

A Fast Register Scheduling Approach to the Architecture of Multiple Clocking Domains*

SHIH-HSU HUANG, CHIA-MING CHANG AND YOW-TYNG NIEH

*Department of Electronic Engineering
Chung Yuan Christian University
Chungli, 320 Taiwan*

Although the clock skew can be exploited as a manageable resource in the design of digital systems, it is very difficult to implement a wide spectrum of dedicated clock delays. The architecture of multiple clocking domains, which restricts the clock arrival time of each register to specified clocking domains, has attracted a lot of attentions recently. However, given a target clock period and an objective function, previous multi-domain clock skew scheduling approach requires very high computational expense to obtain the corresponding optimal multi-domain clock skew schedule. In this paper, we present a general methodology, which is independent of the objective function, to speed up the process of multi-domain clock skew scheduling. Our contribution includes the following two aspects. First, we propose an approach to deriving the ASAP (as-soon-as-possible) schedule and the ALAP (as-late-as-possible) schedule of each register. Due to pruning the redundancies, the problem size can be considerably reduced without sacrificing the exactness (optimality) of the solution. Second, we propose a zone-based scheduling algorithm to solve a large circuit heuristically, in which a zone is part of registers extracted from the large circuit. Experimental data consistently show that the zone-based scheduling algorithm is a very good heuristic.

Keywords: logic synthesis, clock skew, scheduling, digital circuit optimization, mathematical programming

1. INTRODUCTION

It is well known that the clock skew can be exploited as a manageable resource in the design of integrated circuits. A lot of research efforts have been paid to utilize the clock skew to maximize the circuit performance [1-6]. On the other hand, with respect to a target clock period, the clock skew also can be utilized to improve the tolerance to process variation [3, 4, 7, 8] or to reduce the peak current [9]. Although several applications of the clock skew have been addressed, it is very difficult to implement a wide spectrum of dedicated clock delays in an integrated circuit. As a result, conventional clock skew scheduling algorithms [1-9] have a limitation in their practical use.

The architecture of multiple clocking domains, which restricts the clock arrival time of each register to specified clocking domains, has attracted a lot of attentions recently. Ravindran, Keuhlmann, and Sentovich [10] summarized several design methods that can

Received August 31, 2005; revised April 10, 2006; accepted July 17, 2006.

Communicated by Liang-Gee Chen.

*This work was supported in part by the National Science Council of Taiwan, R.O.C. under grant No. NSC 93-2215-E-033-004. A preliminary version, entitled "Fast Multi-Domain Clock Skew Scheduling for Peak Current Reduction," has appeared in the Proceedings of Asia and South Pacific Design Automation Conference, 2006.

be used to implement the architecture of multiple clocking domains in a reliable manner. For example, Carrig [11] used the “structured clock buffers” to implement the phase shifts between clocking domains, and Singh and Brown [12] derived the set of clocking domains from a higher frequency clocking using different tapping points of a single shift register.

Fishburn [13] proposed the first polynomial time complexity algorithm to derive a feasible multi-domain clock skew scheduling with respect to a target clock period. However, in [13], no objective function is considered. Some research efforts [10, 12] were paid to the clock period minimization problem under the architecture of multiple clocking domains. On the other hand, with respect to a target clock period, Vittal, Ha, Brewer, and Marek-Sadowska [14] presented an integer linear programming (ILP) model to optimize the objective function (*e.g.*, the minimization of peak current or the maximization of tolerance to process variation) under the architecture of multiple clocking domains.

Given a target clock period and an objective function, the ILP model presented in [14] guarantees achieving the optimality under the architecture of multiple clocking domains. However, the run time of the mathematical solver grows dramatically with the increase of variables. Therefore, for a large circuit, obtaining an optimal multi-domain clock skew scheduling is very time-consuming. The high computational expense limits the use of [14]. In this paper, we present a methodology to overcome this drawback. Compared with [14], our contribution includes the following two aspects:

- (1) We propose the ASAP (as-soon-as-possible) and ALAP (as-late-as-possible) scheduling algorithm. According to the ASAP and ALAP schedule of each register, a lot of redundant variables can be pruned. As a result, the problem size can be significantly reduced without sacrificing the exactness (optimality) of the solution.
- (2) For a large circuit, the problem complexity may be still very high even though all the redundancies are pruned. We propose a zone-based scheduling algorithm to solve a large circuit heuristically, in which a zone is part of registers extracted from the large circuit. Experimental results show that our zone-based scheduling algorithm is a very good heuristic.

It is noteworthy to mention that the proposed algorithms, including the ASAP and ALAP scheduling algorithm and zone-based scheduling algorithm, are *independent* of the objective functions. Two different objective functions are used in our experiments to demonstrate the generality and effectiveness of our methodology.

The rest of the paper is organized as follows. The background for the paper is provided in section 2. In section 3, we describe the ASAP and ALAP scheduling algorithm. In section 4, we describe the zone-based scheduling algorithm. The experimental results of applying our methodology to benchmark circuits are given in section 5, and concluding remarks are given in section 6.

2. PRELIMINARIES

In section 2.1, we describe the conventional clock skew scheduling. In section 2.2, we describe the multi-domain clock skew scheduling.

2.1 Conventional Clock Skew Scheduling

An edge-triggered circuit consists of registers and logic gates, with wires connecting them. A data path from register R_i to register R_j , denoted as $R_i \rightarrow R_j$, includes the combinational logic from register R_i to register R_j . Thus, an edge-triggered circuit can be modeled as a circuit graph $G(V, E)$, where V is the set of vertices and E is the set of directed edges. Each vertex $R_i \in V$ represents a register, and a special vertex called the *host* is introduced for the synchronization with primary inputs and primary outputs. Each directed edge from register R_i to register R_j represents a data path $R_i \rightarrow R_j$, and it is associated with a weight $(T_{PDi,j(\min)}, T_{PDi,j(\max)})$, where $T_{PDi,j(\min)}$ and $T_{PDi,j(\max)}$ are the minimum delay and the maximum delay of data path $R_i \rightarrow R_j$, respectively. Using the circuit graph shown in Fig. 1 as an example, this circuit graph has 6 registers and 9 data paths. The minimum delay and the maximum delay of data path $R_1 \rightarrow R_2$ are 2 and 2, respectively; the minimum delay and the maximum delay of data path $R_2 \rightarrow R_3$ are 1 and 4, respectively; and so on.

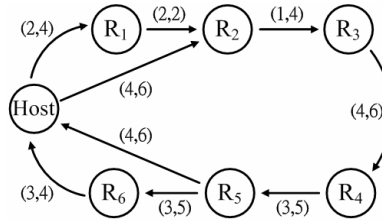


Fig. 1. Circuit graph example.

Let T_{Ci} denote the designed clock arrival time of register R_i . Note that T_{Ci} is defined as the time relative to a global time reference. In general, we use T_{host} as the global time reference, *i.e.*, $T_{host} = 0$. For a data path $R_i \rightarrow R_j$, there are two types of clocking hazards as below.

- (1) Double clocking means that the same clock pulse triggers the same data into two adjacent registers. Let's use Fig. 2 for illustration, in which the notations D_j , Q_i , D_i , C_j , and C_i denote the data input of register R_j , the data output of register R_i , the data input of register R_i , the clock of register R_j , and the clock of register R_i , respectively. If the clock signal arrives at register R_j too late, the data input of register R_j (*i.e.*, the data output of register R_i) is triggered by the same clock pulse. To prevent double clocking, the clock skew must satisfy the constraint: $T_{Cj} - T_{Ci} \leq T_{PDi,j(\min)}$.
- (2) Zero clocking means that the data reaches a register too late relative to the following clock pulse. Using Fig. 3 for illustration, if the clock signal arrives at register R_j too early, the data input of register R_j (*i.e.*, the data output of register R_i) is not triggered by the following clock pulse. To prevent zero clocking, the clock skew must satisfy the constraint: $T_{Ci} - T_{Cj} \leq P - T_{PDi,j(\max)}$, where P is the clock period.

Therefore, given a target clock period (*i.e.*, P is a constant), the two types of clocking hazards define a permissible clock skew range of a data path.

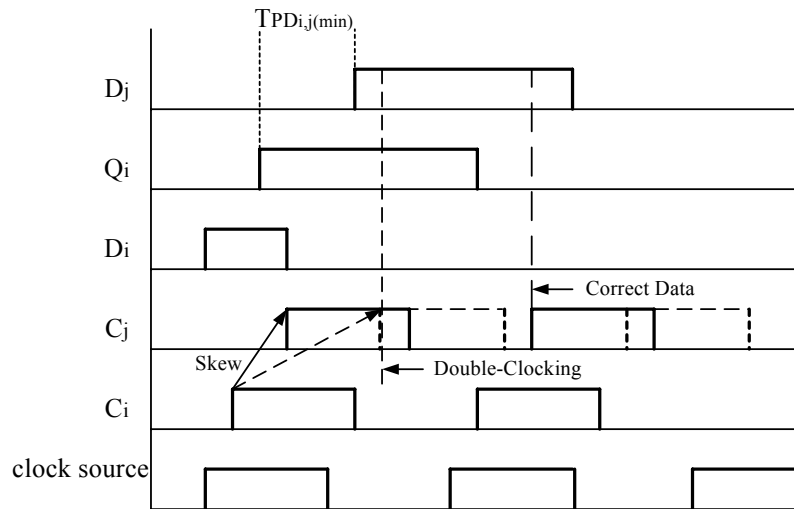


Fig. 2. Double clocking.

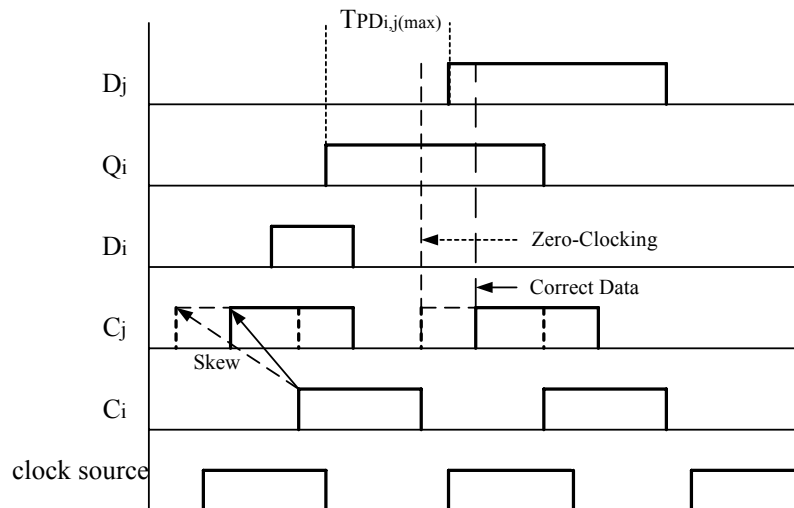


Fig. 3. Zero clocking.

Note that, in the layout implementation, the clock arrival times of registers may have some variations. Fortunately, we can model the clock arrival time of each register R_i as a range instead of a single value. Suppose that the clock arrival time of each register R_i is within the range $[T_{C_i} - \delta/2, T_{C_i} + \delta/2]$, where δ is a constant to model the uncertainty. As described in [2], the constraints of clocking hazards can be rewritten as below: $T_{C_j} - T_{C_i} \leq TPD_{ij}(\min) - \delta$ for double clocking constraint, and $T_{C_i} - T_{C_j} \leq P - (TPD_{ij}(\max) + \delta)$ for zero clocking constraint. As a result, the uncertainties of clock arrival times can be considered. In this paper, for the convenience of presentation and without loss of generality, we assume that the constant δ is 0.

2.2 Multi-Domain Clock Skew Scheduling

Multi-domain clock skew scheduling only uses n discrete clocking domains: d_1, d_2, \dots , and d_n . A clocking domain d_k , where $k = 1, 2, \dots$, and n , is a real number that corresponds to a clock arrival time. Without loss of generality, we assume that $d_1 \leq d_2 \leq \dots \leq d_n$. The clock arrival time of each register must be one of the n clocking domains. Thus, the clock arrival time of register R_i can be represented by n binary variables: $S_{i,1}, S_{i,2}, \dots, S_{i,n}$, where $S_{i,k}$ is 1 if and only if T_{Ci} is equal to d_k . As a consequence, the total number of binary variables is $r * n$, where r is the number of registers and n is the number of clocking domains. In the following, we borrow the materials from [14] to describe the mathematical formulations (ILP formulations) for multi-domain clock skew scheduling.

Each register is associated with exactly one clocking domain. Thus, for each register R_i , we have

$$\sum_{k=1}^n S_{i,k} = 1.$$

For each data path $R_i \rightarrow R_j$, the double clocking constraint can be rewritten as below:

$$\sum_{k=1}^n d_k \cdot S_{j,k} - \sum_{k=1}^n d_k \cdot S_{i,k} \leq T_{PDi,j(\min)}.$$

For each data path $R_i \rightarrow R_j$, the zero clocking constraint can be rewritten as below:

$$\sum_{k=1}^n d_k \cdot S_{i,k} - \sum_{k=1}^n d_k \cdot S_{j,k} \leq P - T_{PDi,j(\max)}.$$

Following the same assumption in [14], in this paper, we assume that the clock period P is a constant. Our objective is to obtain a feasible multi-domain clock skew schedule so that the objective function is optimized (*e.g.*, the minimization of peak current or the maximization of tolerance to process variation).

Using the circuit graph shown in Fig. 1 as an example, this circuit graph has 6 registers and 9 data paths. Suppose that we are given 3 clocking domains: $d_1 = -2$, $d_2 = 0$ and $d_3 = 2$. Therefore, we have the following 18 binary variables: $S_{1,1}, S_{1,2}, S_{1,3}, S_{2,1}, S_{2,2}, S_{2,3}, S_{3,1}, S_{3,2}, S_{3,3}, S_{4,1}, S_{4,2}, S_{4,3}, S_{5,1}, S_{5,2}, S_{5,3}, S_{6,1}, S_{6,2}$ and $S_{6,3}$. Assume that the target clock period $P = 6$. As a result, we have the following constraints:

$$\begin{aligned} S_{1,1} + S_{1,2} + S_{1,3} &= 1; \\ S_{2,1} + S_{2,2} + S_{2,3} &= 1; \\ S_{3,1} + S_{3,2} + S_{3,3} &= 1; \\ S_{4,1} + S_{4,2} + S_{4,3} &= 1; \\ S_{5,1} + S_{5,2} + S_{5,3} &= 1; \\ S_{6,1} + S_{6,2} + S_{6,3} &= 1; \\ -2 * S_{2,1} + 0 * S_{2,2} + 2 * S_{2,3} + 2 * S_{1,1} - 0 * S_{1,2} - 2 * S_{1,3} &\leq 2; \end{aligned}$$

$$\begin{aligned}
& -2 * S_{1,1} + 0 * S_{1,2} + 2 * S_{1,3} + 2 * S_{2,1} - 0 * S_{2,2} - 2 * S_{2,3} \leq 6 - 2; \\
& -2 * S_{3,1} + 0 * S_{3,2} + 2 * S_{3,3} + 2 * S_{2,1} - 0 * S_{2,2} - 2 * S_{2,3} \leq 1; \\
& -2 * S_{2,1} + 0 * S_{2,2} + 2 * S_{2,3} + 2 * S_{3,1} - 0 * S_{3,2} - 2 * S_{3,3} \leq 6 - 4; \\
& -2 * S_{4,1} + 0 * S_{4,2} + 2 * S_{4,3} + 2 * S_{3,1} - 0 * S_{3,2} - 2 * S_{3,3} \leq 4; \\
& -2 * S_{3,1} + 0 * S_{3,2} + 2 * S_{3,3} + 2 * S_{4,1} - 0 * S_{4,2} - 2 * S_{4,3} \leq 6 - 6; \\
& -2 * S_{5,1} + 0 * S_{5,2} + 2 * S_{5,3} + 2 * S_{4,1} - 0 * S_{4,2} - 2 * S_{4,3} \leq 3; \\
& -2 * S_{4,1} + 0 * S_{4,2} + 2 * S_{4,3} + 2 * S_{5,1} - 0 * S_{5,2} - 2 * S_{5,3} \leq 6 - 5; \\
& -2 * S_{6,1} + 0 * S_{6,2} + 2 * S_{6,3} + 2 * S_{5,1} - 0 * S_{5,2} - 2 * S_{5,3} \leq 3; \\
& -2 * S_{5,1} + 0 * S_{5,2} + 2 * S_{5,3} + 2 * S_{6,1} - 0 * S_{6,2} - 2 * S_{6,3} \leq 6 - 5; \\
& -2 * S_{1,1} + 0 * S_{1,2} + 2 * S_{1,3} \leq 2; \\
& 2 * S_{1,1} - 0 * S_{1,2} - 2 * S_{1,3} \leq 6 - 4; \\
& -2 * S_{2,1} + 0 * S_{2,2} + 2 * S_{2,3} \leq 4; \\
& 2 * S_{2,1} - 0 * S_{2,2} - 2 * S_{2,3} \leq 6 - 6; \\
& 2 * S_{5,1} - 0 * S_{5,2} - 2 * S_{5,3} \leq 4; \\
& -2 * S_{5,1} + 0 * S_{5,2} + 2 * S_{5,3} \leq 6 - 6; \\
& 2 * S_{6,1} - 0 * S_{6,2} - 2 * S_{6,3} \leq 3; \\
& -2 * S_{6,1} + 0 * S_{6,2} + 2 * S_{6,3} \leq 6 - 4.
\end{aligned}$$

Without loss of generality, we assume that the objective function of multi-domain clock skew scheduling is to minimize the value of $max_overlap$, which denotes the maximum number of registers scheduled in the same clocking domain.¹ Therefore, we have the following additional constraints:

$$\begin{aligned}
& S_{1,1} + S_{2,1} + S_{3,1} + S_{4,1} + S_{5,1} + S_{6,1} \leq max_overlap; \\
& S_{1,2} + S_{2,2} + S_{3,2} + S_{4,2} + S_{5,2} + S_{6,2} \leq max_overlap; \\
& S_{1,3} + S_{2,3} + S_{3,3} + S_{4,3} + S_{5,3} + S_{6,3} \leq max_overlap.
\end{aligned}$$

By applying the mathematical solver, we have the results that $max_overlap = 2$, $S_{1,1} = 0$, $S_{1,2} = 0$, $S_{1,3} = 1$, $S_{2,1} = 0$, $S_{2,2} = 1$, $S_{2,3} = 0$, $S_{3,1} = 1$, $S_{3,2} = 0$, $S_{3,3} = 0$, $S_{4,1} = 1$, $S_{4,2} = 0$, $S_{4,3} = 0$, $S_{5,1} = 0$, $S_{5,2} = 1$, $S_{5,3} = 0$, $S_{6,1} = 0$, $S_{6,2} = 0$, and $S_{6,3} = 1$. In other words, the maximum number of registers scheduled in the same clocking domain is only 2 under the multi-domain clock skew schedule in which $T_{C1} = d_3 = 2$, $T_{C2} = d_2 = 0$, $T_{C3} = d_1 = -2$, $T_{C4} = d_1 = -2$, $T_{C5} = d_2 = 0$, and $T_{C6} = d_3 = 2$.

3. ASAP AND ALAP SCHEDULING

As described in section 2.2, the number of binary variables used in the mathematical formulations of multi-domain clock skew scheduling is $r * n$, where r is the number of registers and n is the number of clocking domains. In this section, we propose the ASAP and ALAP scheduling algorithm to reduce the problem size without sacrificing the exactness (optimality) of the solution.

¹ In fact, the minimization of $max_overlap$ is relevant to the minimization of peak current caused by the simultaneous switching of registers. But a more accurate objective function for the minimization of peak current can be found in [12]. For the convenience of presentation, here, we use the minimization of $max_overlap$ as the objective function for illustration.

Due to the zero clocking constraints and the double clocking constraints, the permissible clock arrival time of each register is restricted. Therefore, some binary variables are definitely to be 0. In other words, some binary variables are redundant. Moreover, a large fraction of the constraints used in the mathematical formulations are also redundant as they are implied by some of the other constraints. If these redundant binary variables and redundant constraints can be pruned, the problem size can be significantly reduced. Thus, our approach is to find a tight bound on the clock arrival time of each register without sacrificing the exactness of the solution.

Given the clock arrival time x of a register, we define the following two functions: *floor* and *ceil*. The function *floor*(x) gives the latest clocking domain *before* the clock arrival time x . If there is no clocking domain before the clock arrival time x , the function *floor*(x) gives the clocking domain d_1 . The function *ceil*(x) gives the earliest clocking domain *after* the clock arrival time x . If there is no clocking domain after the clock arrival time x , the function *ceil*(x) gives the clocking domain d_n . The pseudo codes of the two functions are shown in Figs. 4 and 5, respectively, where n is the number of clocking domains.

```

Function floor( $x$ )
begin
  for  $i = n$  downto 1 do
    if ( $x \geq d_i$ ) then return ( $d_i$ );
  return ( $d_1$ );
end.

```

Fig. 4. Function floor.

```

Function ceil( $x$ )
begin
  for  $i = 1$  to  $n$  do
    if ( $x \leq d_i$ ) then return ( $d_i$ );
  return ( $d_n$ );
end.

```

Fig. 5. Function ceil.

Given a circuit graph G and a target clock period P , the procedure *OBTAIN_ASAP_ALAP* is to find a tight bound on the clock arrival time of each register. Here, we assume that T_{host} is definitely to be 0. Let $ASAP_i$ and $ALAP_i$ denote the earliest possible clock arrival time (*i.e.*, the earliest possible clocking domain) and the latest possible clocking domain (*i.e.*, the latest possible clocking domain) of register R_i , respectively. For the convenience of presentation, we use the notation $[ASAP_i, ALAP_i]$ to denote the allowable clocking range of register R_i . Initially, we let the allowable clocking range $[ASAP_i, ALAP_i]$ be $[d_1, d_n]$ for each register R_i . Let's use the circuit graph shown in Fig. 1 as an example. Suppose that the target clock period $P = 6$, and we are given 3 clocking domains: $d_1 = -2$, $d_2 = 0$ and $d_3 = 2$. At the beginning, we have $[ASAP_i, ALAP_i] = [d_1, d_3] = [-2, 2]$ for each register R_i , where $i = 1, 2, 3, 4, 5$, and 6.

Fig. 6 gives the pseudo code of the procedure *OBTAIN_ASAP_ALAP*. By iteratively applying the clocking constraints under the target clock period P , we gradually narrow the allowable clocking range of each register. The repeat-until-loop iteration repeats until the allowable clocking range of each register is stable. Thus, the number of repeat-until-loop iterations is $O(r * n)$. In each repeat-until-loop, there are two directions to apply the clocking constraints of data paths: *forward direction* and *backward direction*. The first for-loop applies clocking constraints in the forward direction (*i.e.*, in the direction of data path); and the second for-loop applies clocking constraints in the backward direction (*i.e.*, against the direction of data path). The details are described as below:

```

Procedure OBTAIN_ASAP_ALAP(G, P)
begin
  let [ASAPhost, ALAPhost] be [0, 0];
  for each register Ri do
    [ASAPi, ALAPi] = [d1, dn];
  repeat
    update = 0;
    for each data path Ri → Rj do
      begin
        apply clocking constraints in the forward direction to
        narrow the allowable clocking range [ASAPj, ALAPj];
        if [ASAPj, ALAPj] is narrowed then update = 1;
      end;
    for each data path Ri → Rj do
      begin
        apply clocking constraints in the backward direction to
        narrow the allowable clocking range [ASAPi, ALAPi];
        if [ASAPi, ALAPi] is narrowed then update = 1;
      end
    until (update == 0);
  end.

```

Fig. 6. Procedure *OBTAIN_ASAP_ALAP*.

- (1) The first for-loop applies clocking constraints in the forward direction. Since the clocking constraint of data path $R_i \rightarrow R_j$ is applied in the forward direction, the allowable clocking range $[ASAP_j, ALAP_j]$ of register R_j is narrowed under the assumption that the allowable clocking range $[ASAP_i, ALAP_i]$ of register R_i is fixed. As a result, we have:

$$ASAP_j = \text{maximum}(ASAP_j, \text{ceil}(ASAP_i + T_{PD_{i,j}(\text{max})} - P));$$

$$ALAP_j = \text{minimum}(ALAP_j, \text{floor}(ALAP_i + T_{PD_{i,j}(\text{min})})).$$

Note that, since T_{host} is definitely to be 0, the data paths from registers to the host are ignored when the clocking constraints are applied in the forward direction. Using the circuit graph shown in Fig. 1 as an example, the data paths $R_5 \rightarrow host$ and $R_6 \rightarrow host$ are ignored in the first for-loop.

- (2) The second for-loop applies clocking constraints in the backward direction. Since the clocking constraint of data path $R_i \rightarrow R_j$ is applied in the backward direction, the allowable clocking range $[ASAP_i, ALAP_i]$ of register R_i is narrowed under the assumption that the allowable clocking range $[ASAP_j, ALAP_j]$ of register R_j is fixed. As a result, we have:

$$ASAP_i = \text{maximum}(ASAP_i, \text{ceil}(ASAP_j - T_{PD_{i,j}(\text{min})}));$$

$$ALAP_i = \text{minimum}(ALAP_i, \text{floor}(ALAP_j + P - T_{PD_{i,j}(\text{max})})).$$

Note that, since T_{host} is definitely to be 0, the data paths from the host to registers are

ignored when the clocking constraints are applied in the backward direction. Using the circuit graph shown in Fig. 1 as an example, the data paths $host \rightarrow R_1$ and $host \rightarrow R_2$ are ignored in the second for-loop.

In fact, the sequence of data paths in each for-loop can be arbitrarily specified. However, intuitively, the more dependencies the sequence of data paths respects, the less the number of repeat-until-loop iterations there are. We have the following two deductions:

- (1) In the first for-loop, if the data paths to be tackled do not form any cycle, we can tackle them according to their topological ordering. Using the circuit graph shown in Fig. 1 as an example, the data paths are tackled in the sequence of $host \rightarrow R_1$, $host \rightarrow R_2$, $R_1 \rightarrow R_2$, $R_2 \rightarrow R_3$, $R_3 \rightarrow R_4$, $R_4 \rightarrow R_5$, and $R_5 \rightarrow R_6$.
- (2) In the second for-loop, if the data paths to be tackled do not form any cycle, we can tackle them according to the reverse of their topological ordering. Using the circuit graph shown in Fig. 1 as an example, the data paths are tackled in the sequence of $R_6 \rightarrow host$, $R_5 \rightarrow host$, $R_5 \rightarrow R_6$, $R_4 \rightarrow R_5$, $R_3 \rightarrow R_4$, $R_2 \rightarrow R_3$, and $R_1 \rightarrow R_2$.

Table 1. The snapshots of procedure *OBTAIN_ASAP_ALAP*.

Register	First repeat-until-loop		Second repeat-until-loop	
	Forward	Backward	Forward	Backward
R_1	$[-2, 2]$	$[-2, 2]$	$[-2, 2]$	$[-2, 2]$
R_2	$[0, 2]$	$[0, 2]$	$[0, 2]$	$[0, 2]$
R_3	$[-2, 2]$	$[-2, 2]$	$[-2, 2]$	$[-2, 0]$
R_4	$[-2, 2]$	$[-2, 0]$	$[-2, 0]$	$[-2, 0]$
R_5	$[-2, 2]$	$[-2, 0]$	$[-2, 0]$	$[-2, 0]$
R_6	$[-2, 2]$	$[-2, 2]$	$[-2, 2]$	$[-2, 2]$

Table 1 gives the snapshot of the procedure *OBTAIN_ASAP_ALAP* on the circuit graph shown in Fig. 1. The repeat-until-loop iteration takes two times. The column *forward* denotes the allowable clocking range after the first for-loop is completed, and the column *backward* denotes the allowable clocking range after the second for-loop is completed. After the procedure *OBTAIN_ASAP_ALAP* is completed, we have $[ASAP_1, ALAP_1] = [-2, 2]$, $[ASAP_2, ALAP_2] = [0, 2]$, $[ASAP_3, ALAP_3] = [-2, 0]$, $[ASAP_4, ALAP_4] = [-2, 0]$, $[ASAP_5, ALAP_5] = [-2, 0]$ and $[ASAP_6, ALAP_6] = [-2, 2]$. Thus, according to the ASAP and ALAP schedule of each register, we only need to have 14 binary variables: $S_{1,1}$, $S_{1,2}$, $S_{1,3}$, $S_{2,2}$, $S_{2,3}$, $S_{3,1}$, $S_{3,2}$, $S_{4,1}$, $S_{4,2}$, $S_{5,1}$, $S_{5,2}$, $S_{6,1}$, $S_{6,2}$ and $S_{6,3}$. Compared with the mathematical formulations given in section 2.2, the number of binary variables is reduced from 18 to 14.

Moreover, we can use the ASAP schedule and ALAP schedule of each register to simplify the zero clocking constraints and the double clocking constraints. For example, the zero clocking constraint of the data path $host \rightarrow R_2$ can be simplified to $0 * S_{2,2} - 2 * S_{2,3} \leq 6 - 6$, which can be easily justified as a redundant constraint. The justification of redundant constraints is in polynomial time complexity. By exploiting the ASAP sched-

ule and ALAP schedule of each register, we find that all zero clocking constraints and all the double clocking constraints in this circuit graph become redundant. As a result, we can rewrite the problem of minimizing the value of $max_overlap$ as below:

minimize $max_overlap$
subject to

$$\begin{aligned}
&S_{1,1} + S_{1,2} + S_{1,3} = 1; \\
&S_{2,2} + S_{2,3} = 1; \\
&S_{3,1} + S_{3,2} = 1; \\
&S_{4,1} + S_{4,2} = 1; \\
&S_{5,1} + S_{5,2} = 1; \\
&S_{6,1} + S_{6,2} + S_{6,3} = 1; \\
&0 * S_{2,2} + 2 * S_{2,3} + 2 * S_{1,1} - 0 * S_{1,2} - 2 * S_{1,3} \leq 2; \\
&- 2 * S_{1,1} + 0 * S_{1,2} + 2 * S_{1,3} - 0 * S_{2,2} - 2 * S_{2,3} \leq 6 - 2; \\
&- 2 * S_{3,1} + 0 * S_{3,2} - 0 * S_{2,2} - 2 * S_{2,3} - 1; \\
&0 * S_{2,2} + 2 * S_{2,3} + 2 * S_{3,1} - 0 * S_{3,2} \leq 6 - 4; \\
&- 2 * S_{4,1} + 0 * S_{4,2} + 2 * S_{3,1} - 0 * S_{3,2} \leq 4; \\
&- 2 * S_{3,1} + 0 * S_{3,2} + 2 * S_{4,1} - 0 * S_{4,2} \leq 6 - 6; \\
&- 2 * S_{5,1} + 0 * S_{5,2} + 2 * S_{4,1} - 0 * S_{4,2} \leq 3; \\
&- 2 * S_{4,1} + 0 * S_{4,2} + 2 * S_{5,1} - 0 * S_{5,2} \leq 6 - 5; \\
&- 2 * S_{6,1} + 0 * S_{6,2} + 2 * S_{6,3} + 2 * S_{5,1} - 0 * S_{5,2} \leq 3; \\
&- 2 * S_{5,1} + 0 * S_{5,2} + 2 * S_{6,1} - 0 * S_{6,2} - 2 * S_{6,3} \leq 6 - 5; \\
&S_{1,1} + S_{3,1} + S_{4,1} + S_{5,1} + S_{6,1} \leq max_overlap; \\
&S_{1,2} + S_{2,2} + S_{3,2} + S_{4,2} + S_{5,2} + S_{6,2} \leq max_overlap; \\
&S_{1,3} + S_{2,3} + S_{6,3} \leq max_overlap.
\end{aligned}$$

Compared with the mathematical formulations given in section 2.2, the number of constraints is reduced from 27 to 19.

By applying the mathematical solver, we have the results that $max_overlap = 2$, $S_{1,1} = 0$, $S_{1,2} = 0$, $S_{1,3} = 1$, $S_{2,2} = 1$, $S_{2,3} = 0$, $S_{3,1} = 1$, $S_{3,2} = 0$, $S_{4,1} = 1$, $S_{4,2} = 0$, $S_{5,1} = 0$, $S_{5,2} = 1$, $S_{6,1} = 0$, $S_{6,2} = 0$, and $S_{6,3} = 1$. In other words, the maximum number of registers scheduled in the same clocking domain is only 2 under the multi-domain clock skew schedule in which $T_{C1} = d_3$, $T_{C2} = d_2$, $T_{C3} = d_1$, $T_{C4} = d_1$, $T_{C5} = d_2$, and $T_{C6} = d_3$. Clearly, although both the number of binary variables and the number of constraints are dramatically reduced, the exactness of the solution is not sacrificed.

Finally, we point out an extension of the ASAP and ALAP scheduling algorithm. So far, as shown in Fig. 6, the procedure *OBTAIN_ASAP_ALAP* derives the ASAP schedule and the ALAP schedule of each register in the circuit graph. In fact, even if the clock arrival times of some registers are fixed, we also can derive the ASAP schedule and the ALAP schedule of other registers. The modifications to the procedure *OBTAIN_ASAP_ALAP* are straightforward.

4. ZONE-BASED SCHEDULING

For a large circuit, the number of binary variables may be still very large, even though all the redundancies are pruned. In this section, we propose a two-phase zone-based scheduling algorithm to solve a large circuit heuristically.

Given a circuit graph G and a target clock period P , Fig. 7 gives the procedure `ZONE_BASED_SCHEDULING`, which attempts to obtain a proper multi-domain clock skew schedule (with respect to the objective function) of a large circuit. The notation $sche$ denotes a multi-domain clock skew schedule. At the beginning, we derive a feasible multi-domain clock skew schedule² that works with clock period P as the initial solution of the multi-domain clock skew schedule $sche$.

```

Procedure ZONE_BASED_SCHEDULING( $G, P$ )
begin
   $sche$  = a feasible multi-domain clock skew schedule for the
  circuit graph  $G$  to work with clock period  $P$ ;
  if ( $sche == \text{NULL}$ ) then return ("no solution");
  prune all the redundancies according to the results of the
  procedure OBTAIN_ASAP_ALAP( $G, P$ );
   $phase = 1$ ;
   $RS$  = all the registers in the circuit graph  $G$ ;
   $sche = \text{PARTITION_RESCHEDULING}(phase, RS, G, P, sche)$ ;
   $phase = 2$ ;
  repeat
     $RS$  = all the dominant registers;
     $sche = \text{PARTITION_RESCHEDULING}(phase, RS, G, P, sche)$ ;
  until (the value of the objective function cannot be further improved);
  return ( $sche$ );
end.

```

Fig. 7. Procedure `ZONE_BASED_SCHEDULING`.

The kernel of the zone-based scheduling algorithm is to improve the multi-domain clock skew schedule $sche$ by iteratively applying the procedure `PARTITION_RESCHEDULING`. Fig. 8 gives the pseudo code of the procedure `PARTITION_RESCHEDULING`, in which the notation RS denotes the set of registers that are required to be re-scheduled. At the beginning, all the registers in the set RS are marked. At each time of repeat-until-loop iteration, we derive the ASAP schedules and the ALAP schedules of marked registers under the assumption that the clock arrival times of un-marked registers are fixed. As a result, the binary variables of each marked register can be obtained.³ A zone is defined as a part of registers extracted from the set RS . By specifying the constraint on the maxi-

² If the clock period P is larger than or equal to the longest path delay, the zero skew solution is obviously feasible. Otherwise, we can use the algorithm proposed in [11] to derive a feasible multi-domain clock skew scheduling. It is noteworthy to mention that no objective function is considered in [11].

³ At each time of repeat-until-loop iteration, we re-schedule some marked registers. Since there are more registers whose clock arrival times become fixed, the clocking constraints of remaining marked registers (*i.e.*, the registers that have not been re-scheduled) may be further restricted. As a result, some binary variables of remaining marked registers may become redundant and can be pruned. Therefore, the binary variables of a marked register may dynamically change at each time of repeat-until-loop iteration.

```

Procedure PARTITION_RESCEDULING(phase, RS, G, P, sche)
begin
  if (phase == 1) then RP =  $\emptyset$ 
  else RP = all the registers in the circuit graph G;
  mark all the registers in the set RS;
  repeat
    derive the ASAP schedule and the ALAP schedule of each marked register under
      the assumption that the clock arrival times of un-marked registers are fixed;
    obtain the binary variables of each marked register;
    re-schedule the registers that are associated with only one variable;
    un-mark the registers that are associated with only one variable;
    create a new zone to accommodate as many marked registers as possible;
  begin
    if (phase == 1) then the set RP includes all the un-marked registers and all the
      registers in the current zone;
    sche = re-schedule the current zone (under all the clocking constraints in the
      whole circuit) with respect to the multi-domain clock skew schedule sche and
      the objective function defined by the registers in the set RP;
    un-mark all the registers in the current zone;
  end;
  until (all the registers in the set RS are un-marked);
  return (sche);
end.

```

Fig. 8. Procedure PARTITION_RESCEDULING.

imum number of binary variables within a zone, in each time of repeat-until-loop iteration, a zone is created to accommodate as many marked registers as possible. When a zone is re-scheduled, the clock arrival times of other registers (*i.e.*, the registers outside the current zone) remain no change. Since the maximum number of binary variables within a zone is limited, the mathematical solver can re-schedule each zone efficiently. Note that a register is un-marked if it has been re-scheduled. The process of zone creation repeats until all the registers in the set *RS* become un-marked.

The philosophy of selecting marked registers into a zone is as below. Intuitively, if a register is associated with few binary variables, the flexibility in assigning its clock arrival time is limited; on the other hand, if a register is associated with many binary variables, the flexibility in assigning its clock arrival time is high. In fact, for those registers associated with only one binary variable, their clock arrival times can be determined immediately (*i.e.*, they can be re-scheduled immediately). Therefore, we only need to consider the registers associated with more than one binary variable. For each register associated with more than one binary variable, the fewer associated binary variables, the higher priority (to be selected into the current zone) it has. For example, if registers R_1 , R_2 , R_3 , R_4 , and R_6 are marked and associated with 3, 2, 2, 2, and 3 binary variables, respectively, and the maximum number of binary variables involved in a zone is limited to 6, then a zone $\{R_2, R_3, R_4\}$ will be created.

In fact, as shown in Fig. 7, the zone-based scheduling has the following two phases to apply the procedure PARTITION_RESCEDULING:

- (1) The first phase is to derive a multi-domain clock skew schedule by re-scheduling all the registers in the circuit graph G . The set RP includes the registers that have been re-scheduled (*i.e.*, un-marked) and the registers that are in the current zone. When we re-schedule a zone, the corresponding objective function is defined only by the registers in the set RP .
- (2) The second phase is an iteration process to improve the multi-domain clock skew schedule. Note that, in the second phase, the objective function is defined by all the registers in the circuit graph G . A register is called a *dominant register*, if and only if it determines the current value of the objective function. For example, supposing the objective function is to minimize the value of $max_overlap$, if register R_3 is scheduled to d_1 , registers R_2 , R_4 , and R_5 are scheduled to d_2 , and registers R_1 and R_6 are scheduled to d_3 , then registers R_2 , R_4 , and R_5 are dominant registers since they determine the current value of $max_overlap$. Clearly, the current value of the objective function cannot be further improved, unless there are some dominant registers whose clock arrival times can be adjusted. Therefore, in each time of repeat-until-loop iteration, we apply the procedure PARTITION_RESCEDULING to re-schedule all the current dominant registers. The iteration process repeats until the current value of the objective function cannot be further improved.

Suppose that the constraint on the maximum number of binary variables involved in a zone is 6. We use the circuit graph shown in Fig. 1 as an example to illustrate the zone-based scheduling algorithm. Assume that the target clock period $P = 6$, and we are given 3 clocking domains: $d_1 = -2$, $d_2 = 0$ and $d_3 = 2$. At the beginning, we derive a feasible multi-domain clock skew schedule in which $T_{C1} = d_2$, $T_{C2} = d_2$, $T_{C3} = d_2$, $T_{C4} = d_2$, $T_{C5} = d_2$ and $T_{C6} = d_2$ as the initial solution. Thus, the initial value of $max_overlap$ is 6.

In the first phase of zone-based scheduling, we use the procedure PARTITION_RESCEDULING to re-schedule all the registers in the circuit graph; *i.e.*, $RS = \{R_1, R_2, R_3, R_4, R_5, R_6\}$. Initially, all the registers in the set RS are marked. The *ASAP* schedule and the *ALAP* schedule of each register the set RS is as below: $[ASAP_1, ALAP_1] = [-2, 2]$, $[ASAP_2, ALAP_2] = [0, 2]$, $[ASAP_3, ALAP_3] = [-2, 0]$, $[ASAP_4, ALAP_4] = [-2, 0]$, $[ASAP_5, ALAP_5] = [-2, 0]$ and $[ASAP_6, ALAP_6] = [-2, 2]$. Therefore, registers R_1 , R_2 , R_3 , R_4 , R_5 , and R_6 are associated with 3, 2, 2, 2, 2 and 3 binary variables. Since the maximum number of binary variables involved in a zone is limited to 6, a zone $Z_1 = \{R_2, R_3, R_4\}$ is created. The binary variables associated with zone Z_1 are $S_{2,2}$, $S_{2,3}$, $S_{3,1}$, $S_{3,2}$, $S_{4,1}$ and $S_{4,2}$. We re-schedule the zone $Z_1 = \{R_2, R_3, R_4\}$ under $T_{C1} = d_2$, $T_{C5} = d_2$ and $T_{C6} = d_2$. Note that the set RP is also the same as the zone Z_1 . Therefore, the objective function defined by the set RP is to minimize the value of $max_overlap_{Z_1}$, where the notation $max_overlap_{Z_1}$ denotes the maximum number of registers in zone Z_1 (*i.e.*, in the set RP) that are re-scheduled in the same clocking domain. The mathematical formulations are as below:

minimize $max_overlap_{Z_1}$
subject to

$$\begin{aligned} S_{2,2} + S_{2,3} &= 1; \\ S_{3,1} + S_{3,2} &= 1; \\ S_{4,1} + S_{4,2} &= 1; \end{aligned}$$

$$\begin{aligned}
&0 * S_{2,2} + 2 * S_{2,3} - 0 \leq 2; \\
&0 - 0 * S_{2,2} - 2 * S_{2,3} \leq 6 - 2; \\
&-2 * S_{3,1} + 0 * S_{3,2} - 0 * S_{2,2} - 2 * S_{2,3} \leq 1; \\
&0 * S_{2,2} + 2 * S_{2,3} + 2 * S_{3,1} - 0 * S_{3,2} \leq 6 - 4; \\
&-2 * S_{4,1} + 0 * S_{4,2} + 2 * S_{3,1} - 0 * S_{3,2} \leq 4; \\
&-2 * S_{3,1} + 0 * S_{3,2} + 2 * S_{4,1} - 0 * S_{4,2} \leq 6 - 6; \\
&0 + 2 * S_{4,1} - 0 * S_{4,2} \leq 3; \\
&-2 * S_{4,1} + 0 * S_{4,2} - 0 \leq 6 - 5; \\
&S_{3,1} + S_{4,1} \leq \max_overlap_Z_1; \\
&S_{2,2} + S_{3,2} + S_{4,2} \leq \max_overlap_Z_1; \\
&S_{2,3} \leq \max_overlap_Z_1.
\end{aligned}$$

Note that the constraint $0 * S_{2,2} + 2 * S_{2,3} - 0 \leq 2$, the constraint $0 - 0 * S_{2,2} - 2 * S_{2,3} \leq 6 - 2$; the constraint $-2 * S_{3,1} + 0 * S_{3,2} - 0 * S_{2,2} - 2 * S_{2,3} \leq 1$, the constraint $-2 * S_{4,1} + 0 * S_{4,2} + 2 * S_{3,1} - 0 * S_{3,2} \leq 4$, the constraint $0 + 2 * S_{4,1} - 0 * S_{4,2} \leq 3$, and the constraint $-2 * S_{4,1} + 0 * S_{4,2} - 0 \leq 6 - 5$ are redundant and can be pruned. By applying the mathematical solver, we have the results that $\max_overlap_Z_1 = 2$, $S_{2,2} = 1$, $S_{2,3} = 0$, $S_{3,1} = 1$, $S_{3,2} = 0$, $S_{4,1} = 0$ and $S_{4,2} = 1$. As a result, we have $T_{C2} = d_2$, $T_{C3} = d_1$, and $T_{C4} = d_2$. After the zone Z_1 is re-scheduled, we have $T_{C1} = d_2$, $T_{C2} = d_2$, $T_{C3} = d_1$, $T_{C4} = d_2$, $T_{C5} = d_2$, $T_{C6} = d_2$, and the value of $\max_overlap$ is 5. Thus, the value of $\max_overlap$ is reduced from 6 to 5. Since registers R_2 , R_3 , and R_4 have been re-scheduled, they are un-marked.

Next, since registers R_1 , R_5 , and R_6 are still marked, we attempt to re-schedule them. Under the assumption that $T_{C2} = d_2$, $T_{C3} = d_1$, and $T_{C4} = d_2$, we have $[ASAP_1, ALAP_1] = [-2, 2]$, $[ASAP_5, ALAP_5] = [0, 0]$ and $[ASAP_6, ALAP_6] = [0, 2]$. Therefore, registers R_1 , R_5 , and R_6 are associated with 3, 1, and 2 binary variables, respectively. Since the number of binary variable is 1, register R_5 is re-scheduled to the clocking domain d_2 immediately. So far, registers R_2 , R_3 , R_4 , and R_5 have been un-marked. Then, a zone $Z_2 = \{R_1, R_6\}$ is created. There are five binary variables associated with zone Z_2 : $S_{1,1}$, $S_{1,2}$, $S_{1,3}$, $S_{6,2}$, and $S_{6,3}$. We re-schedule the zone $Z_2 = \{R_1, R_6\}$ under $T_{C2} = d_2$, $T_{C3} = d_1$, $T_{C4} = d_2$, and $T_{C5} = d_2$. Since the set RP includes all the un-marked registers and the registers in the zone Z_2 , the set RP has become $\{R_1, R_2, R_3, R_4, R_5, R_6\}$. Thus, the objective function defined by the set RP corresponds to minimize the value of $\max_overlap$. After pruning the redundant constraints, the mathematical formulations are as below:

minimize $\max_overlap$
subject to

$$\begin{aligned}
&S_{1,1} + S_{1,2} + S_{1,3} = 1; \\
&S_{6,2} + S_{6,3} = 1; \\
&0 * S_{6,2} + 2 * S_{6,3} - 0 \leq 3; \\
&0 - 0 * S_{6,2} - 2 * S_{6,3} \leq 6 - 5; \\
&S_{1,1} + 1 \leq \max_overlap; \\
&S_{1,2} + S_{6,2} + 1 + 1 \leq \max_overlap; \\
&S_{1,3} + S_{6,3} \leq \max_overlap.
\end{aligned}$$

By applying the mathematical solver, we have the results that $max_overlap = 3$, $S_{1,1} = 0$, $S_{1,2} = 0$, $S_{1,3} = 1$, $S_{6,2} = 0$, and $S_{6,3} = 1$. As a result, we have $T_{C1} = d_3$, $T_{C2} = d_2$, $T_{C3} = d_1$, $T_{C4} = d_2$, $T_{C5} = d_2$, $T_{C6} = d_3$, and the value of $max_overlap$ is reduced from 5 to 3. Since registers R_1 and R_6 have been re-scheduled, they are un-marked. Therefore, all the registers in the circuit graph (*i.e.*, all the registers in the set RS) have been un-marked. As a result, the first phase of zone-based scheduling is completed.

Next, we enter the first iteration in the second phase of zone-based scheduling. The current value of $max_overlap$ is 3. In the current multi-domain clock skew schedule, register R_3 is scheduled to d_1 , registers R_2 , R_4 , and R_5 are scheduled to d_2 , and registers R_1 and R_6 are scheduled to d_3 . Clearly, registers R_2 , R_4 , and R_5 are dominant registers, because they determine the current value of $max_overlap$. Letting $RS = \{R_2, R_4, R_5\}$, the procedure PARTITION_RESCHEULING is applied to further reduce the value of $max_overlap$. At the beginning, registers R_2 , R_4 , and R_5 are marked. Under the assumption that $T_{C1} = d_3$, $T_{C3} = d_1$, and $T_{C6} = d_3$, we have $[ASAP_2, ALAP_2] = [0, 0]$, $[ASAP_4, ALAP_4] = [-2, 0]$, and $[ASAP_5, ALAP_5] = [0, 0]$. Therefore, the binary variables of registers R_2 , R_4 , and R_5 are 1, 2, and 1, respectively. Since the number of binary variable is 1, register R_5 and register R_5 are re-scheduled to the clocking domain d_2 immediately. Then, registers R_2 and R_5 are un-marked. A zone $Z_3 = \{R_4\}$ is created. There are two binary variables associated with zone Z_3 : $S_{4,1}$ and $S_{4,2}$. We re-schedule zone Z_3 under $T_{C1} = d_3$, $T_{C2} = d_2$, $T_{C3} = d_1$, $T_{C5} = d_2$, and $T_{C6} = d_3$. After pruning the redundant constraints, the mathematical formulations are as below:

minimize $max_overlap$
subject to

$$\begin{aligned} S_{4,1} + S_{4,2} &= 1; \\ S_{4,1} + 1 &\leq max_overlap; \\ 1 + S_{4,2} + 1 &\leq max_overlap; \\ 1 + 1 &\leq max_overlap. \end{aligned}$$

By applying the mathematical solver, we have the results that $max_overlap = 2$, $S_{4,1} = 1$, and $S_{4,2} = 0$. As a consequence, we have $T_{C1} = d_3$, $T_{C2} = d_2$, $T_{C3} = d_1$, $T_{C4} = d_1$, $T_{C5} = d_2$, and $T_{C6} = d_3$, and the value of $max_overlap$ is reduced from 3 to 2. Since all the registers in the set RS have been re-scheduled (*i.e.*, all the registers in the set RS have been un-marked), the first iteration in the second phase is finished. It is noteworthy to mention that, this circuit graph has 6 registers and we are given 3 clocking domains, the lower bound on the value of $max_overlap$ is 2. Therefore, in fact, the lower bound on the value of $max_overlap$ has been achieved.

We enter the second iteration in the second phase of zone-based scheduling. The current value of $max_overlap$ is 2. We find that registers R_3 and R_4 are scheduled to d_1 , registers R_2 and R_5 are scheduled to d_2 , and registers R_1 and R_6 are scheduled to d_3 . In other words, each clocking domain has two registers scheduled into. Therefore, all the registers are dominant registers for the current value of $max_overlap$. Thus, the set RS must include all the registers in the circuit graph. However, since the lower bound on the value of $max_overlap$ has been achieved, the procedure PARTITION_RESCHEULING

cannot further reduce the value of $max_overlap$. As a consequence, the second phase of zone-based scheduling is completed.

After the zone-based scheduling is completed, the value of $max_overlap$ is 2 under the multi-domain clock skew schedule in which $T_{C1} = d_3$, $T_{C2} = d_2$, $T_{C3} = d_1$, $T_{C4} = d_1$, $T_{C5} = d_2$, and $T_{C6} = d_3$.

5. EXPERIMENTAL RESULTS

Our approach has been implemented in C++ programming language running on a personal computer with AMD K8-3GHz CPU and 1G Bytes RAM. We use Extended LINGO Release 8.0 as the mathematical solver. The circuits from ISCAS'89 benchmark suites are targeted to TSMC 0.35 μ m cell library for the experiments. Table 2 tabulates the characteristics of benchmark circuits. The columns *Registers*, *Gates*, *Data paths*, and *Target clock period* denote the number of registers, the number of gates, the number of data path, and the target clock period, respectively.

Table 2. Summary of benchmark circuits.

Circuit	Registers	Gates	Data paths	Target clock period
S444	21	119	175	4.24
S499	22	120	528	3.99
S1269	37	437	455	9.89
S1512	57	413	686	5.50
S3271	116	1035	1137	8.50
S3384	183	1070	1831	19.86
S5378	179	1001	2313	5.16
S6669	239	2155	2179	28.42
S9234	228	2680	2830	9.42
S13207	669	2573	4660	10.92
S15850	597	3448	16863	14.23
S35932	1728	12204	5159	5.14
S38584	1452	11448	15329	11.85

Without loss of generality, in each benchmark circuit, we assume that: (1) the target clock period P is the longest path delay; (2) we are given $2 \times \lfloor \frac{P}{0.3} \rfloor + 1$ clocking domains, where 0.3 ns denotes the clock skew between two consecutive clocking domains; (3) each clocking domain $d_k = 0.3 \times (k - \lfloor \frac{P}{0.3} \rfloor - 1)$ ns, where $k = 1, \dots, 2 \times \lfloor \frac{P}{0.3} \rfloor + 1$. Under these design assumptions, we apply the procedure *OBTAIN ASAP ALAP* to all the benchmark circuits. We find that, for each benchmark circuit, the ASAP and ALAP schedule of each register can be derived within 1 second.

In the following, we use two different objective functions to test the generality and effectiveness of our methodology. The objective function used in section 5.1 is to minimize the value of $max_overlap$. The objective function used in section 5.2 is to minimize the maximum deviation between the computed clock skew and the ideal clock skew schedule (for the tolerance to process variation).

5.1 The Minimization of Max_Overlap

Here, our objective function is to minimize the value of $max_overlap$ under the constraint of given clocking domains. Table 3 demonstrates the advantage of exploiting the *ASAP* and *ALAP* schedule of each register to prune the redundancies. The column *Original Formulation* describes the original mathematical formulations proposed by [14]. The column *Reduced Formulation* describes our reduced mathematical formulations, in which all the redundant binary variables and redundant constraints are pruned. The column *#vars* denotes the number of binary variables, the column *#cons* denotes the number of constraints, the column *max overlap* denotes the value of $max_overlap$, and the column *CPU time* gives the CPU time (in seconds) spent in the mathematical solver. We use the notation *t/o* to denote that the problem complexity cannot be solved within twelve hours.

Table 3. The advantage of pruning redundancies for the minimization of $max_overlap$.

Circuit	[14]				Reduced Formulation			
	#vars	#cons	max overlap	CPU time (s)	#vars	#cons	max overlap	CPU time (s)
S444	609	400	2	4	239	355	2	1
S499	594	1105	6	6	80	524	6	1
S1269	2405	1012	4	9	312	411	4	3
S1512	2109	1466	6	5	326	825	6	1
S3271	6612	2447	4	30087	3342	2027	4	88
S3384	24339	4316	–	t/o	4917	3531	5	24
S5378	6265	4860	11	16046	2088	342	11	59
S6669	45171	5476	–	t/o	3951	3265	11	122
S9234	14364	6803	–	t/o	6554	6042	7	232
S13207	48837	10062	–	t/o	19586	8416	16	362
S15850	56715	34418	–	t/o	18124	28314	–	t/o
S35932	60480	12081	288	714	17516	3714	288	271
S38584	114708	32189	–	t/o	40018	29569	–	t/o

Table 3 demonstrates that our reduced mathematical formulations do speed up the process of multi-domain clock skew schedule. Especially, in benchmark circuits S3384, S6669, S9234 and S13207, the mathematical solver cannot solve the original formulations within twelve hours, but the mathematical solver can solve our reduced formulations within twelve hours. However, on the other hand, in benchmark circuits S15850 and S38584, although a lot of redundant binary variables and a lot of redundant constraints can be pruned, the corresponding problem complexity still cannot be solved within twelve hours.

Table 4 demonstrates the results of zone-based scheduling. The benchmark circuits that have more than 2000 irredundant binary variables are used to test the effectiveness of zone-based scheduling. For the completeness of our experiments, we assume that the constraint on the maximum number of binary variables involved in a zone is 500, 1000 and 2000, respectively. Without loss of generality, we use the zero skew solution as the

Table 4. The results of zone-based scheduling for the minimization of $max_overlap$.

Circuit	500 variables in a zone			1000 variables in a zone			2000 variables in a zone		
	max overlap	tackled zones	CPU time (s)	max overlap	tackled zones	CPU time (s)	max overlap	tackled zones	CPU time (s)
S3271	4	8	10	4	6	14	4	4	88
S3384	5	10	5	5	6	8	5	4	14
S5378	11	7	7	11	5	28	11	2	54
S6669	11	10	11	11	6	20	11	5	26
S9234	7	8	23	7	7	25	7	5	37
S13207	16	28	205	16	13	238	16	11	285
S15850	15	29	501	15	19	557	15	12	712
S35932	288	35	133	288	18	169	288	9	240
S38584	38	54	334	37	32	420	36	21	513

initial solution. The column $max_overlap$ denotes the value of $max_overlap$. The column $tackled_zones$ denotes the number of times to use the mathematical solver to re-schedule a zone. The column $CPU\ time$ gives the CPU time (in seconds) spent in both the procedure ZONE_BASED_SCHEDULING and the mathematical solver. On the other hand, from Table 3, we can find the results of *whole-circuit scheduling*, which solves the whole circuit as a single zone. Clearly, in each benchmark circuit, the zone-based scheduling is much faster than the whole-circuit scheduling.

Different from the whole-circuit scheduling, the zone-based scheduling does not guarantee the optimality. However, with an analysis to both Tables 3 and 4, the following two facts show that the zone-based scheduling is a very good heuristic:

- (1) In most benchmark circuits, the zone-based scheduling achieves the same results no matter the constraint on the maximum number of binary variables involved in a zone is 500, 1000 or 2000.
- (2) For each benchmark circuit that can be solved by the whole-circuit scheduling, the zone-based scheduling achieves the same results as the whole-circuit scheduling.

Moreover, from Table 4, we have the following observations. If the maximum number of binary variables involved in a zone is larger, the smaller value of $max_overlap$ can be achieved. Using the benchmark circuit S38584 as an example, if the maximum number of binary variables involved in a zone is 500, the value of $max_overlap$ is 38; if the maximum number of binary variables involved in a zone is 2000, the value of $max_overlap$ is 36. However, the more binary variables involved in a zone, the more CPU time is used. Using the benchmark circuit S38584 as an example, if the maximum number of binary variables involved in a zone is 500, the CPU time is 334 seconds; if the maximum number of binary variables involved in a zone is 2000, the CPU time is 513 seconds.

Figs. 9 and 10 give the snapshots of zone-based scheduling on benchmark circuits S5378 and S6669, respectively. The x-axis denotes the number of times to use the mathematical solver to re-schedule a zone. The y-axis denotes the value of $max_overlap$. If the maximum number of binary variables involved in a zone is larger, more registers can be

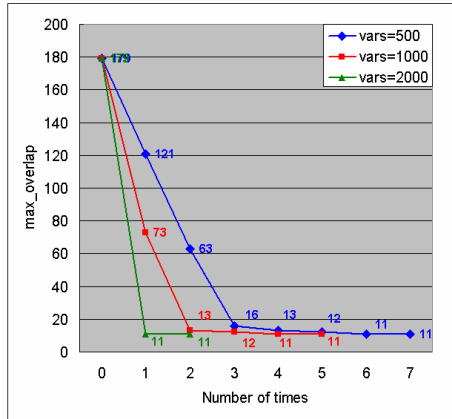


Fig. 9. Zone-based scheduling on the circuit S5378 for the minimization of *max_overlap*.

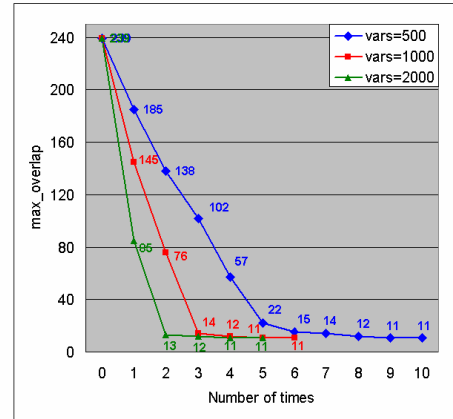


Fig. 10. Zone-based scheduling on the circuit S6669 for the minimization of *max_overlap*.

accommodated into a zone. Therefore, if the maximum number of binary variables involved in a zone is larger, the less number of zones are created. Using the benchmark circuit S5378 shown in Fig. 9 as an example, if the maximum number of binary variables involved in a zone is 2000, the number of times is 2 (*i.e.*, the number of zones is 2); on the other hand, if the maximum number of binary variables involved in a zone is 500, the number of times is 7 (*i.e.*, the number of zones is 7).

5.2 The Minimization of *Max_Deviation*

The clock skew can be utilized to improve the tolerance to process variation. For each data path $R_i \rightarrow R_j$, the permissible clock skew is within the range from $-T_{PDi,j(\min)}$ to $P - T_{PDi,j(\max)}$. To maximize the tolerance to process variation, the ideal clock skew schedule is the one in which the clock skew for each data path is at the middle of the permissible clock skew range. Of course, due to the timing constraints, this ideal clock skew schedule is unlikely to also be feasible. In [8], a linear program has been proposed to obtain a feasible clock skew schedule which is as close as possible to the ideal clock skew schedule. Let the notation *max_deviation* denote the maximum deviation from the ideal clock skew schedule among all the data paths. For each data path $R_i \rightarrow R_j$, let $g_{i,j}$ denote the middle of the permissible clock skew range. The mathematical formulations in [8] are as below:

$$\begin{aligned}
 &\text{minimize } \textit{max_deviation} \\
 &T_{ci} - T_{cj} \geq -T_{PDi,j(\min)}; \\
 &T_{cj} - T_{ci} \leq P - T_{PDi,j(\max)}; \\
 &T_{ci} - T_{cj} \geq g_{i,j} - \textit{max_deviation}; \\
 &T_{ci} - T_{cj} \geq g_{i,j} + \textit{max_deviation}; \\
 &\textit{Max_deviation} \geq 0.
 \end{aligned}$$

In the following, we apply our methodology to the problem of minimizing the value

of $max_deviation$ under the constraint of given clocking domains. Note that, different from previous work [8], we restrict the clock arrival time of each register to given clocking domains. Therefore, the mathematical formulations are rewritten as below:

minimize $max_deviation$

$$\begin{aligned} \sum_{k=1}^n S_{i,k} &= 1; \\ \sum_{k=1}^n d_k \cdot S_{i,k} - \sum_{k=1}^n d_k \cdot S_{j,k} &\geq -T_{PDi(\min)}; \\ \sum_{k=1}^n d_k \cdot S_{i,k} - \sum_{k=1}^n d_k \cdot S_{j,k} &\geq P - T_{PDi,j(\max)}; \\ \sum_{k=1}^n d_k \cdot S_{i,k} - \sum_{k=1}^n d_k \cdot S_{j,k} &\geq g_{i,j} - max_deviation; \\ \sum_{k=1}^n d_k \cdot S_{i,k} - \sum_{k=1}^n d_k \cdot S_{j,k} &\leq g_{i,j} + max_deviation; \\ max_deviation &\geq 0. \end{aligned}$$

Table 5. The advantage of pruning redundancies for the minimization of $max_deviation$.

Circuit	[14]				Reduced Formulation			
	#vars	#cons	max deviation	CPU time (s)	#vars	#cons	max deviation	CPU time (s)
S444	609	723	1.125	9	239	665	1.125	6
S499	594	2136	1.705	5	80	1412	1.705	1
S1269	2405	1859	4.420	13	312	1281	4.420	4
S1512	2109	2804	2.325	13	326	1935	2.325	4
S3271	6612	4666	3.260	34697	3342	4190	3.260	92
S3384	24339	8241	–	t/o	4917	7393	9.360	30
S5378	6265	9433	1.940	8059	2088	7039	1.940	64
S6669	45171	10337	–	t/o	3951	8093	13.610	128
S9234	14364	13270	–	t/o	6554	12410	4.285	243
S13207	48837	19311	–	t/o	19586	17353	5.060	389
S15850	56715	68051	–	t/o	18124	59827	–	t/o
S35932	60480	22366	2.410	856	17516	13594	2.410	427
S38584	114708	62770	–	t/o	40018	58846	–	t/o

Table 5 demonstrates the advantage of exploiting the ASAP and ALAP schedule of each register to prune the redundancies. The column $\#vars$ denotes the number of binary variables, the column $\#cons$ denotes the number of constraints, the column $max_deviation$ denotes the value of $max_overlap$, and the column $CPU\ time$ gives the CPU time (in seconds) spent in the mathematical solver. We use the notation t/o to denote that the problem complexity cannot be solved within twelve hours. Clearly, our reduced mathematical formulations do speed up the process of multi-domain clock skew schedule. Es-

pecially, in benchmark circuits S3384, S6669, S9234 and S13207, the mathematical solver cannot solve the original formulations within twelve hours, but the mathematical solver can solve the reduced formulations within twelve hours. However, on the other hand, in benchmark circuits S15850 and S38584, although a lot of redundant binary variables and redundant constraints can be pruned, the corresponding problem complexity still cannot be solved within twelve hours.

Table 6. The results of zone-based scheduling for the minimization of $max_deviation$.

Circuit	500 variables in a zone			1000 variables in a zone			2000 variables in a zone		
	max deviation	tackled zones	CPU time (s)	max overlap	tackled zones	CPU time (s)	max overlap	tackled zones	CPU time (s)
S3271	3.260	8	13	3.260	5	23	3.260	4	92
S3384	9.360	12	8	9.360	9	12	9.360	7	21
S5378	1.940	10	11	1.940	7	19	1.940	5	34
S6669	13.610	10	21	13.610	8	27	13.610	6	58
S9234	4.285	9	23	4.285	6	35	4.285	4	57
S13207	5.060	21	205	5.060	12	278	5.060	7	326
S15850	6.415	26	557	6.375	14	617	6.325	9	706
S35932	2.410	37	396	2.410	19	504	2.410	11	633
S38584	5.350	48	478	5.350	26	520	5.300	15	791

Table 6 demonstrates the results of zone-based scheduling. The column $max_deviation$ denotes the value of $max_deviation$ in nanoseconds. The column $tackled_zones$ denotes the number of times to use the mathematical solver to re-schedule a zone. The column CPU_time gives the CPU time (in seconds) spent in both the procedure ZONE_BASED_SCHEDULING and the mathematical solver. Although the zone-based scheduling does not guarantee the optimality, the following two facts show that the zone-based scheduling is a very good heuristic:

- (1) In most benchmark circuits, the zone-based scheduling achieves the same results no matter the constraint on the maximum number of binary variables involved in a zone is 500, 1000 or 2000.
- (2) For each benchmark circuit that can be solved by the whole-circuit scheduling, the zone-based scheduling achieves the same results as the whole-circuit scheduling.

Moreover, from Table 6, we have the following observations. If the maximum number of binary variables involved in a zone is larger, the smaller value of $max_deviation$ can be achieved. Using the benchmark circuit S38584 as an example, if the maximum number of binary variables involved in a zone is 500, the value of $max_deviation$ is 5.350 nanoseconds; if the maximum number of binary variables involved in a zone is 2000, the value of $max_deviation$ is 5.300 nanoseconds. However, the more binary variables involved in a zone, the more CPU time is used. Using the benchmark circuit S38584 as an example, if the maximum number of binary variables involved in a zone is 500, the CPU time used is 478 seconds; if the maximum number of binary variables involved in a zone is 2000, the CPU time is 791 seconds.

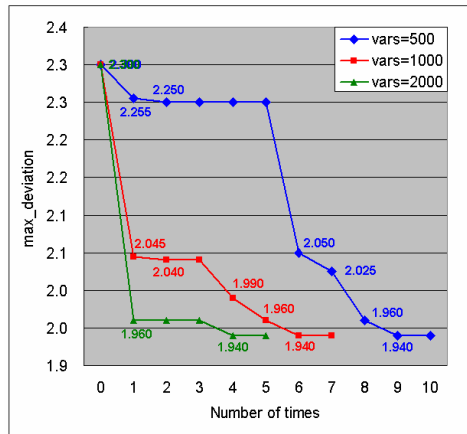


Fig. 11. Zone-based scheduling on the circuit S5378 for the minimization of $max_deviation$.

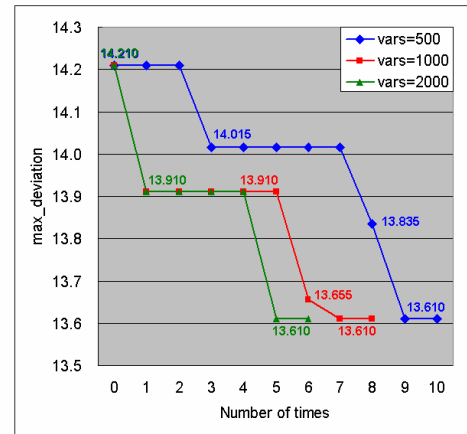


Fig. 12. Zone-based scheduling on the circuit S6669 for the minimization of $max_deviation$.

Figs. 11 and 12 give the snapshots of zone-based scheduling on benchmark circuits S5378 and S6669, respectively. The x-axis denotes the number of times to re-schedule a zone. The y-axis denotes the value of $max_deviation$. If the maximum number of binary variables involved in a zone is larger, more registers can be accommodated into a zone. Therefore, if the maximum number of binary variables involved in a zone is larger, the less number of zones are created. Using the benchmark circuit S5378 shown in Fig. 11 as an example, if the maximum number of binary variables involved in a zone is 2000, the number of times is 5 (*i.e.*, the number of zones is 5); on the other hand, if the maximum number of binary variables involved in a zone is 500, the number of times is 10 (*i.e.*, the number of zones is 10).

Some readers may be interested in the following question: when the objective function of multi-domain clock skew scheduling is to minimize the value of $max_deviation$, how large is the value of $max_overlap$ in the derived multi-domain clock skew schedule? For the convenience of comparisons, Table 7 tabulates the values of $max_overlap$ associated with different combinations of objective functions and zone sizes. The column “*objective function: minimize max_overlap*” gives the value of $max_overlap$ when the objective function is to minimize the value of $max_overlap$. On the other hand, the column “*objective function: minimize max_deviation*” gives the value of $max_overlap$ when the objective function is to minimize the value of $max_deviation$. From Table 7, we have the following two observations. First, when the objective function is to minimize the value of $max_deviation$, the value of $max_overlap$ becomes higher. Second, when the objective function is to minimize the value of $max_deviation$, the values of $max_overlap$ are independent of zone sizes.

6. CONCLUSIONS

In this paper, we present a methodology to speed up the process of multi-domain clock skew scheduling. Given a target clock period and an objective function, our goal is

Table 7. The values of *max_overlap* associated with different combinations of objective functions and zone sizes.

Circuit	objective function: minimize <i>max_overlap</i>			objective function: minimize <i>max_deviation</i>		
	500 variables in a zone	1000 variables in a zone	2000 variables in a zone	500 variables in a zone	1000 variables in a zone	2000 variables in a zone
S3271	4	4	4	17	34	22
S3384	5	5	5	33	22	23
S5378	11	11	11	52	45	47
S6669	11	11	11	52	47	56
S9234	7	7	7	28	28	30
S13207	16	16	16	103	100	71
S15850	15	15	15	60	55	65
S35932	288	288	288	511	540	511
S38584	38	37	36	282	260	206

to optimize the objective function under the architecture of multiple clocking domains. Compared with previous work, our contribution includes the following two aspects. First, we derive the ASAP and ALAP schedule of each register to reduce the problem size without sacrificing the exactness of the solution; secondly, we propose a two-phase zone-based scheduling to solve a large circuit heuristically.

The proposed algorithms, including ASAP scheduling, ALAP scheduling and zone-based scheduling are independent of the objective functions. Two different objective functions are used in our experiments to demonstrate the generality and effectiveness of our methodology. Experimental results consistently show that our methodology works very well in practice.

REFERENCES

1. J. P. Fishburn, "Clock skew optimization," *IEEE Transactions on Computers*, Vol. 39, 1990, pp. 945-951.
2. R. B. Deokar and S. S. Sapatnekar, "A graph-theoretic approach to clock skew optimization," in *Proceedings of IEEE International Symposium on Circuits and Systems*, Vol. 1, 1994, pp. 407-410.
3. N. Maheshwari and S. S. Sapatnekar, *Timing Analysis and Optimization of Sequential Circuits*, Kluwer Academic Publishers, 1999.
4. C. Albrecht, B. Korte, J. Schietke, and J. Vygen, "Cycle time and slack optimization for VLSI chips," in *Proceedings of IEEE/ACM International Conference on Computer Aided Design*, 1999, pp. 232-238.
5. S. H. Huang and Y. T. Nieh, "Clock period minimization of non-zero clock skew circuits," in *Proceedings of IEEE/ACM International Conference on Computer Aided Design*, 2003, pp. 809-812.
6. S. H. Huang, Y. T. Nieh, and F. P. Lu, "Race-condition-aware clock skew scheduling," in *Proceedings of IEEE/ACM Design Automation Conference*, 2005, pp. 475-478.

7. I. S. Kourtev and E. G. Friedman, "Clock skew scheduling for improved reliability via quadratic programming," in *Proceedings of IEEE/ACM International Conference on Computer Aided Design*, 1999, pp. 293-243.
8. R. Mader, E. G. Friedman, A. Litman, and I. S. Kourtev, "Large scale clock skew scheduling techniques for improved reliability of digital synchronous VLSI circuits," in *Proceedings of IEEE International Symposium on Circuits and Systems*, Vol. 1, 2002, pp. 357-360.
9. L. Benini, P. Vuillod, A. Bogliolo, and G. De Micheli, "Clock skew optimization for peak current reduction," *Journal of VLSI Signal Processing*, Vol. 16, 1997, pp. 117-130.
10. K. Ravindran, A. Keuhlmann, and E. Sentovich, "Multi-domain clock skew scheduling," in *Proceedings of IEEE/ACM International Conference on Computer Aided Design*, 2003, pp. 801-808.
11. K. M. Carrig, "Chip clocking effect on performance for IBM's SA-27E ASIC technology," *IBM Micronews*, Vol. 6, 2000, pp. 12-16.
12. D. P. Singh and S. D. Brown, "Constrained clock shifting for field programmable gate arrays," in *Proceedings of ACM International Symposium on Field Programmable Gate Arrays*, 2002, pp. 121-126.
13. J. P. Fishburn, "Solving a system of difference constraints with variables restricted to a finite set," *Information Processing Letters*, Vol. 82, 2002, pp. 143-144.
14. A. Vittal, H. Ha, F. Brewer, and M. Marek-Sadowska, "Clock skew optimization for ground bounce control," in *Proceedings of IEEE/ACM International Conference on Computer Aided Design*, 1996, pp. 395-399.



Shih-Hsu Huang (黃世旭) received the B.S. degree in Computer Science and Information Engineering from National Chiao Tung University, Hsinchu, Taiwan, R.O.C., in 1989, the M.S. degree in Computer Science from National Tsing Hua University, Hsinchu, Taiwan, R.O.C., in 1991, and the Ph.D. degree in Computer Science and Information engineering from National Taiwan University, Taipei, Taiwan, R.O.C., in 1995. From 1995 to 2000, he was with computer and communications research laboratories, industrial technology research institute, Hsinchu, Taiwan, R.O.C., rising to the position of deputy manager of IC design department, responsible for the design of high performance IC's. In 2000, he joined the Department of Electronic Engineering, Chung Yuan Christian University, Chungli, Taiwan, R.O.C., as a faculty member, where he is currently an Associate Professor. His research interests include timing optimization, digital design, and physical synthesis.



Chia-Ming Chang (張家銘) received the B.S. degree in Electronic Engineering from Chun Yuan Christian University, Chungli, Taiwan, R.O.C., in 2002, and the M.S. degree in Electronic Engineering from Chung Yuan Christian University, Chungli, Taiwan, R.O.C., in 2004. He is presently working toward the Ph.D. degree in Electronic Engineering at Chung Yuan Christian University, Chungli, Taiwan, R.O.C. His research interests include low power design and clock tree synthesis.



Yow-Tyng Nieh (聶佑庭) received the B.S. degree in Electrical Engineering from National Sun Yat-Sen University, Kaohsiung, Taiwan, R.O.C., in 1996, and the M.S. degree in Electronic Engineering from Chung Yuan Christian University, Chungli, Taiwan, R.O.C., in 2003. He is presently working toward the Ph.D. degree in Electronic Engineering at Chung Yuan Christian University, Chungli, Taiwan, R.O.C. His research interests include timing optimization, logic synthesis, and physical synthesis.