# 論文 / 著書情報
# Article / Book Information

| | |
|---|---|
| Title | A High-Speed and Low-Power Clock Tree Synthesis by Dynamic Clock Scheduling |
| Authors | Keiichi Kurokawa, Takuya Yasui, Yoichi Matsumura, Masahiko Toyonaga, Atsushi Takahashi |
| Citation | IEICE Trans. Fundamentals, Vol. E85-A, No. 12, pp. 2746-2755 |
| Pub. date | 2002, 12 |
| URL | http://search.ieice.org/ |
| Copyright | (c) 2002 Institute of Electronics, Information and Communication Engineers |

PAPER Special Section on VLSI Design and CAD Algorithms

# A High-Speed and Low-Power Clock Tree Synthesis by Dynamic Clock Scheduling

**Keiichi KUROKAWA**[†a)], **Takuya YASUI**[†], **Yoichi MATSUMURA**[†], ***Nonmembers***,
**Masahiko TOYONAGA**[††], ***and*** **Atsushi TAKAHASHI**[†††], ***Regular Members***

**SUMMARY** In several researches in recent years, it is shown that the circuit of a higher clock frequency can be obtained by controlling the clock-input timing of each register. However, the power consumption of the clock-tree obtained by them tends to be larger since the locations of registers are not well taken into account in clock scheduling. In this paper, we propose a novel clock tree synthesis that attains both the higher clock frequency and the lower power consumption. Our proposed algorithm determines the clock-input timings of registers step by step in constructing a clock tree structure. First, the clock period of a circuit is improved by controlling the clock-input timing of each register, and second, the clock-input timings are modified to construct a low power clock tree without deteriorating the obtained clock period. According to our experiments using several benchmark circuits, the power consumption of our clock trees attain about 9.5% smaller than previous methods.

***key words:*** *clock scheduling, clock tree synthesis, high-speed, low-power*

## 1. Introduction

According to the progress of semiconductor manufacturing process technology, the physical design issues, such as the signal delay caused by the resistance and parasitic capacitance in interconnections, and the power noise caused by the increase of the peak current, become more complicated. In recent days, many LSI designers must overcome not only the complexity of the high performance logic function, but also these physical design issues.

To make them simple, most of designers have been taking a synchronous circuit design methodology that separates a large circuit design into a clock part and a logic part. In this design methodology, the clock part is designed to deliver clock signal to all registers over the chip simultaneously, and the logic part is designed to minimize the maximum path delay between registers.

In Deep Sub-Micron (DSM) processes such as less than 0.25-micrometer rule, the signal propagation delay in LSI is strongly affected by the wire resistances, capacitances, process variations, operating conditions and so on. These factors make the clock design complicated. Thus a new design methodology is required in DSM processes.

The clock tree synthesis based on the semi-synchronous circuit design methodology that does not always require simultaneous clock delivering, could be one of solutions.

The basic idea of semi-synchronous circuit was proposed by Fishburn [1]. He gave the necessary and sufficient conditions for a synchronous circuit to work correctly with a clock period in terms of the maximum and minimum signal path delays between registers and the clock-input timings of registers. Furthermore, he showed the possibility to improve the circuit by tuning the clock-input timings of registers.

Takahashi et al. [2] interpreted the above conditions by using a constraint graph, and introduced a fast clock scheduling algorithm. Neves and Friedman [3], Kourtev and Friedman [4] had extended clock scheduling techniques in a robust circuit design under the process variation. Liu et al. [5] proposed a combined method of the retiming technique and the clock scheduling technique. They showed the clock scheduling technique could improve performance of the circuit, but, they did not complete the layout design.

Inoue et al. [7] showed any clock schedule could be realized physically by constructing a various timing clock tree (VTCT). However, the required wire length and buffers for realizing a clock schedule tend to be larger if the clock schedule is determined without considering register locations [8]. So far, many VTCT synthesis algorithms have been proposed to realize smaller clock trees, such as the bounded skew clock tree [9], [10], the useful skew clock tree [11], and the associative skew clock tree [12]. These methods modify the zero skew clock tree to reduce the size of tree or to improve the circuit performance or reliability. In [13], multi-level/multi-way clock based on a discrete clock schedule was proposed to improve the circuit performance. However, the size and power consumption of the obtained clock tree was comparable or larger than the zero skew clock tree.

In this paper, we propose a novel clock tree syn-

Manuscript received March 20, 2002.
Manuscript revised June 17, 2002.
Final manuscript received August 5, 2002.

[†]The authors are with Semiconductor Company, Matsushita Electric Industrial Co., Ltd., Nagaokakyo-shi, 617-8520 Japan.
[††]The author is with the Faculty of Science, Kochi University, Kochi-shi, 780-8520 Japan.
[†††]The author is with the Department of Communications and Integrated Systems, Tokyo Institute of Technology, Tokyo, 152-8552 Japan.
a) E-mail: kurokawa@mrg.csdd.mei.co.jp

thesis algorithm that attains both the higher clock frequency and the lower power consumption. We pay attention that there are many equivalent clock schedules in terms of the clock frequency. We select the best clock schedule that attains both the higher clock frequency and the lower power consumption from the equivalent schedules. In the clock tree we synthesis, the effect of routing delay is suppressed as small as possible. Thus, the clock-input timing of each register is determined as the sum of gate delays of clock buffers along the path from the clock source to the register.

In our proposed algorithm, first, the clock period is improved by setting the clock-input timing of each register to a multiple of the predefined unit delay. Next, the topology of a clock tree is determined with clock buffer insertion. The registers with the same clock-input timing are partitioned into clusters so that the registers in each cluster are driven by a clock buffer. The gate delay of the clock buffer is tuned to the unit delay by changing the interconnection topology from the clock buffer to registers in the cluster. If the gate delay of the clock buffer is smaller than the unit delay even after tuning, the cluster is expanded by adding registers with smaller clock-input timings unless the clock-input timings of them can be changed without violating the constraints. After the topology of the clock tree is determined, each cluster is rescheduled in order to shorten the wire length of each cluster unless the clock period is not deteriorated.

This paper is organized as follows. In Sect. 2, the condition where synchronous circuits work correctly with a given clock period and the flexibility of clock-input timings of registers are discussed. In Sect. 3, a proposed clock tree synthesis algorithm is explained. In Sect. 4, several experimental results are given to evaluate the effect of a proposed algorithm. Section 5 is the conclusion.

## 2. Preliminaries

### 2.1 Synchronous Circuits

We call the procedure of finding the set of clock-input timings of registers as *clock scheduling*. The clock-input timing $s(v)$ of register $v$ is the difference in clock arrival time between $v$ and an arbitrary chosen (perhaps hypothetical) reference register.

The relation between clock scheduling and a circuit performance is briefly explained using a part of a synchronous circuit shown in Fig. 1. Let $u$ and $v$ be the registers to which a clock with period $T$ is inputted. Let $CL$ be the combinatorial circuit between $u$ and $v$ with the signal propagation path $P_{max}(u, v)$ of the maximum signal path delay $\delta_{max}(u, v)$ and $P_{min}(u, v)$ of the minimum signal path delay $\delta_{min}(u, v)$. $Hold(v)$ and $Setup(v)$ are the hold time and the setup time of register $v$, respectively.

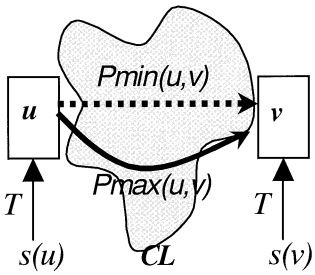



**Fig. 1** A part of a synchronous circuit.

A synchronous circuit works with period $T$ if and only if the following two inequalities are satisfied for every register pair $(u, v)$ with signal propagation path delays.

HOLD constraints

$$s(v) - s(u) \leqq \delta_{min}(u, v) - Hold(v) \tag{1}$$

SETUP constraints

$$s(u) - s(v) \leqq T - (\delta_{max}(u, v) + Setup(v)) \tag{2}$$

In order to represent the above constraints, the constraint graph $G(V, E)$ is defined as follows: a vertex $v \in V$ corresponds to a register, a directed edge $(u, v) \in E$ with weight $w_T(u, v)$ corresponds to a either type of constraints. The weight $w_T(u, v)$ is $\delta_{min}(u, v) - Hold(v)$ in case that the edge corresponds to HOLD constraint, and is $T - (\delta_{max}(v, u) + Setup(u))$ in case that the edge corresponds to SETUP constraint. That is, in the constraint graph, each $(u, v)$ correspond to the constraint that

$$s(v) - s(u) \leqq w_T(u, v) \tag{3}$$

If Eq. (3) is satisfied, for every edge in the constraint graph, the circuit correctly works with period $T$. Thus, the slack of edge $(u, v)$ is defined as

$$\Delta_T(u, v) = s(u) + w_T(u, v) - s(v) \tag{4}$$

For given the maximum and minimum propagation delays between registers, the minimum feasible clock period, under the assumption that the clock-input timing of every register can be controlled, can be determined by using the constraint graph $G$ [2]. Note that the constraints can be satisfied if and only if $G$ contains no negative cycle.

### 2.2 The Flexibility of Clock Scheduling

A clock schedule is called *feasible* if every constraint is satisfied. A feasible clock schedule can be obtained if the constraint graph contains no negative cycle. Generally speaking, a feasible clock schedule is not unique, that is, the clock-input timing of each register that satisfies the constraint is not unique even if the clock-input timings of the other registers are fixed. Let

$r(v) = [s_{min}(v), s_{max}(v)]$ be a range of clock-input timing of register $v$, called *the clock range*. A set of clock ranges $r(v)$ is called *consistent* if the feasible clock schedule is obtained when the clock-input timing of $v$ is chosen within $r(v)$. One way to get a consistent set of clock ranges is as follows: find a feasible clock schedule $s$, and then determine the clock range $r(v)$ of a register $v$ according to Eq. (5).

$$r(v) = \left[ s(v) - \frac{1}{2} \min_{(v,u)\in E} \Delta_T(v,u), s(v) + \frac{1}{2} \min_{(u,v)\in E} \Delta_T(u,v) \right] \quad (5)$$

Note that a consistent set of clock ranges is not unique in general. In this paper a smaller clock circuit is derived by using this flexibility of clock scheduling.

### 2.3 Problem Formulation

In this paper, the clock tree synthesis problem is defined as the minimization of the power consumption in the clock tree subject to the minimum feasible clock period under semi-synchronous framework in discrete clock-scheduling.

The minimum feasible clock period in discrete clock-scheduling is almost same as the minimum feasible clock period in continuous clock-scheduling [13]. Since the clock power consumption depends on the number of clock buffers and the clock wire length, they are expected to be reduced too.

The inputs of our algorithm are the position of each register to which clock is inputted, the minimum and maximum signal path delays between registers, and the position of the clock source. The outputs of our algorithm are the net list of clock tree, are the position of each clock buffer and the routing topology from each clock buffer in the clock tree. The obtained clock tree is evaluated by the clock period and the power consumption.

The layout region is rectangle. In the clock tree, only one type of clock buffer is inserted, and there is no limitation on the number of clock buffers. No obstacles are considered in the clock buffer insertion and the clock tree routing. These assumptions make problem simple. Although the deterioration is expected in the practical situation, it is small enough in our experiences.

In our algorithm, the gated clock is not considered.

## 3. The Clock Tree Synthesis by Dynamic Clock Scheduling

Our proposed algorithm is an enhancement of the algorithm proposed in [13]. Before introducing our proposed algorithm, the algorithm in [13] is briefly explained. A clock tree constructed by [13] is a multi-level multi-way tree as shown in Fig. 2. The root and internal nodes of the tree correspond to the clock buffers, and the leaves correspond to the registers. A clock buffer drives other clock buffers and/or registers.

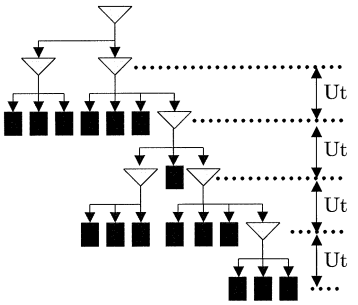



**Fig. 2** A multi-level multi-way clock tree.

The characteristic of this clock tree is that the clock-input timing of each register and each clock buffer is set as a multiple of the unit delay $Ut$. The gate delay of a clock buffer is defined as the sum of an intrinsic delay and the delay proportional to a load capacitance. The maximum load capacitance of a clock buffer is limited by the signal slew constraint. $Ut$ is set equals to the delay of the clock buffer which drives the maximum load capacitance. There are mainly two merits by restricting the clock-input timings to the discrete value. Firstly, the routing delay in the clock tree is made as small as possible so that the clock tree is robust against environmental and manufacturing variations. Secondly, one clock buffer can drive more registers and clock buffers effectively, because the number of registers and clock buffers with the same clock-input timing is larger than the previous semi-synchronous circuits with continuous clock-input timings.

In [13], the clock tree is generated in three stages as follows.

In the first stage, a clock schedule with discrete clock-input timings is obtained. The clock-input timing of each register is set to a multiple of the unit delay value $Ut$, that is, $i \times Ut$ $(1 \leqq i \leqq m)$. $m$ is defined so that the range of clock-input timings in a continuous clock schedule [16] that achieves the minimum feasible clock period is covered. When the clock-input timing of a register is $i \times Ut$, we called the clock-input timing of a register is at level $i$ since the level of the register in the clock tree is $i$.

In the second stage, the topology of the clock tree is determined from level $m$ (the bottom) to level 1 (the top). At each level, the registers with the same clock-input timing are partitioned into clusters. The size of each cluster is limited so that the gate delay of the clock buffer of the cluster is less than $Ut$, and the routing delay of interconnection is small. The clock buffers are regarded as registers in the next higher level.

In the third stage, the interconnection of each cluster is constructed so that the gate delay of the clock

buffer becomes $Ut$. The gate delay of the clock buffer of a cluster is called simply as *the cluster delay* in the following.

We introduce new steps into the algorithm in [13]. In our proposed algorithm, the clock-input timing of each register is dynamically changed in the second and third stage in order to reduce the power consumption of a clock tree. Furthermore, the clock scheduling in the first stage and the clustering algorithm in second stage are improved. The details are described in the following subsections.

### 3.1 The First Stage: The Discrete Clock Scheduling

The method of simulated annealing [14] is used to obtain a discrete clock schedule. A discrete clock schedule that satisfies hold constraints is given as an initial solution of simulated annealing. A clock schedule in which clock-input timings of all registers are equal would be a candidate of an initial solution. The clock-input timings of registers are modified one by one in simulated annealing. The *move operation* of simulated annealing is defined as the change of the clock-input timing of randomly select one register at level $i$ into level $i+1$ or $i-1$. The move is rejected if hold constraint is not satisfied. Otherwise, the move is tested whether it is accepted or not according to the cost function. The cost function is

$$Cs = \alpha \times \sum_{u \in V} \left[ \max_{(u,v) \in Es(u)} (\max(0, \Delta_{Tmf}(u,v))) \right] + \beta \times \sum_{u \in V} (s(u) - s_o)^2 \quad (6)$$

where $Es(u)$ is the set of edges incident to $u$ in the constraint graph that corresponds to SETUP constraint, $s_o$ is the target clock-input timing $i \times Ut$ $(1 \leqq i \leqq m)$, $T_{mf}$ is the minimum feasible clock period derived by the continuous scheduling method [2], and $\alpha$ and $\beta$ are constant values.

The first term in the cost function is the sum of the maximum setup violations related to each register. The second term is to get an akin tree to a zero skew clock tree as mentioned in [13]. Usually $s_o$ is set to $m/2 \times Ut$.

In the cost function in [13], the maximum setup violation and the sum of setup and hold violations are evaluated. However, the reduction of the maximum setup violation is difficult when the maximum setup violation is attained related to more than one register and the changes of clock-input timing of these registers increase the sum of setup and hold violations. Moreover, the hold constraint would not be satisfied in the final solution. If so, the circuit does not work at any clock period.

The achieved clock period $(Ts)$ of the obtained clock schedule by simulated annealing is calculated as follows.

$$T_s = T_{mf} + \max_{(u,v) \in E} (\max(0, -\Delta_{Tmf}(u,v))) \quad (7)$$

Note that the obtained clock schedule is dynamically changed in the following stages.

### 3.2 The Second Stage: Constructing a Clock Tree

In the second stage, the topology of a clock tree is determined by clustering registers from the bottom level to the top level. At each level, clustering is done by two procedures. The first procedure is initial clustering and the second procedure is expansion of clusters. Let $V^i$ be the set of registers at level $i$.

#### 3.2.1 The Initial Clustering

In *Clustering* procedure described in Fig. 3, $V^i$ is partitioned into clusters $C^i$. A start point is defined on a corner of the layout region. First, a seed of a cluster is selected among $V^i$ in order of the distance from the start point, and then a cluster is generated from the seed by adding un-clustered vertices according to *Merge* procedure described in Fig. 4. This is repeated until $V^i$ is clustered.

Let $d(u,v)$ be the Manhattan distance between registers $u$ and $v$. For a cluster $C$, the center-of gravity of $C$ and the length of the minimum spanning tree of $C$ are denoted by $Gr(C)$ and $MST(C)$, respectively. The interconnection length of $C$ is estimated by $MST(C \cup \{Gr(C)\})$. The wire capacitance $Cw(C)$ inside $C$ is estimated by using the estimated interconnection length $MST(C \cup \{Gr(C)\})$ of $C$. The cluster delay $\delta(C)$ of $C$ is estimated by $\delta_I + R \times (C_w(C) + C_p(C))$ where $\delta_I$, $R$, and $C_p$ are the intrinsic delay of a clock buffer, the output resistance of a clock buffer, and the sum of input capacitance of registers in $C$, respectively.

We assume that the routing delay inside a cluster is sufficiently small, since the wire resistance inside a

*Clustering($V^i$, $p$)*
INPUT
$V^i$: vertices(registers and buffers) at level $i$
$p$ : the start point inside layout region

OUTPUT
$C^i$: all clusters at level $i$.

1. $j = 0$
2. While $V^i \neq \phi$
   2.1. $j = j + 1$
   2.2. select $v$ in $V^i$ such that $d(p,v)$ is minimum, and let $V^i = V^i - \{v\}$.
   2.3. $C^i_j = Merge(\{v\}, V^i)$.
3. Return $C^i$.

**Fig. 3** Procedure clustering.

*Merge(C,V)*
INPUT
$C$ : the seed of clustering
$V$ : vertices(registers and buffers) for clustering.

OUTPUT
$C$ : a cluster merged vertices into

1. While $V \neq \phi$
 1.1 select $v \in V$ such that $Cc(v,C)$ is minimum.
 1.2 if the cluster delay $\delta(C \cup \{v\})$ is grater than or equal to $Ut$, then goto 2.
 1.3 else let $C = (C \cup \{v\})$, $V = V - \{v\}$
2. Return $C$.

**Fig. 4** Procedure merging vertices.

*Expand($C_i$)*
INPUT
$C^i$: all clusters at level $i$

OUTPUT
$C'^i$ : all expanded clusters at level $i$

1. Calculate $r(v) \in \{V^1 \cup V^2 \cup \ldots \cup V^{i-1}\}$ by fixing $r(u) = [s(u), s(u)]$ for $u \in \{V^i \cup V^{i+1} \cup V^{i+2} \cup \ldots \cup V^m\}$
2. Let $Vc$ be the set of vertices such that $r(v)$ contains $i \times Ut$.
3. For each $C^i_j \in C^i$ in order of $\delta(C^i_j)$
 3.1 $C'^i_j = Merge(C^i_j, Vc)$
 3.2 insert a vertex to $V^{i-1}$ that corresponds to the buffer which drives cluster $C'^i_j$
4. Return $C'^i$

**Fig. 5** Procedure expanding a cluster.

cluster is sufficiently small.

For a cluster $C$ and register $v \notin C$, let $Cc(v, C) = \eta \times d(Gr(C), v) + \xi \times \min\{d(u, v) | u \in C\}$ where the parameter of $\eta$, $\xi$ are constant values.

By using a start point, clusters are generated one by one without a crevice in the layout region. In *Merge* procedure, the diameter of the cluster and the total wire length of the cluster are kept as small as possible by adding vertex according to the cost function $Cc$.

In [13], clusters initially consist of individual registers, and they are merged each other. However, the size of final clusters is often not enough since it is difficult to find an appropriate merging partner especially in the end of procedure.

### 3.2.2 The Cluster Expansion

If the estimated cluster delay $\delta(C)$ of $C$ is less than $Ut$, registers in higher levels are tried to be added to $C$ by *Expand* procedure described in Fig. 5. The consistent clock range of each register in higher levels is computed by fixing the clock-input timing of each register in lower levels. Note that the topology of clock tree is decided from the bottom level. That is, the registers in lower levels already belong to some cluster. If the obtained consistent clock range of a register contains $i \times Ut$, the register is a candidate of an adding register. It can be added to a cluster at level $i$ without violating the Hold/Setup constraints and without contradicting to the clustering result at lower levels. From the cluster with small estimated cluster delay, the candidates are added to the cluster by *Merge* procedure.

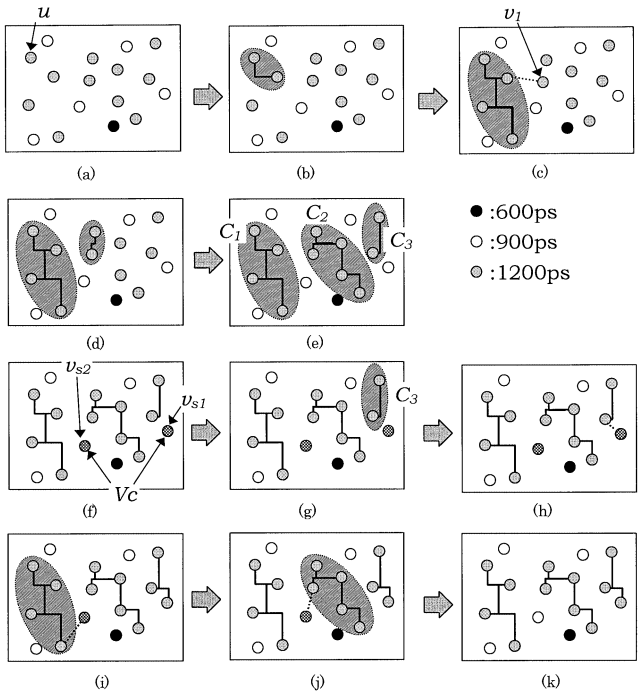



**Fig. 6** Process to construct clusters.

### 3.2.3 Illustrative Example

Figure 6 shows an example of constructing clusters. In Fig. 6 $Ut$ is defined as 300 pico-second (ps) and $m = 4$.

The registers at level 4, i.e. the bottom, of discrete clock schedule are shown in dark circle in Fig. 6(a). They are clustered into three clusters by Initial Clustering as shown in Fig. 6(a) through Fig. 6(e). The start point is given at the upper left corner. First, the vertex $u$ from which the distance to the start point is minimum among registers at level 4 is selected as the seed of the first cluster at level 4 (Fig. 6(a)). Then, three registers are added to the first cluster by *Merge* procedure (Figs. 6(b),(c)). The first cluster consists of 4 registers since the estimated cluster delay exceeds 300 [ps] if the next register $v_1$ is added to the cluster (Fig. 6(c)). The next seed is the register from which the distance to the start point is minimum among the remaining registers at level 4. The clustering result of the second and third clusters are shown in Figs. 6(d) and (e), respectively.

In Expansion, it is found that the clock-input timings of two registers $v_{s1}$, $v_{s2}$ at higher levels contain

$i \times Ut = 1200$ (Fig. 6(f)). Then cluster $C_3$ is selected as a first candidate of expansion since the estimated cluster delay of cluster $C_3$ is minimum among three clusters at level 4 (Fig. 6(g)).

By *Merge* procedure, $v_{s1}$ is added to $C_3$ (Fig. 6(h)), but $v_{s2}$ is not added to $C_3$ since the estimated cluster delay exceeds 300 [ps]. Next, the other clusters are selected, and the *Merge* procedure tries to add $v_{s2}$ to them, but $v_{s2}$ is not added to neither of them since the estimated cluster delay exceeds 300 [ps] (Figs. 6(i),(j)). The clustering result of level 4 after *Expand* procedure is shown in Fig. 6(k).

### 3.3 The Third Stage: The Routing Inside Clusters

In order to make the cluster delay the unit delay $Ut$, the wire length of a cluster by the algorithm in [13] is often larger than the shortest wire length of the cluster. However, the cluster delay does not necessarily need to be $Ut$ if the Hold/Setup constraints are satisfied. Therefore, by *Re-scheduling* procedure in Fig. 7, the clock-input timing of each register in a cluster is changed with the same amount so that increase of the wire length of the cluster is as small as possible unless the Hold/Setup constraints are violated. Note that the size of a cluster is defined so that the routing delay is negligible within the cluster.

In our algorithm, interconnections of clusters are realized by using cost-radius-balanced Steiner tree (CRBST) [15]. For the source $v_s$ and multiple sinks $V$, CRBST constructs a Steiner tree that connects them. A Steiner tree constructed by CRBST is varied by parameter $c$ of CRBST.

The CRBST starts with a single source and iteratively adds a sink with the minimum cost to a partial Steiner tree $St$ until every sink is contained in the Steiner tree. Roughly speaking, the cost of a sink $v$ is defined as

$$Cr(v) = \min_{u \in St} \left\{ c \times \frac{d(v_s, v)}{Ra} \times d_T(v_s, u) + d(u, v) \right\} \tag{8}$$

where $Ra$ is $\max_{u \in V}(d(v_s, u))$, and $d_T(v_s, u)$ is the maximum wire length from $v_s$ to $u$ in a partial Steiner tree $St$. When $c = 0$, the wire length (cost) of an obtained Steiner tree is small but the path length (radius) from the source to sinks is large. While, when $c = 1$, the wire length of an obtained Steiner tree is large but the path length from the source to sinks is small.

Let $\delta t(C)$ be the target cluster delay of cluster $C$. Initially, $\delta t(C)$ takes the value of $Ut$ for each cluster $C$. If $\delta t(C)$ is achieved for every cluster, the Hold/Setup constraints will be satisfied. Let $\delta r(C)$ be the cluster delay of $C$ when the interconnection of $C$ is realized by CRBST with $c = 0$, that is, the wire length of $C$ is expected to be the shortest. Note that $\delta r(C)$ is less than or equal to $\delta t(C)$. If the difference between $\delta t(C)$ and $\delta r(C)$ is large, then the extra wire length of $C$ to achieve $\delta t(C)$ becomes large. However, $\delta r(C)$ can be adopted if the Hold/Setup constraints are satisfied. Even if $\delta r(C)$ does not satisfy the Hold/Setup constraints, the extra wire length is reduced if $\delta t(C)$ is reduced unless Hold/Setup constraints are violated. Since the minimum cluster delay to meet the Hold/Setup constraints depends on the other cluster delays, the target cluster delay $\delta t(C)$ is changed in order to the difference between $\delta t(C)$ and $\delta r(C)$. Notice that if the cluster delay of $C$ is changed, the clock-input timing are changed for registers in lower level clusters driven by $C$ as well as for registers in $C$. Notice also that decrease of the target cluster delay of a cluster may enable the decrease of other target cluster delay.

For example, assume that the unit delay $Ut$, the target delay $\delta t(C)$ and the cluster delay $\delta r(C)$ are 300 [ps], 300 [ps], and 200 [ps], respectively. Then $\delta t(C)$ will be updated as follows. If the minimum cluster delay $\delta m(C)$ is 210 [ps], then $\delta t(C)$ is updated to 210 [ps], and if $\delta m(C)$ is 190 [ps], then $\delta t(C)$ is updated to 200 [ps]. Because $\delta t(C)$ is updated to minimum value between $\delta r(C)$ and $\delta t(C)$ without Hold and Setup violations.

If we can not change the target cluster delay $\delta t(C)$ when $\delta t(C)$ differs from $\delta r(C)$, we increase the parameter $c$ of CRBST in order to increase the wire length of the cluster to achieve the target cluster delay $\delta t(C)$. If $\delta t(C)$ is not achieved, dummy sinks are added to achieve $\delta t(C)$ by increasing the load capacitance of the clock buffer.

*Re-schedule Cluster(C)*

IINPUT
$C$: all clusters.

OUTPUT
$Wd$: the required detour wiring inside all clusters.

1. For each cluster $C_j \in C$.
   1.1 complete routing in $C_j$ by CRBST when $c = 0$.
   1.2 let $\delta_r(C_j)$ be the cluster delay of $C_j$ which is obtained by the routing result.
   1.3 let $\delta_t(C_j) = Ut$.
2. Let each cluster $C_j \in C$ unmarked.
3. Select cluster $C_j$ such that $\delta_t(C_j) - \delta_r(C_j)$ is maximum among unmarked cluster.
4. Find the minimum cluster delay $\delta_m(C_j)$ that satisfies the constraint when the cluster delays of other clusters $C_{j'}$ are $\delta_t(C_{j'})$
5. If $max(\delta_m(C_j), \delta_r(C_j))$ is less than $\delta_t(C_j)$ then let $\delta_t(C_j) = max(\delta_m(C_j), \delta_r(C_j))$ and return to 2.
6. If there are unmarked clusters, then return to 3.
7. For each cluster $C_j$, estimate the required detour $Wd(C_j)$ in order to realize $\delta_t(C_j)$.

**Fig. 7** Procedure re-scheduling clusters.

## 4. Experiments

### 4.1 The Experimental Conditions

Table 1 shows the overview of the benchmark circuits. These circuits are sub blocks of an image processing LSI that has been manufactured in our company by 0.18-micromenter process on 1.8 Volt.

The cell placement results are obtained by using a commercial layout design tool. Our clock tree synthesis defines the structure of the tree, the position of clock buffers and the routing topology from the clock buffers to registers.

The clock buffer of which the output resistance is 850 [pico-second per pico-farad] is selected. The unit delay value $Ut = 300$ [ps] is defined by a load capacitance which satisfied the signal slew constraint in 0.18-micrometer process.

### 4.2 Comparisons

Table 2 shows the minimum feasible clock period of each circuit in zero skew framework (ZS), in semi-synchronous framework that allows continuous clock-input timing [2] (Continuous), and in semi-synchronous framework by the discrete clock scheduling described in Sect. 3.1 (Discrete). The clock period of our new discrete clock scheduling is almost equivalent to that of [13], though its solution space is limited by hold constraints.

We construct four types of clock trees ZS, DC′, DC, and DCER. The clock tree ZS, which corresponds to the zero clock schedule, is constructed by *Clustering* procedure. The clock-input timing of every register is the same in the zero clock schedule. The clock tree DC′, DC, and DCER correspond to the discrete clock scheduling described in Sect. 3.1. The clock tree DC′ is constructed by *Clustering* procedure in [13]. The clock tree DC is constructed by *Clustering* procedure without *Expand* and *Re-schedule cluster* procedures. The clock tree DCER is constructed by *Clustering*, *Expand*, and *Re-schedule cluster* procedures. The number of buffers in clock trees, the wire lengths of clock trees, and the power consumption of these clock trees in one clock cycle are show in Tables 3, 4, and 5, respectively.

DC trees by our proposed *Clustering* procedure, has 7.0% fewer buffers and 13.3% shorter wire length compared with DC′ in [13]. DC has a fewer clusters than DC′, because the clusters are generated finely by the *Clustering* procedure. Therefore, DC has a fewer clock buffers and shorter clock wire length than DC′. Furthermore, the wire length of DCER decreases 34.0% by adding *Expand* and *Re-schedule cluster* procedures. As the results, the power consumption of DCER is 15.7% lower than that of DC′, and 9.5% lower than that of DC. DCER has much shorter clock wire length than DC, because the *Expand* and the *Re-schedule* make the detour in clusters shorter than DC. Moreover, since the number of registers in a cluster is increased by the *Expand* procedure, the clock buffers of DCER are reduced from DC.

**Table 1** The benchmark circuits.

| Circuits | #of Cells | # of Regs. | # of Clock Sources |
|---|---|---|---|
| C1 | 17681 | 4775 | 2 |
| C2 | 68464 | 12461 | 4 |
| C3 | 57398 | 7115 | 5 |
| C4 | 47092 | 8846 | 10 |
| C5 | 51608 | 6976 | 4 |
| C6 | 26483 | 964 | 1 |
| C7 | 17200 | 1993 | 7 |

**Table 2** Comparison of clock period (unit: ns).

| Circuits | ZS | Continuous | Discrete |
|---|---|---|---|
| C1 | 10.04 | 9.63 | 9.74 |
| | **100.0** | **95.9** | **97.0** |
| C2 | 12.62 | 10.92 | 11.28 |
| | **100.0** | **86.5** | **89.4** |
| C3 | 8.85 | 7.06 | 7.17 |
| | **100.0** | **79.8** | **81.0** |
| C4 | 8.44 | 5.58 | 5.79 |
| | **100.0** | **66.1** | **68.6** |
| C5 | 9.51 | 6.16 | 6.51 |
| | **100.0** | **64.7** | **68.5** |
| C6 | 9.48 | 8.97 | 9.27 |
| | **100.0** | **94.6** | **97.8** |
| C7 | 9.44 | 8.85 | 8.95 |
| | **100.0** | **93.8** | **94.8** |
| Ave. | **100.0** | **83.1** | **85.3** |

**Table 3** Comparison of clock buffers.

| Circuits | DC' | DC | DCER | ZS |
|---|---|---|---|---|
| C1 | 247 | 225 | 225 | 223 |
| | **110.7** | **100.9** | **100.9** | **100.0** |
| C2 | 582 | 545 | 533 | 528 |
| | **110.2** | **103.2** | **100.9** | **100.0** |
| C3 | 554 | 508 | 499 | 498 |
| | **111.2** | **102.0** | **100.2** | **100.0** |
| C4 | 759 | 703 | 690 | 666 |
| | **114.0** | **105.6** | **103.6** | **100.0** |
| C5 | 532 | 498 | 462 | 406 |
| | **131.0** | **122.6** | **113.8** | **100.0** |
| C6 | 54 | 51 | 51 | 50 |
| | **108.0** | **102.0** | **102.0** | **100.0** |
| C7 | 155 | 145 | 145 | 142 |
| | **109.2** | **102.1** | **102.1** | **100.0** |
| Ave. | **113.5** | **105.5** | **103.3** | **100.0** |

**Table 4** Comparison of the wire length of clock tree (unit: mm).

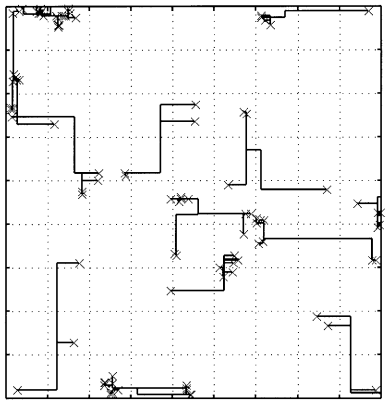
| Circuits | DC' | DC | DCER | ZS |
|---|---|---|---|---|
| C1 | 116.64 | 96.44 | 78.11 | 94.60 |
| | **123.3** | **101.9** | **82.6** | **100.0** |
| C2 | 313.11 | 279.14 | 228.91 | 248.15 |
| | **126.2** | **112.5** | **92.2** | **100.0** |
| C3 | 301.75 | 249.02 | 188.63 | 237.56 |
| | **127.0** | **104.8** | **79.4** | **100.0** |
| C4 | 456.20 | 392.00 | 294.78 | 351.78 |
| | **129.7** | **111.4** | **83.8** | **100.0** |
| C5 | 343.60 | 304.63 | 226.90 | 199.18 |
| | **172.5** | **152.9** | **113.9** | **100.0** |
| C6 | 27.03 | 24.28 | 18.10 | 23.36 |
| | **115.7** | **103.9** | **77.5** | **100.0** |
| C7 | 95.54 | 84.08 | 59.16 | 80.64 |
| | **118.5** | **104.3** | **73.4** | **100.0** |
| Ave. | **130.4** | **113.1** | **86.1** | **100.0** |

**Table 5** Comparison of clock power consumption (unit: $\mu$W/MHz).

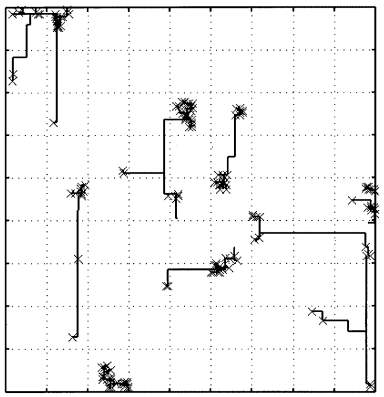
| Circuits | DC' | DC | DCER | ZS |
|---|---|---|---|---|
| C1 | 167.00 | 152.10 | 141.31 | 150.77 |
| | **110.8** | **100.9** | **93.7** | **100.0** |
| C2 | 393.48 | 368.46 | 337.25 | 356.97 |
| | **110.2** | **103.2** | **94.5** | **100.0** |
| C3 | 449.35 | 412.03 | 375.52 | 403.93 |
| | **111.2** | **102.0** | **93.0** | **100.0** |
| C4 | 614.19 | 568.77 | 510.16 | 540.19 |
| | **113.7** | **106** | **94** | **100.0** |
| C5 | 431.50 | 403.93 | 374.74 | 329.32 |
| | **131.0** | **122.7** | **113.8** | **100.0** |
| C6 | 36.50 | 34.49 | 30.83 | 33.80 |
| | **108.0** | **102.0** | **91.2** | **100.0** |
| C7 | 125.72 | 117.60 | 102.52 | 115.19 |
| | **109.1** | **102.1** | **89.0** | **100.0** |
| Ave. | **113.4** | **105.6** | **95.6** | **100.0** |

In comparison between DCER and ZS, the number of buffers increases slightly but the wire length decreases much by DCER. Thus, the power consumption of clock trees by our proposed algorithm DCER is 4.4% smaller than that of clock trees by ZS. Note that the clock period of DCER is 14.7% smaller than that of ZS.

In Fig. 8(a) and Fig. 8(b), registers at level 9 in circuit C4 by DC and DCER are shown, respectively. The registers (crosses) within each cluster are connected by lines. Note that some registers in Fig. 8(a) disappear in Fig. 8(b) since they are re-scheduled to lower levels by *Expand* procedure, while some registers appear in Fig. 8(b) since they are re-scheduled from higher levels by *Expand* procedure. The increase of the number of registers in a cluster without increasing the diameter of the cluster is seen at Fig. 8(b). Figure 9 shows the clock tree of C6 obtained by the algorithm DCER.



(a)



(b)

**Fig. 8** Clustering results. (a) Clusters obtained by the algorithm DC. (b) Clusters obtained by our algorithm DCER.

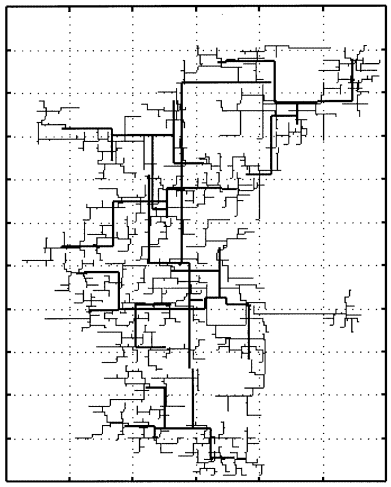


**Fig. 9** Interconnections.

## 5. Conclusion

In this paper, we proposed a novel clock tree synthesis that attains both the higher clock frequency and the lower power consumption of a clock-tree.

The clock tree in our algorithm is constructed by determining the clock-input timings of registers in constructing a clock tree structure step by step, while the previous methods were based on a firmed clock scheduling.

In the experiments using several benchmark circuits, our clock trees are 9.5% lower power consumption than the previous methods, and are about 15.7% lower power consumption than the clock trees obtained by [13], while the achieved clock periods are the same level. We compared it with the zero skew clock trees and found the clock periods are improved up to 14.7%.

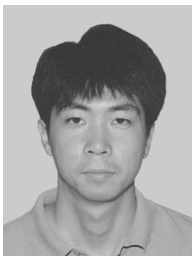
In our algorithm, we assume that the routing delay in each cluster is sufficiently small. However, since the routing delay is not zero in actual situation, we should take it into account by using a delay margin. If the large delay margin is inevitable and it affects the achievable clock period, then an enhancement of our algorithm is required to handle the routing delay within a cluster. Moreover, a simultaneous optimization of the combinatorial logic circuit and the clock tree circuit is a future work in order to further improve the performance and power of a circuit.

## Acknowledgments

We thank to Prof. Yoji Kajitani of the University of Kitakyushu for his useful suggestions, and Mr. Yoshifumi Okamoto, Mr. Seiji Yamaguchi of Matsushita Electric Industrial Co., Ltd. for their supports.

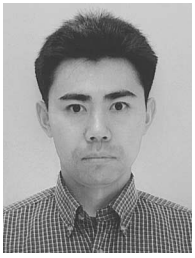
### References

[1] J.P. Fishburn, "Clock skew optimization," IEEE Trans. Comput., vol.39, pp.945–951, 1990.
[2] A. Takahashi and Y. Kajitani, "Performance and reliability driven clock scheduling of sequential logic circuits," Proc. Asia and South Pacific Design Automation Conference, pp.37–42, 1997.
[3] J.L. Neves and E.G. Friedman, "Optimal clock skew scheduling tolerant to process variations," Proc. 33rd Design Automation Conference, pp.623–628, 1996.
[4] I.S. Kourtev and E.G. Friedman, "Clock skew scheduling for improved reliability via quadratic programming," Proc. ICCAD, p.239, 1999.
[5] X. Liu, M.C. Capaefthymiou, and E.G. Friedman, "Maximizing performance by retiming and clock skew scheduling," Proc. 36th Design Automation Conference, pp.231–236, 1999.
[6] M. Edahiro, "A clustering-based optimization algorithm in zero skew routing," Proc. 30th Design Automation Conference, pp.612–616, 1993.
[7] A. Takahashi, K. Inoue, and Y. Kajitani, "Clock-tree routing realizing a clock-schedule for semi-synchronous circuits,"Proc. ICCAD, pp.260–265, 1997.
[8] 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.
[9] D.J.-H. Huang, A.B. Kahng, and C.-W.A. Tsao, "On the bounded-skew clock and steiner routing problems," Proc. 32nd Design Automation Conference, pp.508–513, 1995.
[10] A.B. Kahng and C.-W.A. Tsao, "More practical bounded-skew clock routing," Proc. 34th Design Automation Conference, pp.594–599, 1997.
[11] J.G. Xi and W.M. Dai, "Useful-skew clock routing with gate sizing for low power design," Proc. 33rd Design Automation Conference, pp.383–388, 1996.
[12] Y. Chen, A.B. Kahng, G. Qu, and A. Zelikovsky, "The associative-skew clock routing problem," Proc. ICCAD, pp.168–172, 1999.
[13] 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.
[14] S. Kirkpatrick, C.D. Gelatt, Jr., and M.P. Vecchi, "Optimization by simulated annealing," Science, vol.220, pp.671–680, 1983.
[15] 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.
[16] 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.



**Keiichi Kurokawa** received his B.E., M.E. degrees in mechanical engineering from Ritsumeikan University, Kyoto, Japan, in 1990 and 1992, respectively. He joined Matsushita Electric Industrial Co., Ltd., Osaka, Japan, in 1992. His research interests are high-level synthesis, physical design synthesis of VLSI circuit, and combinational optimization algorithms. He is a member of IEEE.

**Takuya Yasui** received his B.E., M.E. degrees information engineering from Hiroshima University, Hiroshima, Japan, in 1991 and 1993, respectively. He joined Matsushita Electric Industrial Co., Ltd., in 1993. His research interests include high-level synthesis, physical design synthesis of VLSI circuit for design automation and graph algorithm. He is a member of the Information Processing Society of Japan.



**Yoichi Matsumura** received the B.Eng., and M.Eng. degrees from Waseda University in 1995, and 1997, respectively, all in electrical engineering. In 1997, he joined Matsushita Electric Industrial Co., Ltd. Since then, he has been working on research and development of VLSI design automation. His research interests are clock tree optimization and low power design.

**Masahiko Toyonaga** received B.S. degree from Yamaguchi University, M.S. degree from Kobe University, D.E. degree in electronics engineering Osaka University, Japan, in 1979, 1981, and 1995, respectively. He joined Matsushita Electric Corporation, Kyoto, Japan, in 1982. He moved Semiconductor Research Center of Matsushita Electric Industrial Co. Ltd., Osaka, Japan, in 1993. He joined Semiconductor Industrial Research Institute Japan, in 2000. He has been a professor of the faculty of science in Kochi University since 2002. His research interests include logic/high-level synthesis, physical design synthesis of VLSI circuit, and stochastic optimization methods for design automation. He is a member of IEEE, and IPSJ.

**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.