Fast Multi-Domain Clock Skew Scheduling For Peak Current Reduction

Shih-Hsu Huang, Chia-Ming Chang, and Yow-Tyng Nieh

Department of Electronic Engineering
Chung Yuan Christian University
Chung Li, Taiwan, R.O.C.

Abstract - Given several specific clocking domains, the peak current minimization problem can be formulated as a 0-1 integer linear program. However, if the number of binary variables is large, the run time is unacceptable. In this paper, we study the reduction of this high computational expense. Our approach includes the following two aspects. First, we derive the ASAP schedule and the ALAP schedule to prune the redundancies without sacrificing the exactness (optimality) of the solution. Second, we propose a zone-based scheduling algorithm to solve a large circuit heuristically.

I. Introduction

It is well known that the clock skew can be utilized to shorten the clock period [1] or reduce the peak current [2]. However, it is very difficult to implement a wide spectrum of dedicated clock delays.

The architecture of multiple clocking domains [3] provides an alternative to unconstrained clock skew scheduling. Fishburn [4] presented a polynomial time complexity algorithm to derive a multi-domain clock skew schedule for a target clock period. Ravindran, Keuhlmann, and Sentovich [5] also studied the clock period minimization problem using multiple clocking domains combined with small within-domain latency.

Vittal, Ha, Brewer and Marek-Sadowska [6] used the 0-1 integer linear programming (ILP) formulations to minimize the peak current under the constraint of multi-domain clock skew scheduling. Although their approach guarantees the optimality (in terms of the specified cost function), the run time of the 0-1 ILP solver grows dramatically with the increase of binary variables. The high computational expense has limited the use of their approach.

This paper presents an approach to overcoming the limitation. Our approach includes the following two aspects:

(1) We derive the ASAP (as-soon-as-possible) schedule and the ALAP (as-late-as-possible) schedule of each register. As a result, we can prune all the redundancies without sacrificing the exactness (optimality) of the solution.

(2) For a large circuit, the number of binary variables and the number of constraints are still very large even though all the redundancies are pruned. Thus, we propose a zone-based approach to solve a large circuit heuristically.

Note that the proposed algorithms are *independent* of the used peak current model. Therefore, in fact, our approach is a *general methodology* for the reduction of peak current under the constraint of multi-domain clock skew scheduling.

II. Preliminaries

Multi-domain clock skew scheduling only uses n discrete clocking domains: d_1, d_2, …and d_n. A clocking domain d_k, where k = 1, 2, … and n, corresponds to a clock arrival time. Without loss of generality, we assume that $d_1 \le d_2 \le \ldots \le d_n$.

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: $S_{i,1}$, $S_{i,2}$, …, $S_{i,n}$, where $S_{i,k}$ is 1 if and only if T_{Ci} is equal to d_k. 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:

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

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

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

The objective is to minimize the value *peak_current*. Without loss of generality and for the convenience of illustration, in the following, we assume that the total current derivative at the clocking domain d_k is $\sum_{i=1}^{r} S_{i,k} \cdot I_i$, where r denotes the number of registers and I_i denotes the maximum possible current of register R_i caused by the clock edge. As a

0-7803-9451-8/06/$20.00

result, for each clocking domain d_k, the peak current constraint is as below:

$$\sum_{i=1}^{r} S_{i,k} \cdot I_i \le peak\_current.$$

Thus, the peak current minimization problem formulated by [6] has $r+2s+n$ constraints, where s is the number of data paths.

Using Fig. 1 as an example, the circuit graph has 6 registers and 10 data paths. Suppose that clock period P = 6 tu (time units) and $I_1 = I_2 = I_3 = I_4 = I_5 = I_6 = c$. If the circuit is fully synchronous, a huge peak current $6*c$ is observed at the clock edge. Assume that we are given 3 clocking domains: $d_1 = -2$ tu, $d_2 = 0$ tu and $d_3 = 2$ tu. If we directly use the 0-1 ILP formulations as described in [6], the number of binary variables is 6*3 = 18 and the number of constraints is 6+2*10+3 = 29. After solving the 0-1 ILP formulations, we find that the peak current is reduced to $2*c$ under the multi-domain clock skew schedule in which $T_{C1} = d_1 = -2$ tu, $T_{C2} = d_2 = 0$ tu, $T_{C3} = d_1 = -2$ tu, $T_{C4} = d_3 = 2$ tu, $T_{C5} = d_2 = 0$ tu, and $T_{C6} = d_3 = 2$ tu.

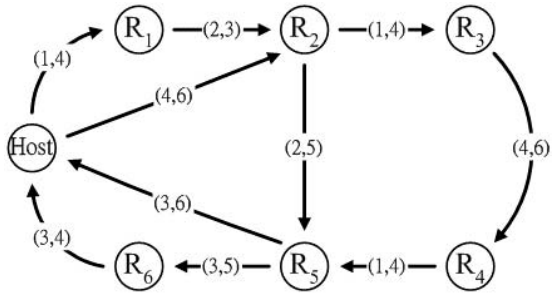

Fig. 1: Circuit graph example.

III. The Approach

Section 3.1 derives an ASAP schedule and an ALAP schedule to prune the redundancies. Section 3.2 presents a zone-based scheduling algorithm for large circuits.

A. ASAP and ALAP

In fact, due to the zero clocking constraints and the double clocking constraints, many binary variables used in the 0-1 ILP formulations are definitely to be 0. Therefore, a large fraction of the binary variables are redundant. Moreover, a large fraction of the constraints used in the 0-1 ILP 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 (optimality) 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; the function ceil(x) gives the earliest clocking domain after the clock arrival time x. The pseudo codes of the two functions are shown in Fig. 2 and Fig. 3, respectively, where *n* is the number of clocking domains.

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. Let $ASAP_i$ and $ALAP_i$ denote the allowable earliest clocking domain and the allowable latest clocking domain of register R_i, respectively. Thus, we say that $[ASAP_i,ALAP_i]$ is 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. Then, by iteratively applying the clocking constraints under the target clock period P, we 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). Fig. 4 gives the pseudo code

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

Fig. 2: Function floor.

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

Fig. 3: Function ceil.

```
Procedure OBTAIN_ASAP_ALAP(G,P)
begin
 let [ASAP_host,ALAP_host] be [0,0];
  for each register R_i do
   [ASAP_i,ALAP_i] = [d_1,d_n];
  repeat
    update = 0;
  for each data path R_i→R_j do
   begin
     apply clocking constraints in the forward direction to narrow
     the allowable clocking range [ASAP_j,ALAP_j];
     if [ASAP_j,ALAP_j] is narrowed then update = 1;
   end;
  for each data path R_i→R_j do
   begin
     apply clocking constraints in the backward direction to narrow
     the allowable clocking range [ASAP_i,ALAP_i];
     if [ASAP_i,ALAP_i] is narrowed then update = 1;
   end
  until (update == 0);
end.
```

Fig. 4: Procedure OBTAIN_ASAP_ALAP.

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). Note that the sequence of data paths in each for-loop, which applies the clocking constraints, can be arbitrarily specified without affecting the calculation of ASAP schedule and ALAP schedule. The details are described as below.

(1) If 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:

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

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

(2) If 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:

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

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

Let's use the circuit graph shown in Fig. 1 as an example to demonstrate the procedure OBTAIN_ASAP_ALAP. Suppose that clock period P = 6 tu, $I_1 = I_2 = I_3 = I_4 = I_5 = I_6 = c$, and we are given 3 clocking domains: $d_1 = -2$ tu, $d_2 = 0$ tu and $d_3 = 2$ tu. At the beginning, we have $[ASAP_i, ALAP_i] = [-2,2]$ for each register R_i where i = 1, 2, 3, 4, 5 and 6. Then, we enter the repeat-until-loop. We assume that the first for-loop (i.e., applying clocking constraints in forward direction) tackles the data paths in the sequence of host$\rightarrow R_1$, host$\rightarrow R_2$, $R_1 \rightarrow R_2$, $R_2 \rightarrow R_3$, $R_2 \rightarrow R_5$, $R_3 \rightarrow R_4$, $R_4 \rightarrow R_5$, and $R_5 \rightarrow R_6$, and the second for-loop (i.e., applying clocking constraints in backward direction) tackles the data paths in the sequence of $R_6 \rightarrow$host, $R_5 \rightarrow$host, $R_5 \rightarrow R_6$, $R_4 \rightarrow R_5$, $R_3 \rightarrow R_4$, $R_2 \rightarrow R_5$, $R_2 \rightarrow R_3$, and $R_1 \rightarrow R_2$. Table I gives the snapshot. 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.

Table I. The snapshots of procedure OBTAIN_ASAP_ALAP.

	First repeat-until-loop		Second repeat-until-loop	
	forward	backward	forward	backward
R_1	[-2,0]	[-2,0]	[-2,0]	[-2,0]
R_2	[0,2]	[0,0]	[0,0]	[0,0]
R_3	[-2,2]	[-2,0]	[-2,0]	[-2,0]
R_4	[-2,2]	[-2,2]	[-2,2]	[0,2]
R_5	[-2,2]	[-2,0]	[0,0]	[0,0]
R_6	[-2,2]	[-2,2]	[0,2]	[0,2]

After the procedure OBTAIN_ASAP_ALAP is completed, we have $[ASAP_1, ALAP_1] = [-2,0]$, $[ASAP_2, ALAP_2] = [0,0]$, $[ASAP_3, ALAP_3] = [-2,0]$, $[ASAP_4, ALAP_4] = [0,2]$, $[ASAP_5, ALAP_5] = [0,0]$ and $[ASAP_6, ALAP_6] = [0,2]$. Since $ASAP_2 = ALAP_2 = 0$ tu and $ASAP_5 = ALAP_5 = 0$ tu, we have $T_{C2} = d_2 = 0$ tu and $T_{C5} = d_2 = 0$ tu. Thus, we only need to use 8 binary variables: $S_{1,1}$, $S_{1,2}$, $S_{3,1}$, $S_{3,2}$, $S_{4,2}$, $S_{4,3}$, $S_{6,2}$ and $S_{6,3}$. Moreover, we can use both the ASAP schedule and the ALAP schedule to simplify the zero clocking constraints and the double clocking constraints. For example, the double clocking constraint of the data path $R_1 \rightarrow R_2$ can be simplified to $2*S_{1,1} \leq T_{PD1,2(min)} = 2$ and thus can be easily justified as a redundant constraint. The justification of redundant constraints is in polynomial time complexity. By exploiting both the ASAP schedule and the ALAP schedule, we find that all zero clocking constraints and all the double clocking constraints in this circuit graph become redundant. Consequently, the number of constraints is reduced from 29 to 7. As a result, we can rewrite the 0-1 ILP formulations as below:

minimize *peak_current*

subject to

$S_{1,1} + S_{1,2} = 1$;

$S_{3,1} + S_{3,2} = 1$;

$S_{4,2} + S_{4,3} = 1$;

$S_{6,2} + S_{6,3} = 1$;

$S_{1,1}*I_1 + S_{3,1}*I_3 \leq peak\_current$;

$S_{1,2}*I_1 + 1*I_2 + S_{3,2}*I_3 + S_{4,2}*I_4 + 1*I_5 + S_{6,2}*I_6 \leq peak\_current$;

$S_{4,3}*I_4 + S_{6,3}*I_6 \leq peak\_current$;

By applying the 0-1 ILP solver, we have the results that $S_{1,1} = 1$, $S_{1,2} = 0$, $S_{3,1} = 1$, $S_{3,2} = 0$, $S_{4,2} = 0$, $S_{4,3} = 1$, $S_{6,2} = 0$ and $S_{6,3} = 1$. In other words, the peak current is $2*c$ under the multi-domain clock skew schedule in which $T_{C1} = d_1$, $T_{C2} = d_2$, $T_{C3} = d_1$, $T_{C4} = d_3$, $T_{C5} = d_2$ and $T_{C6} = d_3$. Clearly, both the number of binary variables and the number of constraints are dramatically reduced without scarifying the exactness of the solution.

B. Zone-Based Scheduling

For a large circuit, the number of binary variables is still very large even though all the redundant binary variables are pruned. Given a circuit graph G and a target clock period P, Fig. 5 gives the procedure ZONE_BASED_SCHEDULING to deal with a large circuit. At the beginning, in addition to prune all the redundant binary variables, we also use the algorithm proposed in [4] to derive a feasible multi-domain clock skew schedule for the circuit graph G to work with clock period P. We let the feasible schedule be the initial solution. Note that the algorithm proposed in [4] does not attempt to minimize the peak current. The kernel of zone-based scheduling is an iteration process of the procedure ZONE_PARTITION_SCHEDULING.

The procedure ZONE_PARTITION_SCHEDULING is to reduce the peak current by re-scheduling the registers in the set RS. Fig. 6 gives the pseudo code. By specifying the constraint on the maximum number of binary variables in the ILP formulations, we partition the set of registers RS into some sub-sets. Each sub-set is called a zone. The philosophy of zone partitioning 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 very high. Therefore, in order to evenly spread the currents over the clocking domains, a reasonable heuristic is to first re-schedule the registers that are associated fewer binary variables. Based on this philosophy, a zone is created to accommodate as many registers as possible. The fewer associated binary variables, the higher priority (to be assigned into the zone) the register has. If the current zone cannot accommodate any more register (due to the limitation on the maximum number of binary variables), we create another new zone. The iteration of zone creation repeats until all the registers are assigned. Note that the earlier created zone has a higher priority to be re-scheduled. When a zone is re-scheduled, the clock arrival times of other registers are assumed to be fixed. Since the maximum number of binary variables within a zone is limited, the 0-1 ILP solver can re-schedule each zone efficiently.

```
Procedure ZONE_BASED_SCHEDULING(G,P)
begin
  prune all the redundant binary variables according to the results
  of the procedure OBTAIN_ASAP_ALAP(G,P);
  apply [4] to derive a feasible multi-domain clock skew schedule
  for the circuit graph G to work with clock period P;
  phase = 1;
  RS = all the registers in the circuit graph G;
  call ZONE_PARTITION_SCHEDULING(phase,RS, G,P);
  phase = 2;
  repeat
     RS=all the registers scheduled into hottest clocking domains;
     call ZONE_PARTITION_SCHEDULING(phase,RS,G,P);
  until (the peak current cannot be further reduce);
end.
```

Fig. 5: Procedure ZONE_BASED_SCHEDULING.

In fact, the zone-based scheduling uses two phases to apply the procedure ZONE_PARTITION_SCHEDULING as below.

(1) The first phase is to derive a multi-domain clock skew schedule that attempts to minimize the peak current. We use the procedure ZONE_PARTITION_SCHEDULING to re-schedule all the registers in the circuit graph G. When a zone is re-scheduled, our objective is to minimize the peak current produced by the registers that have been re-scheduled. In the pseudo code of Fig. 6, we use the notation RP to denote the set of all the registers that have been re-scheduled.

(2) The second phase attempts to further reduce the peak current. A clocking domain is called the hottest clocking domain, if and only if the total current derivative at this clocking domain is exactly the same as the peak current. The peak current cannot be further reduced, unless some registers scheduled in hottest clocking domains can be re-scheduled into other clocking domains. Thus, we iteratively apply the procedure ZONE_PARTITION_SCHEDULING to re-schedule the registers scheduled in hottest clocking domains. The iteration process repeats until the peak current cannot be further reduced. Note that, in the second phase, we let the set RP be all the registers in the circuit graph G.

```
Procedure ZONE_PARTITION_SCHEDULING(phase,RS,G,P)
begin
  partition all the registers in the set RS into zones (under the
  constraint on the maximum number of binary variables);
  if (phase == 1) then RP = ∅
  else RP = all the registers in the circuit graph G;
  for each zone do (in the sequence of their priorities)
  begin
    if (phase == 1) then add all the registers in this zone into RP;
  re-schedule this zone under all the clocking constraints to reduce
  the peak current produced by the registers in the set RP;
  end
end.
```

Fig. 6: Procedure ZONE_PARTITION_SCHEDULING.

Let's use Fig. 1 as an example. First, we derive both the ASAP schedule and ALAP schedule as shown in Table I. Also, 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 $T_{C6} = d_2$ to work with clock period 6 tu as the initial solution. Therefore, initially, the peak current is 6*c. Suppose that the maximum number of binary variables involved in a zone is only 4. The process of zone-based scheduling is as below.

First, we enter the first phase of zone-based scheduling. The first phase re-schedules all the registers in the circuit graph. From both the ASAP schedule and the ALAP schedule, we know that R_1, R_2, R_3, R_4, R_5 and R_6 are associated with 2, 0, 2, 2, 0 and 2 binary variables. Thus, all the registers in the circuit graph are partitioned into two zones: zone $Z_1 = \{R_1, R_2, R_3, R_5\}$ and zone $Z_2 = \{R_4, R_6\}$, and zone Z_1 has a higher priority to be re-scheduled. In the following, we reduce the peak current in the sequence of zone Z_1 and zone Z_2.

We reduce the peak current by re-scheduling zone Z_1 under $T_{C4} = d_2$ and $T_{C6} = d_2$. The 0-1 ILP formulations are as below:
minimize *peak_current_*Z_1

subject to
$S_{1,1} + S_{1,2} = 1$;
$S_{3,1} + S_{3,2} = 1$;
$S_{1,1}*I_1+S_{3,1}*I_3 \le peak\_current\_Z_1$;
$S_{1,2}*I_1+1*I_2+S_{3,2}*I_3+1*I_5 \le peak\_current\_Z_1$;

We have the results that $peak\_current\_Z_1 = 2*c$, $S_{1,1} = 1$, $S_{1,2} = 0$, $S_{3,1} = 1$ and $S_{3,2} = 0$. As a result, we have $T_{C1} = d_1$, $T_{C2} = d_2$, $T_{C3} = d_1$, $T_{C4} = d_2$, $T_{C5} = d_2$, $T_{C6} = d_2$ and the peak current is 4*c. Thus, the peak current is reduced from 6*c to 4*c.

We further reduce the peak current by re-scheduling zone Z_2 under $T_{C1} = d_1$, $T_{C2} = d_2$, $T_{C3} = d_1$ and $T_{C5} = d_2$. The 0-1 ILP formulations are as below:
minimize $peak\_current\_Z_1\_Z_2$
subject to
$S_{4,2} + S_{4,3} = 1$;
$S_{6,2} + S_{6,3} = 1$;
$1*I_1+1*I_3 \le peak\_current\_Z_1\_Z_2$;
$1*I_2+S_{4,2}*I_4+1*I_5+S_{6,2}*I_6 \le peak\_current\_Z_1\_Z_2$;
$S_{4,3}*I_4+S_{6,3}*I_6 \le peak\_current\_Z_1\_Z_2$;

We have the results that $peak\_current\_Z_1\_Z_2 = 2*c$, $S_{4,2} = 0$, $S_{4,3} = 1$, $S_{6,2} = 0$ and $S_{6,3} = 1$. Thus, the peak current is reduced from 4*c to 2*c.

Next, we enter the second phase of zone-based scheduling. The peak current is 2*c. Since the total current derivative at clocking domains d_1, d_2 and d_3 are 2*c, 2*c, 2*c, respectively, they are hottest clocking domains. We find that R_1 and R_3 are scheduled to d_1, R_2 and R_5 are scheduled to d_2, and R_4 and R_6 are scheduled to d_3. Thus, we partition these six registers into two zones: zone $Z_3 = \{R_1, R_2, R_3, R_5\}$ and zone $Z_4 = \{R_4, R_6\}$, and zone Z_3 has a higher priority to be re-scheduled.

We re-schedule zone Z_3 under $T_{C4} = d_3$ and $T_{C6} = d_3$. The 0-1 ILP formulations are as below:
minimize *peak_current*
subject to
$S_{1,1} + S_{1,2} = 1$;
$S_{3,1} + S_{3,2} = 1$;
$S_{1,1}*I_1+S_{3,1}*I_3 \le peak\_current$;
$S_{1,2}*I_1+1*I_2+S_{3,2}*I_3+1*I_5 \le peak\_current$;
$1*I_4+1*I_6 \le peak\_current$;

We have the results that $peak\_current = 2*c$, $S_{1,1} = 1$, $S_{1,2} = 0$, $S_{3,1} = 1$ and $S_{3,2} = 0$. The peak current is still 2*c.

We re-schedule zone Z_4 under $T_{C1} = d_1$, $T_{C2} = d_2$, $T_{C3} = d_1$ and $T_{C5} = d_2$. The 0-1 ILP formulations are as below:
minimize *peak_current*
subject to
$S_{4,2} + S_{4,3} = 1$;
$S_{6,2} + S_{6,3} = 1$;
$1*I_1+1*I_3 \le peak\_current$;
$1*I_2+S_{4,2}*I_4+1*I_5+S_{6,2}*I_6 \le peak\_current$;
$S_{4,3}*I_4+S_{6,3}*I_6 \le peak\_current$;

The second phase of zone-based scheduling is finished with the results that $peak\_current = 2*c$, $S_{4,2} = 0$, $S_{4,3} = 1$, $S_{6,2} = 0$ and $S_{6,3} = 1$. After the zone-based scheduling is completed, the peak current is 2*c under the multi-domain clock skew schedule in which $T_{C1} = d_1$, $T_{C2} = d_2$, $T_{C3} = d_1$, $T_{C4} = d_3$, $T_{C5} = d_2$ and $T_{C6} = d_3$.

IV. Experimental Results

Our approach has been implemented in a C++ program running on a personal computer with AMD K8-3GHz CPU and 1G Bytes RAM. We use Extended LINGO Release 8.0 as the 0-1 ILP solver. The circuits from ISCAS'89 benchmark suites are targeted to a 0.35μm cell library for the experiments. Without loss of generality, in each benchmark circuit, we assume that: (1) the clock period P is the longest path delay; (2) we are given $2\times\left\lfloor \frac{P}{0.3} \right\rfloor + 1$ clocking domains, where 0.3 ns denotes the clock skew between two consecutive clocking domains; (3) each clocking domain $d_k = 0.3\times(k-\left\lfloor \frac{P}{0.3} \right\rfloor - 1)$ ns, where k = 1, …, $2\times\left\lfloor \frac{P}{0.3} \right\rfloor + 1$; (4) the maximum possible current of each register is *c* (at the clock arrival time); and (5) if the circuit is fully synchronous, the peak current occurs when all the registers are switching simultaneously.

Table II gives the characteristics of benchmark circuits. The column *zero skew* denotes the peak current of fully synchronous implementation, while the column [6] denotes the peak current solved by the 0-1 ILP formulations as described in [6]. We use the notation -- to denote that the problem complexity cannot be solved within 12 hours.

Table II. Characteristics of benchmark circuits.

circuit	registers	gates	data paths	clock period	peak currrent	
					zero skew	[6]
S444	21	119	175	4.24	21*c	2*c
S499	22	120	528	3.99	22*c	6*c
S1269	37	437	455	9.89	37*c	4*c
S1512	57	413	686	5.50	57*c	6*c
S3271	116	1035	1137	8.50	116*c	4*c
S3384	183	1070	1831	19.86	183*c	--
S5378	179	1001	2313	5.16	179*c	11*c
S6669	239	2155	2179	28.42	239*c	--
S9234	228	2680	2830	9.42	228*c	--
S13207	669	2573	4660	10.92	669*c	--
S15850	597	3448	16863	14.23	597*c	--
S35932	1728	12204	5159	5.14	1728*c	288*c
S38584	1452	11448	15329	11.85	1452*c	--

Table III demonstrates the advantage of exploiting both the ASAP schedule and the ALAP schedule to prune the

redundancies. Note that, for each benchmark circuit, both the ASAP schedule and the ALAP scheduled can be derived within 1 second. The column [6] describes the original 0-1 ILP formulations. The column *Reduced formulations* describes the reduced 0-1 ILP formulations, in which all the redundancies are pruned. The column *#vars* denotes the number of binary variables, the column *#cons* denotes the number of constraints, and the column *CPU time* gives the CPU time in seconds.

Table III. The advantage of pruning redundancies.

Circuit	[6]			**Reduced Formulations**		
	#vars	#cons	CPU time (s)	#vars	#cons	CPU time (s)
S444	609	400	4	239	355	1
S499	594	1105	6	80	524	1
S1269	2405	1012	9	312	411	3
S1512	2109	1466	5	326	825	1
S3271	6612	2447	30087	3342	2027	88
S3384	24339	4316	--	4917	3531	24
S5378	6265	4860	16046	2088	342	59
S6669	45171	5476	--	3951	3265	122
S9234	14364	6803	--	6554	6042	232
S13207	48837	10062	--	20057	8416	362
S15850	56715	34418	--	18124	28314	--
S35932	60480	12081	714	17516	3714	271
S38584	114708	32189	--	40018	29569	--

Table IV 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. Without loss of generality, here we use the fully synchronous solution as the initial solution. The column *tackled zones* denotes the number of times to use the 0-1 ILP solver to re-schedule a zone. The CPU time includes the time spent in both our C++ program and the 0-1 ILP solver. For the convenience of comparisons, we also report the results obtained by *whole-circuit scheduling*, which solves the whole circuit as a single zone. We find that the whole-circuit scheduling cannot solve benchmark circuits S15850 and S38584 within 12 hours. However, on the other hand, the zone-based scheduling can solve each benchmark circuit within 712 seconds no matter the constraint on the maximum number of binary variables involved in a zone is 500, 1000 or 2000.

V. Conclusions

This paper investigates a fast multi-domain clock skew scheduling for peak current reduction. The main contribution of our work is that it overcomes the high computational expense of previous work. Benchmark data consistently show that our approach achieves very good results within an acceptable run time.

Acknowledgements

This work was supported in part by the National Science Council of R.O.C. under grant number of NSC 93-2215-E-033-004.

References

[1] J.P. Fishburn, “Clock Skew Optimization”, IEEE Trans. on Computers, Vol. 39, No. 7, pp. 945—951, 1990.

[2] L. Benini, P. Vuillod, A. Bogliolo, and G. De Micheli, “Clock Skew Optimization for Peak Current Reduction”, Journal of VLSI Signal Processing, vol. 16, pp. 117—130, 1997.

[3] A. Vittal and M. Marek-Sadowska, “Power-Optimal Buffered Clock Tree Design”, in the Proc. of IEEE/ACM Design Automation Conference, pp. 497—502, 1995.

[4] J.P. Fishburn, “Solving a System of Difference Constraints with Variables Restricted to a Finite Set”, Information Processing Letters, vol. 82, no. 3, pp. 143—144, 2002.

[5] K. Ravindran, A, Keuhlmann, and E. Sentovich, “Multi-Domain Clock Skew Scheduling”, in the Proc. of IEEE/ACM International Conference on Computer Aided Design, pp. 801—808, 2003.

[6] A. Vittal, H. Ha, F. Brewer, and M. Marek-Sadowska, “Clock Skew Optimization for Ground Bounce Control”, in the Proc. of IEEE/ACM International Conference on Computer Aided Design, pp. 395—399, 1996.

Table IV. The results of zone-based scheduling.

circuit	500 variables in a zone			1000 variables in a zone			2000 variables in a zone			whole-circuit	
	peak current	tackled zones	CPU time (s)	peak current	tackled zones	CPU Time (s)	peak current	tackled zones	CPU time (s)	peak current	CPU time (s)
S3271	4*c	8	10	4*c	6	14	4*c	4	88	4*c	88
S3384	5*c	10	5	5*c	6	8	5*c	4	14	5*c	24
S5378	11*c	7	7	11*c	5	28	11*c	2	54	11*c	59
S6669	11*c	10	11	11*c	6	20	11*c	5	26	11*c	122
S9234	7*c	8	23	7*c	7	25	7*c	5	37	7*c	232
S13207	16*c	28	205	16*c	13	238	16*c	11	285	16*c	362
S15850	15*c	29	501	15*c	19	557	15*c	12	712	--	--
S35932	288*c	35	133	288*c	18	169	288*c	9	240	288*c	271
S38584	38*c	54	334	37*c	32	420	36*c	21	513	--	--