論文 / 著書情報
Article / Book Information

Title	Schedule-Clock-Tree Routing for Semi-Synchronous Circuits
Authors	Kazunori Inoue, Wataru Takahashi, Atsushi Takahashi, Yoji Kajitani
Citation	IEICE Trans. Fundamentals, Vol. E82-A, No. 11, pp. 2431-2439
Pub. date	1999, 11
URL	http://search.ieice.org/
Copyright	(c) 1999 Institute of Electronics, Information and Communication Engineers

PAPER *Special Section on VLSI Design and CAD Algorithms*

Schedule-Clock-Tree Routing for Semi-Synchronous Circuits**

Kazunori INOUE†, *Nonmember*, **Wataru TAKAHASHI**††*, **Atsushi TAKAHASHI**††, *and* **Yoji KAJITANI**††, *Members*

SUMMARY It is known that the clock-period can be shorter than the maximum of signal-delays between registers if the clock arrival time to each register is properly scheduled. The algorithm to design an optimal clock-schedule was given. In this paper, we propose a clock-tree routing algorithm that realizes a given clock-schedule using the Elmore-delay model. Following the deferred-merge-embedding (DME) framework, the algorithm generates a topology of the clock-tree and simultaneously determines the locations and sizes of intermediate buffers. The experimental results showed that this method constructs a clock-tree with moderate wire length for a random layout of scheduled registers. Notably, the required wire length for a gentle layout of scheduled registers was shown to be almost equal to that of zero-skew clock-trees.

key words: *clock-tree, clock-scheduling, semi-synchronous circuit, deferred-merge embedding*

1. Introduction

In layout synthesis, the distribution of the clock is critical to the performance of sequential circuits. In the *complete-synchronous* system, the clock is assumed to be distributed periodically and simultaneously to every register. The *clock-skew*, which is the maximum difference of delays to the clock pins on registers from the clock source, exerts a negative effect against speeding up a sequential circuit. Efforts have been made to eliminate it. Surveys are found in [5], [13]. In the *semi-synchronous* system, the clock is assumed to be distributed periodically to each individual register, though not necessarily to all registers simultaneously. The *clock-timing* of a register is the difference between clock-delays to the register and to a reference register. A *clock-schedule* is a set of clock-timings of registers. Given signal-delays between registers, a clock-schedule that realizes the minimum clock-period is called an *optimum clock-schedule*. An optimum clock-schedule in a semi-synchronous system is determined by a graph theoretical algorithm [22]. It is known that the minimum clock-period is usually shorter than the maximum signal-delay between registers. The minimum clock-period is obtained by a linear programming [12], or by using the decision version of the problem with a binary search strategy [8]. Similar discussions can be found on multi-phase clock-schedules [15], [19], [20].

The crucial problem in a semi-synchronous system design is the layout realization of the clock-schedule. We call a clock-tree that realizes the given clock-schedule a *schedule-clock-tree*. Herein, we propose a schedule-clock-tree routing algorithm.

Neves and Friedman [16]–[18] proposed methods by which to construct a topology of a schedule-clock-tree and to determine the specification of delay at each edge of the clock-tree. However, no specific routing algorithm to embed the topology in the Manhattan plane is given. The main target of their work is in terms of hierarchical data path design.

Zero-skew clock-tree routing is a type of schedule-clock-tree design. Tsay [24] noted that there is an algorithm for zero-skew clock-tree routing that can be extended for use with general schedule-clock-tree design by adding a fictitious delay element in each clock pin. The deferred-merge embedding (DME) algorithm for zero-skew clock-tree routing was introduced independently by Edahiro [9], [11], Chao et al. [2], [3], and by Boese and Kahng [1], [3].

Substantive related researches have been done using the DME framework [4], [6], [7], [14], [25]–[27]. For example, Xi and Dai [26], [27] proposed clock routing algorithms that modified a zero-skew clock-tree constructed by the DME algorithm; by using the allowable range of the clock-timing of each register, this method had the potential to reduce power consumption or improve the reliability of the clock-tree. Surveys concerned with the DME framework and clock synthesis can be found in [5], [13].

The DME algorithms consist of two phases; the bottom-up phase of topology generation and the top-down phase of embedding the topology in the Manhattan plane. The most successful DME algorithm, called the clustering-based DME algorithm [10], constructs a topology of the clock-tree by merging a pair of the nearest-neighbors in the bottom-up phase. The connection between two subtrees seldom makes a detour since

Manuscript received March 15, 1999.
Manuscript revised June 10, 1999.

†The author is with Software Development Center, Hitachi ULSI Systems Corporation, Kokubunji-shi, 185–0014 Japan.
††The authors are with the Department of Electrical and Electronic Engineering, Tokyo Institute of Technology, Tokyo, 152–8552 Japan.
*Presently, with C&C Media Research Laboratories, NEC Corporation.

**A preliminary version [21] was presented in ICCAD '97.

any two subtrees are usually balanced. In this manner, a small total connection length is achieved. However, in schedule-clock-tree routing, because the two subtrees may be unbalanced due to the clock-timing assigned to each register, the required connection length often differs significantly from the Manhattan distance between the roots.

The schedule-clock-tree routing algorithm proposed in this paper follows the clustering-based DME algorithm in [10]. However, the proposed algorithm selects a merging pair so that the required connection length is small, as it is in [3]. Moreover, the buffer insertion and sizing are considered in bottom-up topology generation phase, as they do in [4], [25]. In the top-down phase of embedding, each internal vertex of the clock-tree is embedded in the Manhattan plane so that the connection length from the parent vertex is minimized.

The experimental results showed that this method constructs a schedule-clock-tree with wire length that is moderate compared with that of a zero-skew clock-tree. For randomly generated pin locations and the clock-schedule, the total connection length was about 1.5 times larger than that of a zero-skew clock-tree. For pins that are located randomly but scheduled gently (that is, clock-timings of two clock-pins are near when their locations are close to each other), the wire length of a schedule-clock-tree is almost equal to that of a zero-skew clock tree.

The rest of this paper is organized as follows. In Sect. 2, we give basic definitions and an overview of the problem. The generation of the merging-segment is discussed in Sect. 3. The definition of the merging-cost of a merging-segment is given in Sect. 4. The outline of the proposed algorithm SCT-Routing is given in Sect. 5. In Sect. 6, we describe the buffer insertion and sizing procedure. The experimental results are presented in Sect. 7. Section 8 is the conclusion.

2. Preliminaries

Assume locations of the clock source pin p_0 and clock pins on registers $P = \{p_1, p_2, \ldots, p_n\}$ on the Manhattan plane, and a clock-schedule S of registers are given. We construct a clock-tree T that realizes S using the Elmore-delay model.

A clock from p_0 arrives at each clock pin p_i with some delay which is called the *clock-delay* of p_i. To describe the clock-schedule, we take an arbitrary (perhaps hypothetical) register as the reference register such that it is ticked by a clock with the *reference clock-delay*. Then pin p_i is ticked $d(p_i)$ time after the reference clock pin is ticked. $d(p_i)$ is called the *clock-timing* of p_i. We chose the reference register in order that the clock-timing $d(p_i)$ of each register would be non-negative. If the reference clock pin is ticked on time $(\ldots, -t, 0, t, 2t, \ldots)$ where t is the clock-period of the circuit concerned, then p_i is ticked on time $(\ldots, -t + d(p_i), d(p_i), t + d(p_i), 2t + d(p_i), \ldots)$.

A clock-tree T is a rooted binary tree whose root corresponds to p_0 and n leaves correspond to pins in P. For a given schedule S, T is called a *schedule-clock-tree* of S if S is realized by T. A subtree T_v is defined as the subtree of T rooted by a vertex v in T. Let $\tau(v)$ be the difference of the reference clock-delay from the clock-delay to v in T, and $\tau(v, p_i)$ be the propagation delay from v to a pin p_i in T_v. In a schedule-clock-tree, $\tau(v) = \tau(v, p_i) - d(p_i)$, for any pin p_i in T_v. See Fig. 1.

The delay model is analyzed in the following. Let r and c denote the resistance and capacitance per unit length of wire, respectively. Let v_1 and v_2 be the children of v on T, and let l_1 and l_2 be the wire length from v to v_1 and to v_2, respectively.

Let $C(v)$ be the total load capacitance of v, including wire capacitance and gate capacitance. Then $C(v)$ is calculated by

$$C(v) = \begin{cases} c_i & \text{if } v = p_i, \\ c_{buf} & \text{if a buffer is inserted into } v, \\ C^*(v) & \text{otherwise,} \end{cases}$$

where $C^*(v) = C(v_1) + C(v_2) + c(l_1 + l_2)$, and c_i and c_{buf} are the load capacitance of pin p_i and the input capacitance of the inserted buffer, respectively. Similarly, $\tau(v)$ is calculated by

$$\tau(v) = \begin{cases} -d(p_i) & \text{if } v = p_i, \\ \tau^*(v) + \tau_{buf} & \text{if a buffer is inserted into } v, \\ \tau^*(v) & \text{otherwise,} \end{cases}$$

where $\tau^*(v) = rl_1\left(\frac{cl_1}{2} + C(v_1)\right) + \tau(v_1)$ and τ_{buf} is the internal delay of the inserted buffer.

Here, similar to the case of zero-skew routing [24], the following equation derived from the π-model is assumed to be satisfied in a schedule-clock-tree:

$$\tau^*(v) = rl_2\left(\frac{cl_2}{2} + C(v_2)\right) + \tau(v_2). \quad (1)$$

A location of v is called a *delay-balance-point* of two subtrees if Eq. (1) is satisfied when the wire length connecting to the root of each subtree is equal to the Manhattan distance. In our algorithm, we list, if they exist,

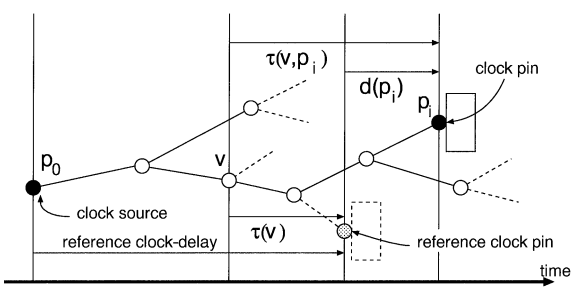

Fig. 1 Delays in a schedule-clock-tree.

the delay-balance-points of two subtrees such that the required connection length is equal to the Manhattan distance between the roots of two subtrees that are candidates for the parent vertex locations. If there is no such point, we select the candidates for the parent vertex locations from the locations of the root of either subtree such that the Manhattan distance between the roots of two subtrees is minimal. We call such a set of candidate locations of the parent vertex v a *merging-segment* of two subtrees, and denote it by $ms(v)$. For the clock pin p_i, the merging-segment $ms(p_i)$ is defined as the location of p_i.

3. Generating the Merging-Segment

Let l denote the Manhattan distance between merging-segments $ms(v_1)$ and $ms(v_2)$. By Eq. (1), assuming $l_1 + l_2 = l$, we have that

$$l_1 = \frac{\tau(v_2) - \tau(v_1) + rl(C(v_2) + cl/2)}{r(cl + C(v_1) + C(v_2))}.$$

If $0 \le l_1 \le l$ then, no detour is requested to connect two subtrees; and there are delay-balance-points of two subtrees T_{v_1} and T_{v_2} at the minimum connection length, forming $ms(v)$ of Manhattan arc with ± 1 slope. In the case that $l_1 < 0$ or $l_1 > l$, we conclude that the two subtrees are too much out of balance. If $l_1 < 0$ then, we place v on the root v_1 expecting to minimize the total connection length. The connection between v and v_2 makes a detour. The required connection length l'_2 is

$$l'_2 = \frac{\sqrt{(rC(v_2))^2 + 2rc(\tau(v_1) - \tau(v_2))} - rC(v_2)}{rc}.$$

If $l_1 > l$, then we place v on the v_2, and the length l'_1 of the connection between v and v_1 is

$$l'_1 = \frac{\sqrt{(rC(v_1))^2 + 2rc(\tau(v_2) - \tau(v_1))} - rC(v_1)}{rc}.$$

In either case, the merging-segment $ms(v)$ of the two subtrees is contained in either $ms(v_1)$ or $ms(v_2)$.

4. Merging-Cost of Merging-Segment

The *merging-cost* of two merging-segments consists of the required wire length connecting two merging-segments and the buffer insertion penalty.

The required wire length connecting two subtrees T_{v_1} and T_{v_2} tends to be large as the difference between $\tau(v_1)$ and $\tau(v_2)$ is large although v_1 is near v_2. In zero-skew routing by nearest-neighbors strategy, as in [10], the connection seldom makes a detour since the difference between $\tau(v_1)$ and $\tau(v_2)$ is relatively small. However, in schedule-clock-tree routing, we should take the detour into account. Thus, we take the required connection length rather than the distance as part of the merging-cost.

The clock-tree without any intermediate buffers is impractical since the load capacitance of the clock source is too large. Intermediate buffers are inserted into the clock-tree for the purpose of separating capacitances, to reduce clock-delays and total power dissipation, and to improve reliability against process variations. Moreover, it is possible to reduce the total connection length by buffer insertion.

We insert one buffer into a vertex if the load capacitance of the vertex exceeds the predefined value c_{lim}. Otherwise, we determine whether to insert a buffer or not according to its cost, since excessive buffer insertion may cause negative effects such as area or power dissipation by the buffers. We introduce the buffer insertion penalty $P(v)$ of subtree T_v

$$P(v) = \begin{cases} \log \dfrac{c_{lim}}{2C^*(v)} & \text{if a buffer is inserted into } v \\ & \text{and } 2C^*(v) < c_{lim}, \\ 0 & \text{otherwise.} \end{cases}$$

$P(v)$ controls the buffer insertion when the load capacitance is small. The cost depends on this penalty, but also depends on the required connection length.

For any pair of subtrees, four solutions exist with respect to the buffer insertion into the roots of those subtrees. We select one solution for the pair by using the procedure mentioned in Sect. 6.1. The insertion depends on the combination of the two subtrees. Note that the buffer is inserted into the root of a subtree or not is fixed when the subtree is merged to the other subtree.

Moreover, we assume that the inserting buffer can be selected from various sizes of buffers $\{b_1, b_2, \ldots, b_m\}$. The input capacitance of each buffer is c_{buf}. The internal delay of each buffer depends on its load capacitance. However, we assume that, for any buffer b_i and b_j $(i < j)$, the internal delays τ_{buf_i} and τ_{buf_j} satisfy $\tau_{buf_i} > \tau_{buf_j}$ for the same load capacitance. The detailed sizing procedure for two subtrees is described in Sect. 6.2.

The *merging-cost* $mc(v)$ of two subtrees T_{v_1} and T_{v_2} is defined following the solution selected by the procedure in Sect. 6,

$$mc(v) = l + \beta(P(v_1) + P(v_2)),$$

where l and β denote the required connection length and the constant coefficient, respectively. Note that either $P(v_1)$ or $P(v_2)$ is 0 since we insert buffers into both roots of the two subtrees only if the total load capacitance of each subtree exceeds c_{lim}.

5. Schedule-Clock-Tree Routing

The proposed schedule-clock-tree routing algorithm SCT-Routing follows the DME framework.

In the bottom-up phase of topology generation, a

Algorithm Topology Generation:
1. $K := \{ms(p_i)|1 \leqq i \leqq n\}$;
2. **While** $(|K| > 1)\{$
3. $G(V,E) := GRAPH(K)$;
4. **Repeat** $MID(1, |K|/k, |K|-1)$ times {
5. Find $e(v_1, v_2)$ from $G(V,E)$ such that the weight is minimum;
6. **If** $(\{ms(v_1), ms(v_2)\} \subseteqq K)$ {
7. Compete $ms(v)$ from $ms(v_1)$ and $ms(v_2)$;
8. $K := (K - \{ms(v_1), ms(v_2)\}) \cup \{ms(v)\}$;
}
9. Delete $e(v_1, v_2)$ from $G(V,E)$;
}
}

Fig. 2 The bottom-up phase of SCT-Routing.

Algorithm Topology Embedding:
1. Choose the location of the root v of the tree of merging-segments from $ms(v)$ such that the Manhattan distance from the clock source pin is minimal. Connect the clock source pin and v.
2. Local-Embedding(v);

Procedure Local-Embedding (v):
1. **If** (v has children) {
2. Choose the locations of children v_1 and v_2 from $ms(v_1)$ and $ms(v_2)$, respectively, such that the Manhattan distance from v is minimal. Connect v and v_1. Connect v and v_2;
3. Local-Embedding(v_1);
4. Local-Embedding(v_2);
}

Fig. 3 The top-down phase of SCT-Routing.

tree of merging-segments is constructed that represents possible locations (merging-segments) of vertices in a schedule-clock-tree.

Let K denote a set of merging-segments that initially consists of all the clock pin locations; that is, $K = \{ms(p_i)\}$. The algorithm iteratively finds the pair in K, that is, $ms(v_1)$ and $ms(v_2)$, such that the weight of edge $e(v_1, v_2)$ is minimal in the merging-cost graph whose vertices correspond to K and in which the weight of the edge represents the *merging-cost* of two merging-segments. The edge set of the merging-cost graph consists of the edges $e(v_i, v_j)$ such that the merging-cost of $ms(v_i)$ and $ms(v_j)$ is minimal over all $ms(v_j)$ or minimal over all $ms(v_i)$ $(i \neq j)$. A new merging-segment $ms(v)$ is computed for vertex v from the delay-balance-points of two subtrees T_{v_1} and T_{v_2}. K is updated by adding $ms(v)$ and deleting both $ms(v_1)$ and $ms(v_2)$. After $n-1$ operations, K consists of the merging-segment for the root of the topology.

The bottom-up phase is shown in Fig. 2. In the algorithm, $GRAPH(K)$ represents a merging-cost graph generation and $MID(a, b, c)$ is a function that returns the middle value of a, b, and c. The merging-cost graph is updated after several mergings for speed-up.

In the top-down phase of embedding, the exact locations of vertices are determined in reverse order of that of the bottom-up phase. First, the location of the root v of the tree of merging-segments is determined on the merging-segment $ms(v)$ so as to minimize the Manhattan distance from the clock source pin. Once the location of a vertex is determined, the locations of its children on merging-segments are easily determined so as to minimize the Manhattan distance from the parent vertex location. The top-down phase is shown in Fig. 3.

6. Buffer Insertion and Sizing

6.1 Buffer Insertion

For each pair of subtrees, four solutions exist with respect to the buffer insertion into the roots v_1 and v_2 of the two subtrees, respectively. However, we consider at most two solutions for each pair. In the following, we show the procedure that determines the solution for each pair depending on whether $C^*(v_1)$ and $C^*(v_2)$ exceed c_{lim} or not.

In the case that neither $C^*(v_1)$ nor $C^*(v_2)$ exceeds c_{lim}, the cost without buffer insertion is first calculated, which equals the required connection length. If no detour is requested, the procedure adopts the solution of no buffer being inserted into the root of either subtree. Otherwise, the connection to either v_1 or v_2 makes a detour. Without loss of generality, we assume that the connection to v_1 makes a detour. The cost when the buffer is inserted into v_1 is calculated, which is $l' + \beta P(v_1)$ where l' is the required connection length, then the procedure selects the lower cost solution.

In the case that $C^*(v_1)$ exceeds c_{lim} but $C^*(v_2)$ does not, the cost when a buffer is inserted into v_1 is first calculated, which equals the required connection length. If no detour is requested, the procedure adopts this solution. Otherwise, we calculate the cost when the buffer is inserted into both v_1 and v_2, which is $l' + \beta P(v_2)$ where l' is the required connection length. Then the procedure selects the lower cost solution. The procedure similarly selects the solution when $C^*(v_2)$ exceeds c_{lim}, but $C^*(v_1)$ does not.

In the case that both $C^*(v_1)$ and $C^*(v_2)$ exceed c_{lim}, the procedure selects the solution that buffers are inserted into both v_1 and v_2. The cost is the required wire length.

6.2 Buffer Sizing

First, we consider the case that a buffer is inserted into

v_1, but not into v_2. By Eq. (1), assuming $l_1 + l_2 = l$, we have that

$$l_1 = \frac{\tau(v_2) - (\tau^*(v_1) + \tau_{buf}) + rl(C(v_2) + cl/2)}{r(cl + c_{buf} + C(v_2))}$$

where τ_{buf} is the internal delay of the inserted buffer. If $\tau_{buf} \leq \tau(v_2) - \tau^*(v_1) + rl(C(v_2) + cl/2)$ then $l_1 \geq 0$; that is, no detour is requested to connect v and v_2. Similarly if $\tau_{buf} \geq \tau(v_2) - \tau^*(v_1) - rl(c_{buf} + cl/2)$ then if $l_1 \leq l$; that is, no detour is requested to connect v and v_1.

The internal delay of a buffer is related to the size of the buffer. The size and power dissipation of a buffer is small when the internal delay is large. To minimize the detour and the size and power of a inserted buffer, we insert a buffer such that the internal delay is maximal unless the connection from v to v_2 makes a detour. Similarly, we select a buffer to insert into v_2 when no buffer is inserted into v_1.

Next, we consider the case that buffers are inserted into both v_1 and v_2. The required connection length from v to v_1 (v_2) depends on both inserted buffers. If $C^*(v_1)$ or $C^*(v_2)$ does not exceed c_{lim}, then we first select the buffer of the root whose load capacitance exceeds c_{lim} by the above procedure, assuming no buffer is inserted into the other root. Then the buffer of the other root is selected by the above procedure. Otherwise, we test pairs of buffers for v_1 and v_2 in the following order: pick a buffer for v_1 from the maximum delay buffer to the minimum delay buffer, and for each buffer for v_1, pick a buffer for v_2 from the minimum delay buffer to the maximum delay buffer. We select the first pair for which the connection from v to v_2 makes no detour.

7. Experimental Results

The proposed schedule-clock-tree routing algorithm SCT-Routing is implemented in C++. We tested it on five different examples used in [24], though the location and load capacitance of each pin were randomly generated. The statistics of the examples are shown in Table 1. The load capacitance c_i of pin p_i ranged from 30 to 80 fF. The buffer size is chosen from 13 varieties. The maximum load capacitance of buffers c_{lim} is 1 pF. The process parameters are set to $r = 100\,\mathrm{m}\Omega$ and $c = 0.06$ fF. We use the algorithm parameters $\beta = 10$ and $k = 8$.

We show three types of experimental results for a random clock-schedule and one for a gentle clock-schedule.

7.1 Random Clock-Schedule

7.1.1 Schedule-Clock-Tree Algorithms

Our SCT-Routing and the clustering-based DME algorithm (developed for zero-skew routing) in [10], called ZS-Routing here, are applied to schedule-clock-tree routing. The results are shown in Table 2.

In Table 2, ZS and ZS+S correspond to ZS-Routing. SCT and SCT\B correspond to SCT-Routing, but no intermediate buffers are inserted in SCT\B. The clock-timing of each register is set to 0 ns in ZS, while either 0.0 ns, 0.5 ns, 1.0 ns, 1.5 ns, or 2.0 ns is randomly assigned to registers in the other cases. The total connection length, the total connection length over that in ZS, the clock-delay from the clock source pin to a reference clock pin, and the numbers of inserted intermediate buffers are shown in columns "len," "ratio," "d," and "buf," respectively.

The results show that the simple nearest-neighbors strategy, adopted in [10], is not wholly applicable for schedule-clock-tree generation. The resultant total connection length of the schedule-clock-tree is more than 15 times larger than that of the zero-skew clock-tree. A zero-skew clock-tree layout, an example of r1 by ZS-Routing, is shown in Fig. 4 for reference.

The total connection length of the SCT \ B and of the SCT is far smaller than that of ZS-Routing for the same input data. Inserted intermediate buffers reduce both the total wire lengths and clock-delays. The clock-

Table 1 Statistics of the tested examples.

data	# of pins	width [μm]	height [μm]
r1	267	6998	7000
r2	598	9401	9313
r3	862	9700	9850
r4	1903	12697	12698
r5	3101	14292	14522

Table 2 The results of ZS-Routing and SCT-Routing.

	ZS		ZS+S			SCT\B			SCT			
algorithm	ZS-Routing					SCT-Routing						
buffer	no intermediate buffers								with intermediate buffers			
clock-timing	0 [ns]		from 0 to 2 [ns] by 0.5 [ns] step									
data	len [μm]	d [ns]	len [μm]	ratio	d [ns]	len [μm]	ratio	d [ns]	len [μm]	ratio	d [ns]	buf
r1	149,044	3.1	2,621,783	17.59	23.9	320,847	2.15	4.9	226,643	1.52	4.3	145
r2	307,403	9.4	5,900,991	19.19	68.2	695,462	2.26	14.5	478,979	1.56	4.6	298
r3	390,316	10.1	8,915,795	22.84	127.5	796,512	2.04	18.2	583,456	1.49	4.5	404
r4	776,362	35.3	18,965,725	24.43	356.9	1,580,134	2.04	42.3	1,150,775	1.48	5.7	848
r5	1,180,594	62.1	30,302,038	25.67	553.6	2,270,389	1.92	87.1	1,648,764	1.40	6.4	1332

Table 3 The results of SCT-Routing for different clock-timings.

clock-timing	from 0 to 2 ns by 1 ps step				from 0 to 10 ns by 2.5 ns step			
data	len [μm]	ratio	d [ns]	buf	len [μm]	ratio	d [ns]	buf
r1	303,502	2.04	4.4	226	394,983	2.65	12.1	42
r2	604,219	1.97	5.0	537	732,336	2.38	12.7	93
r3	770,325	1.97	5.6	780	856,136	2.19	12.8	121
r4	1,535,543	1.98	6.4	1705	1,648,321	2.12	14.1	252
r5	2,247,764	1.90	6.9	2803	2,330,142	1.97	13.9	367

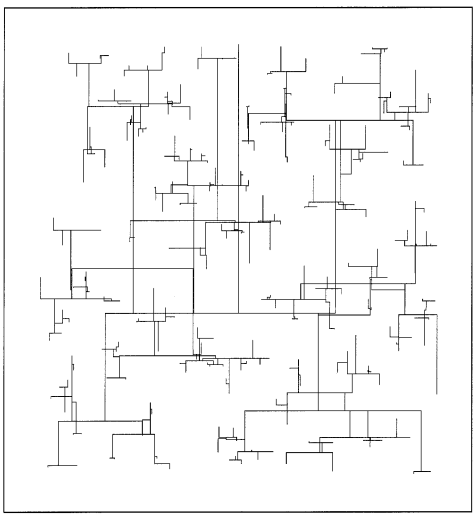

Fig. 4 A zero-skew clock-tree layout of the example r1 by ZS-Routing.

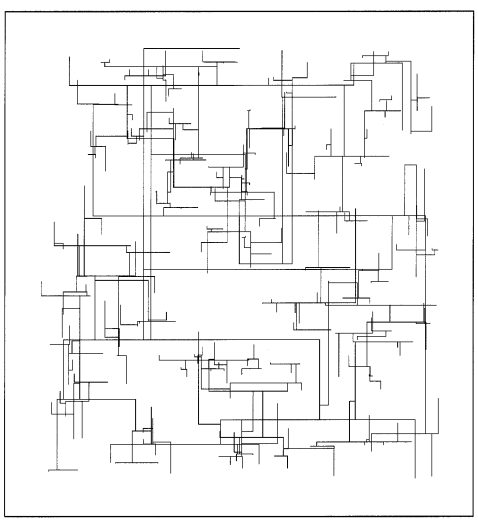

Fig. 5 A random schedule-clock-tree layout of the example r1 by SCT-Routing.

delay reductions are significant for a large clock-tree. The total connection length by SCT-Routing is about 1.5 times larger than that of the zero-skew clock-tree. A schedule-clock-tree layout, an example of r1 by SCT-Routing, is shown in Fig. 5.

Table 4 Schedule-clock-tree using zero-skew clock-trees (clock-timings are from 0 to 2 ns by 0.5 ns step).

data	len [μm]	ratio	d [ns]	buf
r1	308,396	2.07	3.3	39
r2	642,919	2.09	4.1	88
r3	821,867	2.11	4.3	107
r4	1,621,783	2.09	4.9	239
r5	2,373,154	2.01	5.5	377

7.1.2 Clock-Schedule Dependence

When the variation of clock-timings assigned to each register is large, the total connection length of SCT-Routing is large. The results shown in Table 3 reflect a clock-timing from 0 to 2 ns by 1 ps step assigned to each register. The average wire length ratio is 1.97 with respect to ZS-Routing for zero-skew routing. Although the numbers of inserted buffers are increased, the clock-delays are almost equivalent to those shown in Table 2.

The maximum difference of clock-timings also affects the total connection length and clock-delay. In Table 3, the results when clock-timing from 0 to 10 ns by 2.5 ns step is assigned to each register are shown. The wire length ratios increase compared those shown in Table 2, particularly for small size data. The clock-delays also increase although the number of inserted buffers is small.

7.1.3 Schedule-Clock-Tree Consisting of Zero-Skew Trees

If the set of pins is classified such that the clock-timing of each class is equivalent, it is possible to construct a schedule-clock-tree as the sum of the zero-skew clock-trees each designed for its class.

In Table 4, the results for the same data used in Table 2 are shown when we construct five zero-skew clock-trees for the registers each assigned to the same clock-timing by SCT-Routing. The total connection length is larger than that of the original SCT-Routing.

7.2 Gentle Clock-Schedule

A clock-schedule is said to be *gentle* if the clock-timings of two clock-pins are near when their locations are close to each other. The required wire length of a clock-tree is assumed to be small when the clock-schedule is gentle

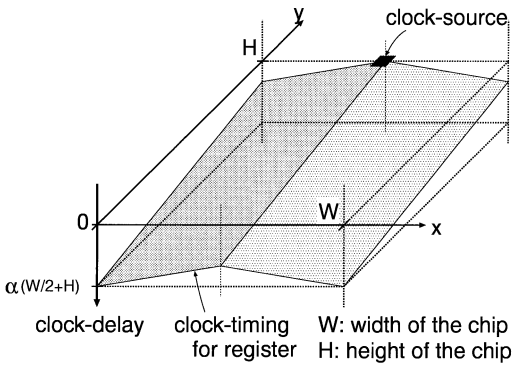

Fig. 6 Clock-delay map.

Table 5 Schedule-clock-tree using clock-routing driven layout ($\alpha = 0.5$ ns/mm).

data	len [μm]	ratio	d [ns]	buf	m [ns]
r1	164,480	1.10	6.3	79	5.0
r2	377,092	1.23	8.1	152	6.5
r3	467,100	1.20	8.8	191	7.0
r4	803,141	1.03	10.8	372	9.0
r5	1,254,727	1.06	13.0	599	10.5

rather than random.

To shorten the wire length of a clock-tree in addition to shortening the clock-period, a new design methodology that produces a gentle clock-schedule was proposed in [23]. In [23], the clock-timing of each clock-pin is determined by its location according to the clock-delay map as shown in Fig. 6. The reduction of the wire length of a clock-tree that realizes a clock-delay map using a simple delay model is assumed to be about 30% compared with that of the zero-skew clock-tree.

Here we construct a clock-tree that realizes the clock-delay map in Fig. 6 using the Elmore-delay model. The clock-timing $d(v)$ of a register v determined by the clock-delay map is proportional to the Manhattan distance from p_0 to v. That is, $d(v) = \alpha L_2(p_0, v)$ where α is a certain coefficient that transforms the length to delay, and $L_2(p_0, v)$ denotes the Manhattan distance between p_0 and v. In experiments, we set $\alpha = 0.51$ ns/mm and assign a clock-timing rounded to 0.5 ns step to each register. The results are shown in Table 5. The maximum difference of clock-timings is shown in column "m."

The results show that the wire length of a constructed clock-tree is almost equal to that of the zero-skew clock-tree and the number of inserted intermediate buffers is smaller than that of a random schedule-clock-tree. A layout result of the example r1 for a gentle clock-schedule is shown in Fig. 7.

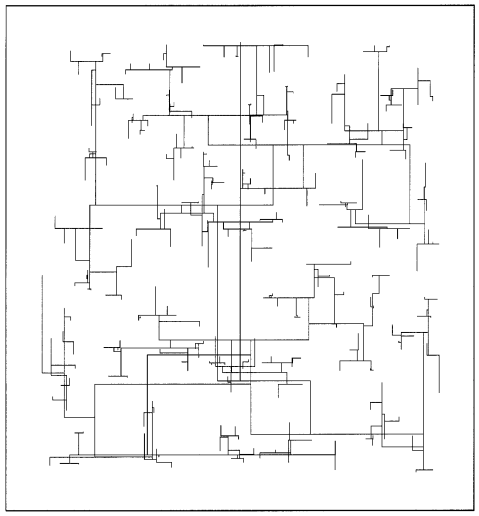

Fig. 7 A gentle schedule-clock-tree layout of the example r1 by SCT-Routing.

8. Conclusions

In this paper, we show that a schedule-clock-tree for semi-synchronous circuit design can be constructed with a wire length that is moderate compared with that of a zero-skew clock-tree for a random clock-schedule and almost equal to that for a gentle clock-schedule.

It appears that the reliability of a general schedule-clock-tree against temperature or process variation is inferior to that of a zero-skew clock-tree; a constructed clock-tree is unbalanced due to the clock-timing difference. We should analyze the sensitivity of the constructed schedule-clock-trees and improve our algorithm to increase reliability.

Our future work will also pursue the generation of feasible buffer placement in our algorithm.

The performance of a constructed schedule-clock-tree highly depends on a given clock-schedule. Taking the given clock-schedule into consideration, it is possible to adopt techniques such as those remarked in Sect. 7.1.3. It is also possible to restrict the variety of the clock-schedule, as was done in [23]. It is apparent that a clock-schedule exists for which the required wire length of the clock-tree is far smaller than that of a zero-skew clock-tree. However, at this point in our experiments, our method can only construct a clock-tree with advantages comparable to those of a zero-skew clock-tree. Further reduction of wire length may be possible if we define the merging-cost by incorporating the information from the clock-schedule or construct a clock-tree topology in a top-down manner.

We believe that high-performance circuits with a shorter wire length of the clock-tree cannot be constructed without adhering to the concept of the semi-synchronous framework. This paper includes a basic consideration of the semi-synchronous system design.

Acknowledgments

This work is part of a project of CAD21 at the Tokyo Institute of Technology, and was supported in part by a

Grant-in-Aid for Scientific Research from the Ministry of Education, Science and Culture of Japan.

References

[1] 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.
[2] T.H. Chao, Y.C. Hsu, and J.M. Ho, "Zero skew clock net routing," Proc. 29th DAC, pp.518–523, 1992.
[3] 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.
[4] Y.P. Chen and D.F. Wong, "An algorithm for zero-skew clock tree routing with buffer insertion," Proc. European Design and Test Conf., pp.66–71, 1996.
[5] J. Cong, L. He, C.K. Koh, and P.H. Madden, "Performance optimization of VLSI interconnect layout," INTEGRATION, the VLSI Journal, vol.21, pp.1–94, 1996.
[6] J. Cong, A.B. Kahng, C.K. Koh, and C.W.A. Tsao, "Bounded-skew clock and Steiner routing under Elmore delay," Proc. 1995 ICCAD, pp.66–71, 1995.
[7] J. Cong and C.K. Koh, "Minimum-cost bounded-skew clock routing," Proc. ISCAS 95, vol.1, pp.215–218, 1995.
[8] R.B. Deokar and S.S. Sapatnekar, "A graph-theoretic approach to clock skew optimization," Proc. ISCAS '94, vol.1, pp.407–410, 1994.
[9] M. Edahiro, "Minimum skew and minimum path length routing," NEC Research & Development, vol.32, no.4, pp.569–575, 1991.
[10] M.Edahiro, "A clustering-based optimization algorithm zero-skew routings," Proc. 30th DAC, pp.612–616, 1993.
[11] M. Edahiro and T. Yoshimura, "Minimum path-length equi-distant routing," Proc. APCCAS 92, pp.41–46, 1992.
[12] J.P. Fishburn, "Clock skew optimization," IEEE Trans. Comput., vol.39, no.7, pp.945–951, 1990.
[13] E.G. Friedman, ed., Clock Distribution Networks VLSI Circuits and Systems: A Selected Reprint Volume, IEEE Press, 1995.
[14] D.J.H. Huang, A.B. Kahng, and C.W.A. Tsao, "On the bounded-skew routing tree problem," Proc. 32nd DAC, pp.508–513, 1995.
[15] D.A. Joy and M.J. Ciesielski, "Placement for clock period minimization with multiple wave propagation," Proc. 28th DAC, pp.640–643, 1991.
[16] J.L. Neves and E.G. Friedman, "Topological design of clock distribution networks based on non-zero clock skew specifications," Proc. 36th Midwest Symp. on Circuits and Systems, pp.468–471, 1993.
[17] J.L. Neves and E.G. Friedman, "Circuit synthesis of clock distribution networks based on non-zero clock skew," Proc. ISCAS '94, vol.4, pp.175–178, 1994.
[18] J.L. Neves and E.G. Friedman, "Minimizing power dissipation non-zero skew-based clock distribution networks," Proc. ISCAS' 95, vol.3, pp.1577–1579, 1995.
[19] K.A. Sakallah, T.N. Mudge, and O.A. Olukotun, "Analysis and design of latch-controlled synchronous digital circuits," Proc. 27th DAC, pp.111–117, 1990.
[20] T.G. Szymanski, "Computing optimal clock schedules," Proc. 29th DAC, pp.399–404, 1992.
[21] A. Takahashi, K. Inoue, and Y. Kajitani, "Clock-tree routing realizing a clock-schedule for semi-synchronous circuits," Proc. 1997 ICCAD, pp.260–265, 1997.
[22] A. Takahashi and Y. Kajitani, "Performance and reliability driven clock scheduling of sequential logic circuits," Proc. ASP-DAC'97, pp.37–42, 1997.
[23] A. Takahashi, W. Takahashi, and Y. Kajitani, "Clock-routing driven layout methodology for semi-synchronous circuit design," Proc. TAU '97, pp.63–66, 1997.
[24] R.S. Tsay, "Exact zero skew," Proc. 1991 ICCAD, pp.336–339, 1991.
[25] A. Vittal and M. Marek-Sadowska, "Power optimal buffered clock tree design," Proc. 32nd DAC, pp.497–502, 1995.
[26] J.G. Xi and W.W.M. Dai, "Jitter-tolerant clock routing two-phase synchronous systems," Proc. 1996 ICCAD, pp.316–320, 1996.
[27] J.G. Xi and W.W.M. Dai, "Useful-skew clock routing with gate sizing for low power design," Proc. 33rd DAC, pp.383–388, 1996.

Kazunori Inoue received the B.E. and M.E. degrees in electrical and electronic engineering from the Tokyo Institute of Technology in 1995 and 1997, respectively. He is a member of the technical staff with Hitachi ULSI Systems Corp. where he is involved in the chip layout design of RISC microprocessors. The most of this work was performed while he was at the Tokyo Institute of Technology.

Wataru Takahashi received the B.E. and M.E. degrees in electrical and electronic engineering from the Tokyo Institute of Technology, Tokyo, Japan, in 1996 and 1998, respectively. He currently works at NEC Corporation. His current research interests include high level synthesis. This work was done while he was a master course student at the Tokyo Institute of Technology.

Atsushi Takahashi received his B.E., M.E., and D.E. degrees in electrical and electronic engineering from the 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 in the Department of Electrical and Electronic Engineering. His research interests include in VLSI layout design and combinational algorithms. He is a member of the IPSJ and IEEE.

Yoji Kajitani received his B.S., M.S., and D.E. degrees from the Tokyo Institute of Technology, Tokyo, Japan, all in electrical engineering, in 1964, 1966, and 1969, respectively. He has been a professor of Department of Electrical and Electronic Engineering at the Tokyo Institute of Technology since 1985, and has been a professor of Japan Advanced Institute of Science and Technology from 1992 to 1995. His current research interests are in combinatorial algorithms applied to VLSI layout design. He received the best paper awards in 1969, 1973, and 1985 all from IEICE. He was awarded IEEE Fellowship in 1992.