reliability in [14]. However, the realization of a clock schedule is not considered at all. In [15], [22], 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 located in a close area with the same clock input timing 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 several circuits. The computational time to obtain the clock schedule is about 81 seconds for the circuit with 888 registers by Athlon 1.4 [GHz]. The clock period achieved is about 13% shorter than that achieved by a zero-skew clock tree, and about 4% 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) [18] 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]:

Hold Constraints :
$s(v) - s(u) \leqq d_{\min}(u,v)$
Setup Constraints :
$s(u) - s(v) \leqq T - d_{\max}(u,v)$

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

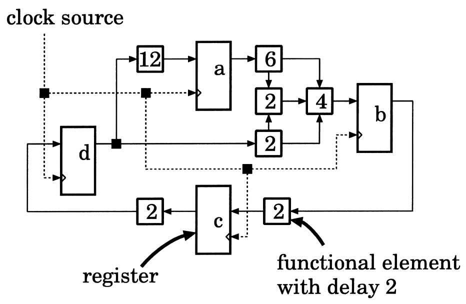


Fig. 1 Circuit.

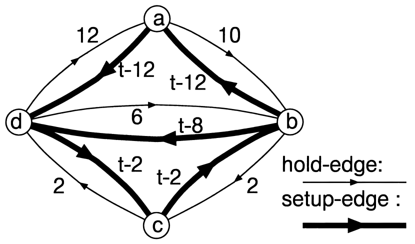


Fig. 2 Constraint graph G_t.

follows: a vertex $v \in V$ corresponds to a register, a directed edge $(u,v) \in E$ corresponds to either type of constraints; edge (u,v) corresponding to the hold (setup) constraint is called hold-edge (setup-edge), and the weight $w(u,v)$ of (u,v) is $d_{\min}(u,v)$ $(T-d_{\max}(v,u))$. The constraint graph G when the clock period is t is denoted by G_t. Similarly, weight $w(u,v)$ when the clock period is t is denoted by $w_t(u,v)$.

For example, the constraint graph G_t of the circuit shown in Fig. 1 is shown in Fig. 2.

For clock schedule s in clock period t, edge (u,v) in G_t is said to be *legal* if $s(v) - s(u) \leqq w_t(u,v)$, *illegal* otherwise. The slack of edge (u,v) for clock schedule s in clock period t is defined as

$$\Delta_{s,t}(u,v) = s(u) + w_t(u,v) - s(v).$$

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

Note that the constraints can be satisfied if and only if G contains no negative cycle [16], [21]. The smallest clock period t such that G contains no negative cycle is denoted by $T(G)$. For given the maximum and minimum propagation delays between registers, the minimum feasible clock period $T(G)$, under the assumption that the clock timing of every register can be controlled, can be determined by using the constraint graph G_0 [21]. Note that there exists a critical cycle in G_t for a feasible clock schedule if and only if the clock period t is $T(G)$.

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 $r(v)$ be a range of clock timing of register v. The set r of ranges is called *consistent* in clock period t 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 in clock period t; let $r(v)$ be

$$\left[s(v) - \frac{1}{2}\min_{(v,u)\in E}\Delta_{s,t}(v,u),\right.$$
$$\left. s(v) + \frac{1}{2}\min_{(u,v)\in E}\Delta_{s,t}(u,v)\right].$$

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 $G^C = (V^C, E^C)$ under this constraint is obtained from G by contracting vertices in each cluster into one vertex. V^C corresponds to the set of clusters and E^C corresponds to the set of hold and setup constraints between clusters. The constraint graph G^C when the clock period is t is denoted by G^C_t. 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 output resistance of the buffer. Even in deep-sub-micron technology era, the maximum routing delay is small enough if the maximum wire length is bounded. Thus, 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 [15], [22].

By assuming that the clock delay from the 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.

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 $\delta$ 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 rhombuses adjacent to rhombus $h(v)$ and that the intersection of the consistent range of $v$ and that of $c'(v)$ is maximum; move $v$ from cluster $c(v)$ 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 associated with $h(v)$ 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.
```

Fig. 3 Outline of proposed algorithm.

The outline of the proposed algorithm CBCS is shown in Fig. 3. In the outline, $c(v)$ denotes the cluster that contains register v, and $h(v)$ does the rhombus in which v is located. A rhombus is said to be adjacent to itself as well as surrounding eight rhombuses. A rhombus is adjacent to at most nine rhombuses.

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 so that a buffer can drive the registers in each cluster. By restricting the registers of a cluster to

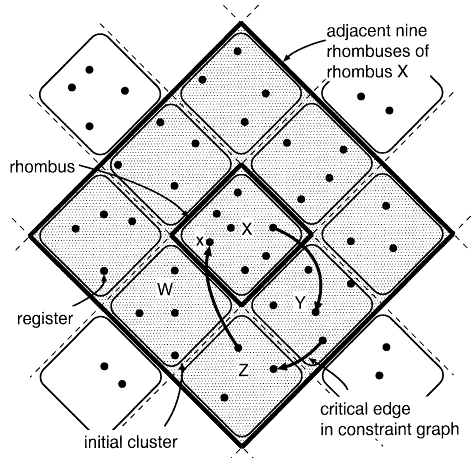

adjacent nine rhombuses of rhombus X
rhombus
x
X
W
Y
register
Z
critical edge in constraint graph
initial cluster

Fig. 4 Rhombuses and initial clustering.

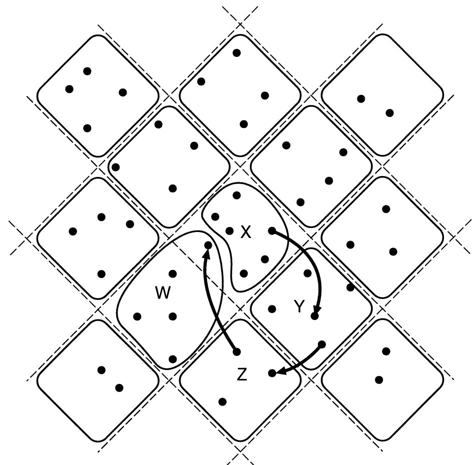


Fig. 5 Relaxing timing constraints.

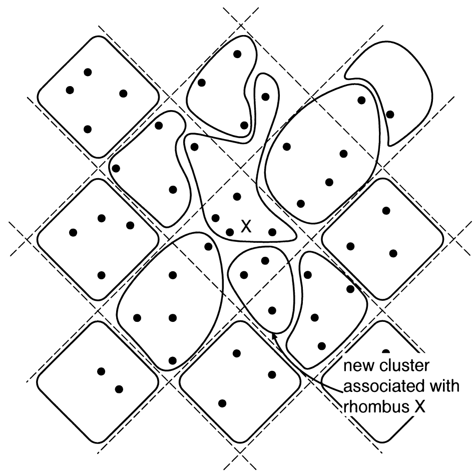


Fig. 6 Final clustering.

adjacent nine rhombuses, routing delays could be kept small enough to be ignored. Although the alignment of rhombuses would affect the final result, we select an arbitrary one, for example, boundary lines are drawn at equal intervals such that one of them is started from a corner of the area. An illustrative example is shown in Fig. 4.

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 [25]. Note that there is a critical cycle in $G^C_{T(G^C)}$ for s^C. If there is a critical cycle in $G_{T(G^C)}$ for s^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 $G_{T(G^C)}$ by deleting non-critical edges becomes a directed acyclic graph. In Fig. 4, clusters associated with X, Y, and Z form a critical cycle in $G^C_{T(G^C)}$, but there is no critical cycle in $G_{T(G^C)}$.

In Step 3, clock schedule s' is obtained by the clock schedule algorithm in [25]. Then, 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 $G_{T(G^C)}$ for s^C, except one register, are contained in S if P contains a setup-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 hold-edges, each critical cycle in $G^C_{T(G^C)}$ 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 if the consistent range of a register v and that of a cluster $c'(v)$ is neither empty nor unique. In Fig. 4, assume that the intersection of the consistent range of register x and that of the cluster associated with rhombus W is neither empty nor unique. Then, the critical cycle is broken by moving register x from cluster associated with rhombus X to cluster associated with rhombus W as shown in Fig. 5.

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 some critical cycles are broken.

In Step 6, one new cluster is created, if possible, to break a critical cycle, and return to Step 2. The number of new clusters for each rhombus is restricted to at most one to keep the number of clusters small.

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.

An illustrative example of final clustering is shown in Fig. 6. From the initial cluster associated with rhombus X, three registers are removed to relax the timing constraints, while two registers within surrounding eight rhombuses of rhombus X are included to the cluster. A new cluster associated with rhombus X which contains three registers within adjacent nine rhombuses of rhombus X is created. Each final cluster consists of registers within adjacent nine rhombuses.

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 hold-edge might be negative, and a critical cycle consisting only of hold-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 is easy to obtain.

4. Experiments

In the circuit performance optimization in semi-synchronous framework, we expect that a composition of block level local optimizations will be comparable to the whole chip level global optimization. Thus, the algorithm CBCS is applied to three circuits from industry which are parts of a chip designed in 0.25 [μm] technology. The statistics are shown in Table 1. In Table 1, "area," "#reg," "max-d," and "min-clk" are the layout area, the number of registers, the maximum signal propagation delay between registers, and the minimum clock period in semi-synchronous framework, respectively. The percentages in "max-d" are the ratios from "min-clk."

In experiments, the maximum radius of a cluster is set 300 [μm] in order to neglect routing delay in 0.25 [μm] technology. The timings of inputs and outputs of a circuit are constrained to be equal.

The clustering results are shown in Table 2. In Table 2, "Initial," "Middle," and "Final" correspond to the initial clustering, the clustering before Step 7, and the final clustering, respectively. "clk," "#clst," "#move," "time" are the achieved clock period, the number of cluster, the number of registers moved from initial clusters, and the computational time by Athlon 1.4 [GHz], respectively. The percentages in "clk" are the ratios from "min-clk" in Table 1. The clustering result of circuit C1 is shown in Fig. 7. In Fig. 7, the registers in each final cluster are plotted by the same symbol. The registers in each area of the oblique lattice form an initial cluster.

The clock period achieved in average is about 13% shorter than that achieved by a zero-skew clock tree, and about 4% longer than the theoretical minimum without considering the realization of clock schedule. The computational time for C2 is especially small since the minimum clock period is achived in early stages.

In order to confirm the quality of clustering, the clock trees are constructed by the algorithm for low power proposed in [1].

In clock tree synthesis, seven types of buffers are used. We adopt a linear delay model and a linear power model of a buffer, that is, they are a constant times the load capacitance of a buffer plus a constant. The load capacitance of a buffer is the sum of the capacitance of wire and the input capacitance of the driven buffers. Constants are fitted to 0.25 [μm] technology. We assume that there are no obstacles in routing and clock buffer insertion. The clock source is located at the lower

Table 1 Circuit statistics.

circuit	area [μm^2]	#reg	max-d [ps](%)	min-clk [ps]
C1	728 × 710	888	11569 (139.0)	8323
C2	1518 ×1510	5363	11911 (102.2)	11655
C3	1950 ×1909	7672	11808 (123.6)	9552

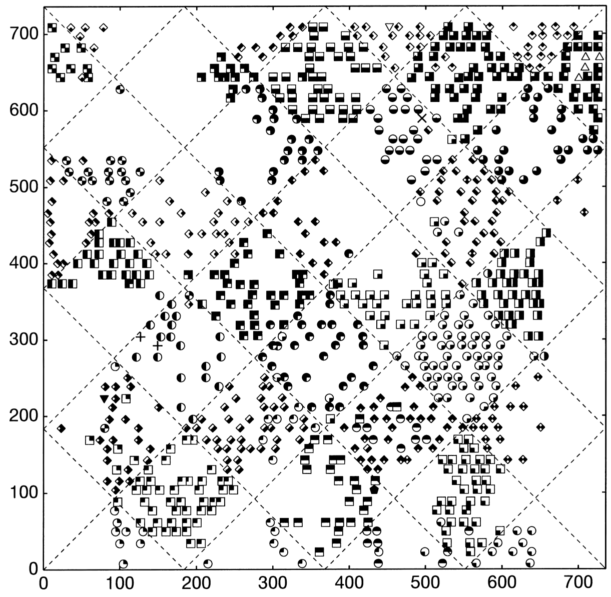


Fig. 7 Clustering result (49 clusters).

Table 2 Clustering result.

circuit	Initial			Middle		Final				
	#clst	clk [ps]	(%)	#clst	#move	#clst	#move	clk [ps]	(%)	time [s]
C1	37	10154	(122.0)	47	86	47	82	8446	(101.5)	81
C2	138	11670	(100.1)	138	2	138	2	11655	(100.0)	128
C3	220	11307	(118.4)	224	87	223	80	10533	(110.3)	1650

Table 3 Clock tree statistics.

circuit	clustering	clock period		wire length		(intra,	inter)	#buf	power	
		[ps]	(%)	[μm]	(%)	[μm],	[μm])		[μW/MHz]	(%)
C1	Final	8446	(101.5)	28427	(—)	(19319,	9108)	80	60.5	(—)
		9000	(108.1)	27995	(98.5)	(19319,	8677)	73	58.4	(96.6)
		10000	(120.1)	27249	(95.9)	(19319,	7931)	59	56.6	(93.7)
		11569	(139.0)	26733	(94.0)	(19319,	7414)	51	55.3	(91.5)
	(zero skew)	11569	(139.0)	27440	(96.5)	(19319,	8121)	78	58.5	(96.8)
	Initial	10146	(121.9)	22694	(79.8)	(15410,	7284)	58	50.4	(83.4)
		11569	(139.0)	22194	(78.1)	(15410,	6784)	47	48.8	(80.3)
	(zero skew)	11569	(139.0)	22510	(79.2)	(15410,	7100)	65	50.6	(83.8)
	Flat	8391	(100.8)	39857	(140.2)	(—,	39857)	207	76.4	(126.4)
		11569	(139.0)	29156	(102.6)	(—,	29156)	138	57.5	(95.1)
	(zero skew)	11569	(139.0)	35066	(123.4)	(—,	35066)	163	65.6	(108.5)
	Industry	11569	(139.0)	25947	(91.3)	(,	)	84	62.8	(103.9)
C2	Final	11655	(100.0)	120614	(—)	(90285,	30329)	217	302.3	(—)
	Initial	11670	(100.1)	120468	(99.9)	(90045,	30423)	219	302.5	(100.1)
	Flat	11655	(100.0)	174760	(144.9)	(—,	174760)	834	353.6	(117.0)
C3	Final	10533	(110.3)	194469	(—)	(146405,	48063)	459	451.7	(—)
	Initial	11307	(118.4)	193697	(99.6)	(140370,	53326)	374	438.6	(97.1)
	Flat	9552	(100.0)	326989	(168.1)	(—,	326989)	1559	858.1	(190.0)

left corner of the layout area.

The clock tree algorithm consists of intra-cluster routing and inter-cluster routing. In intra-cluster routing, CRBST [18] 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. The position of the register which is nearest to the center of gravity point of a cluster is used for the position of the source of the cluster. 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. We can obtain several clock tree by setting the target clock period. The larger target clock period is, the smaller the wire length and power consumption of clock tree is since the consistent ranges of clusters become large.

In Table 3, the statistics of clock trees are shown. The columns consist of the type of clustering, the target clock period, the total wire length, the number of inserted buffers, and the power consumption.

The clock trees in "Final" and "Initial" are obtained by the final clustering and the initial clustering, respectively. The clock trees in "Flat" are obtained without clustering. In the clock tree algorithm, intra-clustering routing is skipped. The zero skew clock trees are obtained by setting clock input timings of all registers to be equal. The clock tree "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" are better than others. But the minimum clock period obtained by clustering "Initial" is larger than that obtained by clustering "Final." Whenever clustering is modified in order to achieve shorter clock period, the required wire length and power consumption are increased since the radius of each cluster becomes large or the number of clusters is increased. By using the clustering "Final," the shorter clock period can be achieved. In constructing a clock trees in "Final" in "C1," the shorter we set the target 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.

The layout of the clock tree that achieves the clock period 8446 [ps] is shown in Fig. 8. In Fig. 8, the registers, the clock source and sources of clusters, intra-cluster routing, and inter-cluster routing are written in crosses, rectangles, solid lines, and dotted lines, respectively.

5. Conclusion

In this paper, we proposed a fast clock-scheduling algorithm that achieves a smaller clock period and that takes the register locations into account. In experiments, the clock period is reduced about 13% compared to the complete synchronous framework. The cost of

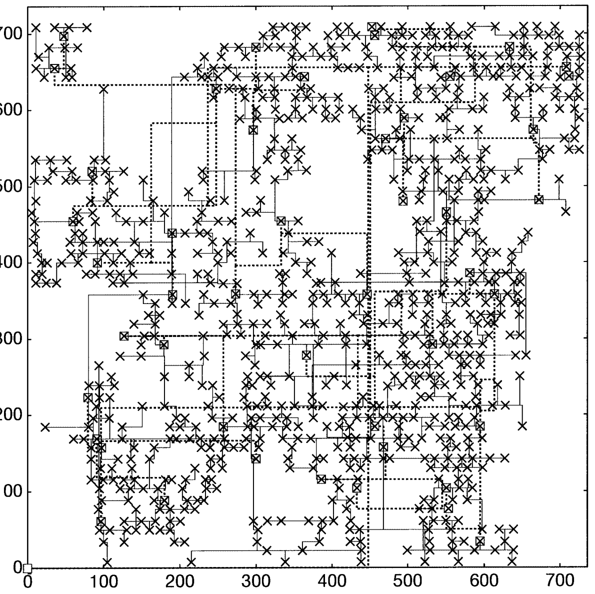


Fig. 8 Clock tree layout (8446 [ps]).

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 Professor Yoji KAJITANI, the University of Kitakyushu, Dr. Masahiko TOYONAGA, Kochi University, Mr. Keiichi KUROKAWA, and Mr. Takuya YASUI, Matsushita Electric Industrial Co., Ltd., for their helpful comments, Mr. Hajime YAMAZAKI, Tokyo Institute of Technology, for his help in experiments. This work is part of a project of CAD21 at Tokyo Institute of Technology.

References

[1] M. Azuma, M. Saitoh, and A. Takahashi, "A clock tree routing algorithm for low power using feasible range of clock schedule," IPSJ SIG Notes, vol.2000, no.79, pp.63–68, 2000.

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

[3] T.H. Chao, Y.C. Hsu, and J.M. Ho, "Zero skew clock net routing," Proc. 29th Design Automation Conference (DAC), pp.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. Circuits & Syst., vol.39, no.11, pp.799–814, 1992.

[5] Y. Chen, A.B. Kahng, G. Qu, and A. Zelikovsky, "The associative-skew clock routing problem," Proc. International Conference on Computer-Aided-Design (ICCAD), pp.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," Proc. International Conference on Computer-Aided-Design (ICCAD), pp.66–71, 1995.

[7] J. Cong and C.K. Koh, "Minimum-cost bounded-skew clock routing," Proc. International Symposium on Circuits and Systems (ISCAS), vol.1, pp.215–218, 1995.

[8] R.B. Deokar and S.S. Sapatnekar, "A graph-theoretic approach to clock skew optimization," Proc. International Symposium on Circuits and Systems (ISCAS), vol.1, pp.407–410, 1994.

[9] M. Edahiro, "A clustering-based optimization algorithm in zero-skew routings," Proc. 30th Design Automation Conference (DAC), pp.612–616, 1993.

[10] M. Edahiro and T. Yoshimura, "Minimum path-length equi-distant routing," Proc. Asia-Pacific Conference on Circuits and Systems (APCCAS), pp.41–46, 1992.

[11] J.P. Fishburn, "Clock skew optimization," IEEE Trans. Comput., vol.39, no.7, pp.945–951, 1990.

[12] D.J.H. Huang, A.B. Kahng, and C.W.A. Tsao, "On the bounded-skew routing tree problem," Proc. 32nd Design Automation Conference (DAC), pp.508–513, 1995.

[13] K. Inoue, W. Takahashi, A. Takahashi, and Y. Kajitani, "Schedule-clock-tree routing for semi-synchronous circuits," IEICE Trans. Fundamentals, vol.E82-A, no.11, pp.2431–2439, Nov. 1999.

[14] I.S. Kourtev and E.G. Friedman, "Clock skew scheduling for improved reliability via quadratic programming," Proc. International Conference on Computer-Aided-Design (ICCAD), pp.239–243, 1999.

[15] K. Kurokawa, T. Yasui, M. Toyonaga, and A. Takahashi, "A practical clock tree synthesis for semi-synchronous circuits," IEICE Trans. Fundamentals, vol.E84-A, no.11, pp.2705–2713, Nov. 2001.

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

[17] X. Liu, M.C. Papaefthymiou, and E.G. Friedman, "Maximizing performance by retiming and clock skew scheduling," Proc. 36th Design Automation Conference (DAC), pp.231–236, 1999.

[18] H. Mitsubayashi, A. Takahashi, and Y. Kajitani, "Cost-radius balanced spanning/Steiner trees," IEICE Trans. Fundamentals, vol.E80-A, no.4, pp.689–694, April 1997.

[19] M. Saitoh, M. Azuma, and A. Takahashi, "Clustering based fast clock scheduling for light clock-tree," Proc. Design Automation and Test in Europe Conference and Exhibition (DATE), pp.240–244, 2001.

[20] A. Takahashi, K. Inoue, and Y. Kajitani, "Clock-tree routing realizing a clock-schedule for semi-synchronous circuits," Proc. International Conference on Computer-Aided-Design (ICCAD), pp.260–265, 1997.

[21] A. Takahashi and Y. Kajitani, "Performance and reliability driven clock scheduling of sequential logic circuits," Proc. Asia and South Pacific Design Automation Conference (ASPDAC), pp.37–42, 1997.

[22] M. Toyonaga, K. Kurokawa, T. Yasui, and A. Takahashi, "A practical clock tree synthesis for semi-synchronous circuits," Proc. ACM International Symposium on Physical Design (ISPD), pp.159–164, 2000.

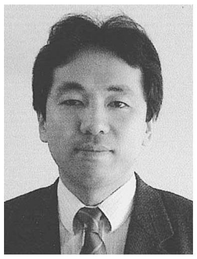
[23] J.G. Xi and W.W.M. Dai, "Jitter-tolerant clock routing in two-phase synchronous systems," Proc. International Conference on Computer-Aided-Design (ICCAD), pp.316–320, 1996.

[24] J.G. Xi and W.W.M. Dai, "Useful-skew clock routing with gate sizing for low power design," Proc. 33rd Design Automation Conference (DAC), pp.383–388, 1996.

[25] T. Yoda and A. Takahashi, "Clock schedule design for minimum realization cost," IEICE Trans. Fundamentals, vol.E83-A, no.12, pp.2552–2557, Dec. 2000.

Makoto Saitoh received his B.E., M.E. degrees in electrical and electronic engineering from Tokyo Institute of Technology, Tokyo, Japan, in 1999, and 2001, respectively. He currently works at Fujitsu Corporation. His research interests are in VLSI design automation.

Masaaki Azuma received his B.E. and M.E. degrees in electrical and electronic engineering from Tokyo Institute of Technology, Tokyo, Japan, in 1999 and 2001, respectively. He currently works for Japan Patent Office. He is an assistant examiner in the field of the transmission circuitry.



Atsushi Takahashi received his B.E., M.E., and D.E. degrees in electrical and electronic engineering from Tokyo Institute of Technology, Tokyo, Japan, in 1989, 1991, and 1996, respectively. He had been with the Tokyo Institute of Technology as a research associate from 1991 to 1997 and has been an associate professor since 1997. He is currently with Department of Communications and Integrated Systems, Graduate School of Science and Engineering. His research interests are in VLSI layout design and combinational algorithms. He is a member of the Information Processing Society of Japan and IEEE.