Proc. IEEE/ACM International Conference on Computer Aided Design 97, pp. 260–265

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

Atsushi TAKAHASHI, Kazunori INOUE†‡, and Yoji KAJITANI
Dept. of Electrical and Electronic Engineering
Tokyo Institute of Technology
Tokyo 152, Japan

†Hitachi ULSI Engineering
3-1-1 Higashi-Koigakubo, Kokubunji
Tokyo 185, Japan

Abstract

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 determines the locations and sizes of intermediate buffers simultaneously. The experimental results show that this method constructs clock-trees with moderate wire length compared with that of zero-skew clock-trees.

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. Therefore, the *clock-skew*, the maximum difference of delays to the clock pins on registers from the clock source is a negative effect against speeding up a sequential circuit. Thus efforts have been towards its elimination. Surveys are found in [13, 5].

While, in the *semi-synchronous* system, the clock is assumed to be distributed periodically to every register, but not necessarily simultaneously. A *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 [21]. It is known that the clock-period of an optimum clock-schedule is usually shorter than the maximum signal-delay between registers. The minimum clock-period is also obtained by a linear programming [12], or by using the decision version of the problem with binary search strategy [8]. Similar discussions are found in multi-phase clock-schedule [19, 15, 20].

The crucial problem in 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*. In this paper, we propose a schedule-clock-tree routing algorithm.

Neves and Friedman [16, 17, 18] proposed the methods that construct a topology of a schedule-clock-tree and 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 works is for hierarchical data path design.

Zero-skew clock-tree routing is a kind of schedule-clock-tree designs. Tsay [22] mentioned that there is an algorithm for zero-skew clock-tree routing that can be extended to general schedule-clock-tree design by adding a fictitious delay element on each clock pin. The deferred-merge embedding (DME) algorithm was introduced for zero-skew clock-tree routing independently by Edahiro [9, 11], Chao et al. [2, 3], and Boese and Kahng [1, 3]. There are many related researches using the DME framework [4, 6, 7, 14, 23, 24, 25]. Surveys concerned with the DME framework and clock synthesis are found in [13, 5].

The DME algorithms consist of two phases, the bottom-up phase of topology generation and the top-down phase of embedding the topology on the Manhattan plane. The most successful DME algorithm, called the clustering-based DME algorithm [10], constructs a topology of clock-tree by merging a pair of nearest-neighbors in bottom-up phase. The connection between two subtrees seldom makes a detour since any two subtrees are usually balanced. Thus the small total connection length is achieved. However, in schedule-clock-tree routing, the required connection length often differs significantly from the Manhattan distance between roots of two subtrees because two subtrees may be unbalanced due to the clock-timing assigned to each register.

A schedule-clock-tree routing algorithm proposed in this paper follows the clustering-based DME algorithm in [10]. However, the proposing algorithm selects a merging pair such that the required connection length is small in the way as used in [3]. Moreover, the buffer insertion and sizing are considered in bottom-up topology generation phase in the way as used in [23, 4]. 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.

*This work was supported in part by CAD21 and in part by Grant-in-Aid for Scientific Research of the Ministry of Education, Science and Culture of Japan.

‡This work was performed while the author was at TIT.

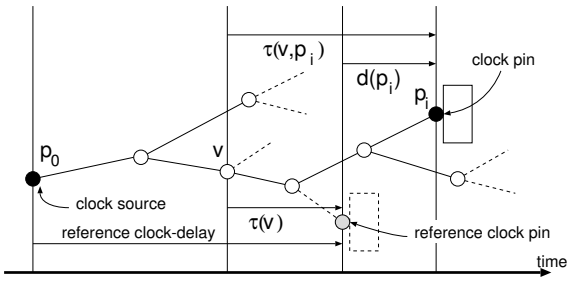

Figure 1: Delays in a schedule-clock-tree.

The experimental results showed that this method constructs a schedule-clock-tree with the moderate wire length compared with that of a zero-skew clock-tree. For randomly generated pin locations and clock-schedule, the total connection length was about 1.8 times larger than that of a zero-skew clock-tree.

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

2 Preliminaries

Assume that 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 (maybe an imaginary) 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. In this paper, we choose the reference register so that the clock-timing $d(p_i)$ of each register is 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 l_1 and l_2 be the wire length from v to v_1 and v_2, respectively.

Let $C(v)$ be the total load capacitance of v including wire capacitance as well as 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 zero-skew routing [22], the following equation derived from π-model is assumed to be satisfied in a schedule-clock-tree:

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

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 exist, delay-balance-points of two subtrees such that the required connection length is equal to the Manhattan distance between the roots of two subtrees as the candidates of the parent vertex locations of two subtrees. If there is no such point, we list the candidate location of either child such that the distance between two children is minimal as the (candidate) parent vertex location. We call such a set of candidate locations of the parent vertex v a *merging-segment* of two subtrees, and denoted 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 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} with the minimum connection length which form $ms(v)$ of Manhattan arc with ± 1 slope. In 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 two subtrees consists of a single location which 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 the 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 to separate capacitances, to reduce clock-delays and total power dissipation, and to improve the 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 a 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

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

$P(v)$ 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 a pair of two subtrees, there are four situations with respect to the buffer insertion into the roots of two subtrees. We select one situation for the pair by the procedure mentioned in Section 6.1. The insertion depends on the combination of 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 buffers 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 are described in Section 6.2.

The *merging-cost* $mc(v)$ of two subtrees T_{v_1} and T_{v_2} is defined following the situation selected by the procedure in Section 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 two subtrees only if the total load capacitance of each subtree exceeds c_{lim}.

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

Figure 2: The bottom-up phase of SCT-Routing.

5 Schedule-Clock-Tree Routing

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

In the bottom-up phase of topology generation, the tree of merging-segments are constructed which represent possible locations (merging-segment) of vertices in a schedule-clock-tree.

Let K denote a set of merging-segments which 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 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 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 the reverse order 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 two subtrees, there are four situations with respect to the buffer insertion into the roots v_1

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);
}

Figure 3: The top-down phase of SCT-Routing.

and v_2 of two subtrees. However, we consider at most two situations for each pair. In the following, we show the procedure that determines the situation for each pair depending on whether $C^*(v_1)$ and $C^*(v_2)$ exceed c_{lim} or not.

In case that neither $C^*(v_1)$ nor $C^*(v_2)$ exceeds c_{lim}, the cost without buffer insertion is first calculated, which is the required connection length. If no detour is requested, the procedure adopts this situation, that is, a buffer is inserted into neither root of the 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. Then 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 situation.

In 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 is the required connection length. If no detour is requested, the procedure adopts this situation. 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 situation. The procedure similarly selects the situation when $C^*(v_2)$ exceeds c_{lim}, but $C^*(v_1)$ does not.

In case that both $C^*(v_1)$ and $C^*(v_2)$ exceed c_{lim}, the procedure selects the situation 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.

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

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 inserting 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. And then, we select the buffer of the other root by the above procedure. Otherwise, we test the inserted buffers pair into v_1 and v_2 from the maximum delay buffer to the minimum delay buffer for v_1, and from the minimum delay buffer to the maximum delay buffer for v_2. We select the first pair such that 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 test it on five different examples used in [22], though the location and load capacitance of each pin are randomly generated. The statistics of the examples are shown in Table 1. The load capacitance c_i of pin p_i is ranged from 30 to 80 fF. The buffer size is chosen from 20 varieties and the maximum load capacitance of buffers c_{lim} is 2 pF. The process parameters are set to $r = 60$ mΩ and $c = 0.04$ fF. We use the algorithm parameters $\beta = 1000$, $k = 2$.

7.1 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 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 well applicable for schedule-clock-tree generation. The resultant total

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	147,392	1.82	3,947,102	26.78	13.02	310,379	2.11	4.13	266,514	1.81	4.78	62
r2	294,797	6.07	9,034,421	30.65	51.08	636,612	2.16	7.44	543,183	1.84	6.35	129
r3	366,265	5.85	13,139,596	35.87	83.45	815,826	2.23	7.38	679,889	1.86	6.93	177
r4	727,893	21.75	28,361,307	38.96	198.30	1,514,973	2.08	21.75	1,310,030	1.80	9.17	350
r5	1,063,507	38.52	45,920,134	43.18	327.47	2,195,560	2.06	50.56	1,886,817	1.77	5.16	515

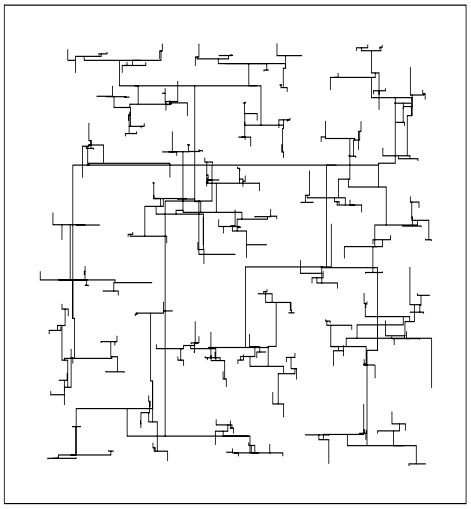

Figure 4: A zero-skew clock-tree layout of the example r1 by ZS-Routing.

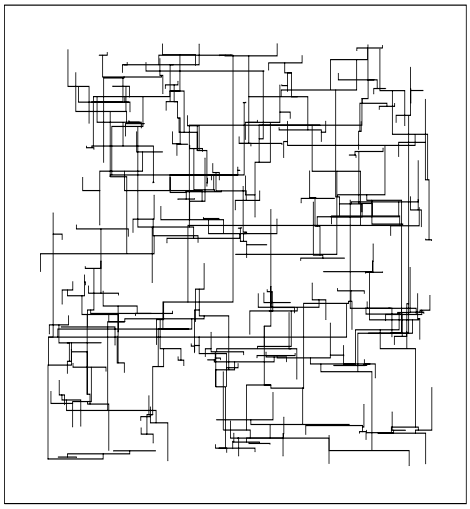

Figure 5: A schedule-clock-tree layout of the example r1 by SCT-Routing.

connection length of the schedule-clock-tree is more than 25 times larger than that of the zero-skew clock-tree. A zero-skew clock-tree layout of the example r1 by ZS-Routing is shown in Fig. 4 for a reference.

The total connection lengths of both SCT\B and SCT are 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-delay reductions are significant for large clock-tree. The total connection length by SCT-Routing is about 1.8 times larger than that of the zero-skew clock-tree. A schedule-clock-tree layout result of the example r1 by SCT-Routing is shown in Fig. 5.

7.2 Clock-Schedule Dependence

When the variation of clock-timings assigned to each register is large, the total connection length of SCT-Routing is large. In Table 3 the results when either clock-timing from 0 to 2 ns by 1 ps step is assigned to each register are shown. The average wire length ratio is 2.63 with respect to ZS-Routing for zero-skew routing. Although the numbers of inserted buffers are increased, the clock-delays are almost same compared with the results in Table 2.

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

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

If the set of pins is classified such that the clock-timing of each class is same, it is possible to construct a schedule-clock-tree as the sum of zero-skew clock-trees each designed for each 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. Although the numbers of inserted buffers are relatively small, the total connection length is larger than that of the original SCT-Routing.

8 Conclusions

In this paper, we show that we can construct a schedule-clock-tree for semi-synchronous circuit design

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	343,853	2.33	4.09	227	411,754	2.79	28.05	22
r2	793,969	2.69	5.59	500	674,988	2.29	13.24	45
r3	995,917	2.72	7.67	704	849,932	2.32	13.79	59
r4	1,967,399	2.70	6.18	1556	1,593,813	2.00	15.03	133
r5	2,914,751	2.74	6.11	2502	2,299,953	2.16	21.28	198

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	buf
r1	308,366	2.09	10
r2	674,844	2.29	34
r3	863,385	2.36	52
r4	1,673,431	2.30	101
r5	2,422,248	2.28	162

with moderate wire length compared with that of a zero-skew clock-tree.

It seems that the reliability of a general schedule-clock-tree against the temperature or process variation is inferior to that of a zero-skew clock-tree, since 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 the reliability.

The generation of feasible buffer placement in our algorithm is also the future work.

The performance of a constructed schedule-clock-tree highly depends on a given clock-schedule. It is possible to adopt techniques such as mentioned in Section 7.3 depending on the given clock-schedule. Another way is to restrict the variety of the clock-schedules. It is apparent that there exists a clock-schedule whose required wire length of clock-tree is far smaller than that of a zero-skew clock-tree. We believe that high-performance circuits with less wire length of the clock-tree, can not be constructed without following the concept of the semi-synchronous framework. This paper includes a basic consideration for the semi-synchronous system design.

References

[1] K. D. Boese and A. B. Kahng, "Zero-skew clock routing trees with minimum wirelength," in *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," in *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. on Circuits and Systems*, 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," in *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," in *Proc. 1995 ICCAD*, pp. 66–71, 1995.

[7] J. Cong and C. K. Koh, "Minimum-cost bounded-skew clock routing," in *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," in *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 in zero-skew routings," in *Proc. 30th DAC*, pp. 612–616, 1993.

[11] M. Edahiro and T. Yoshimura, "Minimum path-length equi-distant routing," in *Proc. APCCAS 92*, pp. 41–46, 1992.

[12] J. P. Fishburn, "Clock skew optimization," *IEEE Trans. on Computers*, vol. 39, no. 7, pp. 945–951, 1990.

[13] E. G. Friedman, ed., *Clock Distribution Networks in 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," in *Proc. 32nd DAC*, pp. 508–513, 1995.

[15] D. A. Joy and M. J. Ciesielski, "Placement for clock period minimization with multiple wave propagation," in *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," in *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," in *Proc. ISCAS '94*, vol. 4, pp. 175–178, 1994.

[18] J. L. Neves and E. G. Friedman, "Minimizing power dissipation in non-zero skew-based clock distribution networks," in *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," in *Proc. 27th DAC*, pp. 111–117, 1990.

[20] T. G. Szymanski, "Computing optimal clock schedules," in *Proc. 29th DAC*, pp. 399–404, 1992.

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

[22] R. S. Tsay, "Exact zero skew," in *Proc. 1991 ICCAD*, pp. 336–339, 1991.

[23] A. Vittal and M. Marek-Sadowska, "Power optimal buffered clock tree design," in *Proc. 32nd DAC*, pp. 497–502, 1995.

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

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