

Optimal Multi-Domain Clock Skew Scheduling

Li Li, Yinghai Lu, and Hai Zhou

Department of Electrical Engineering and Computer Science
Northwestern University

Abstract—Clock skew scheduling is an effective technique to improve the performance of sequential circuits. However, with process variations, it becomes more difficult to implement a large number of clock delays in a precise manner. Multi-domain clock skew scheduling is one way to overcome this limitation. In this paper, we prove the NP-completeness of multi-domain clock scheduling problem, and design a practical optimal algorithm to solve it. Given the domain number, we bound the number of all possible skew assignments and develop an optimal algorithm with efficient pruning techniques. Experiment results on ISCAS89 sequential benchmarks show the optimality and efficiency of our method compared with existing approaches.

Categories and Subject Descriptors: J.6 [Computer-Aided Engineering]: Computer-aided design (CAD)

General Terms: Algorithms

Keywords: Multi-Domain, Clock Skew, Optimization

I. INTRODUCTION

In a sequential circuit, clock signals are distributed to the memory elements such as latches or registers by clock tree or network. The clock arrival time to a register is referred to as its clock latency. The difference of clock latencies between registers is called clock skew. Clock skew scheduling [1], [2], [3] assigns different clock latencies to the registers of a sequential circuit to minimize the cycle period, by borrowing time from the paths with slacks and applying it to the critical paths. However, such an unconstrained clock skew schedule with a large set of arbitrary delays cannot be realized reliably. Due to process variations, it is difficult to implement clock schedule with a large set of arbitrary delays. In practice, it is desirable that the clock skew scheduling is constrained to a limited number of skew domains.

Multi-domain clock skew scheduling (MDCSS) was first proposed by Ravindran et al. [4] to meet this practical design requirement. Instead of assigning arbitrary clock latencies, multi-domain skew scheduling restricts the feasible clock latencies to a small set, called clock domains. In [4], the authors formulated the multi-domain clock skew scheduling problem as a mixed integer programming problem and solved it by a SAT-based algorithm, in which a CNF-based SAT solver is used to enumerate the assignment of registers to clock skew domains based on their encoding in boolean variables. Despite its optimality, the SAT-based algorithm incurs a penalty of high computational cost.

Since the pioneer work of [4], other researchers have been looking into the MDCSS problem. Zhi et al. [5] proposed an exact algorithm for the same problem based on branch-and-bound search framework with greedy speeding up heuristics. In [6], a multi-level clustering

algorithm was proposed to solve the same problem. The algorithm recursively merges half of the registers at each level until the total number of clusters is small enough. Compared to the work of [4], this algorithm is much faster. However, the algorithm is based on heuristics and thus cannot guarantee the optimality of solution. For example, their algorithm results in a 7% performance degradation with respect to the optimal solution on a circuit with only 23 registers [6]. In [7], a special mixed-integer linear program was formulated and solved to give an initial register clustering, which was then improved by iterative refinement. In [8], the authors solved a slightly different problem, which is to minimize the number of domains for the optimal clock period. While the clock period is optimized, their method may introduce excessive number of clock domains, which will result in high cost on implementing complex clock trees [4], [9] and will compromise the overall quality of the design. Huang et al. [10] and Lin et al. [11] also proposed MDCSS techniques considering peak current and delay padding.

Despite the previous efforts to solve the MDCSS problem, there are still many open questions in this topic. The existing optimal algorithms solved the problem by casting it into a mixed integer linear program, which in general is intractable. However, the complexity of the problem has never been answered by any existing work. Is the problem NP-hard or is there a polynomial-time algorithm? In this paper, we show that the MDCSS problem is NP-complete if the domain number is large. But it is polynomially solvable if the domain number is a small constant. Since the sole reason for MDCSS is to restrict domain values to a small set [4], [6] to alleviate the implementation of complex clock networks [9], we can use this fact to design an efficient optimal algorithm for the problem. The contribution of this paper includes:

- We carry out a thorough investigation on the hardness of MDCSS problem, proving it to be NP-complete when number of domains is large.
- We discover that MDCSS problem is tractable with number of domains being a small constant, which provides guidance to the design of an optimal algorithm.
- Based on the theoretical analysis, we design an optimal algorithm with efficient pruning techniques for MDCSS with small number of domains. The resulting algorithm is very efficient, finishing all the ISCAS89 test cases within 4 seconds.

The rest of the paper is organized as follows. In Section II, the multi-domain clock skew scheduling problem is formulated formally. We show in Section III that the MDCSS problem is NP-complete when the number of domains is large and in Section IV that it is polynomially solvable when the number is small constant. The proposed algorithm is described in Section V with highly effective pruning techniques. Experimental results are presented and discussed in Section VI. The paper is concluded in Section VII.

II. PROBLEM FORMULATION

A sequential circuit can be modeled as a directed graph $G(V, E)$, where each vertex $v \in V$ represents a register, with its clock latency

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

DAC'11, June 5-10, 2011, San Diego, California, USA

Copyright © 2011 ACM 978-1-4503-0636-2/11/06...\$10.00

$l(v)$ labeled. In addition, a virtual vertex v_0 is added to represent the primary inputs and outputs. Between every pair of vertices u and v that are physically connected by combinational paths, there are a forward edge and a backward edge representing the setup and hold constraints respectively. Given a clock period T , the setup time constraints on forward edge $(u, v) \in E_f$ ensures that the signal from u to v has enough time to stabilize its value before v stores it:

$$l(u) + D(u, v) \leq T + l(v) - d_{setup}(v), \quad (1)$$

where $D(u, v)$ is the maximum combinational delay from u to v , and $d_{setup}(v)$ is the setup time for register v . On the other hand, the hold time constraint on backward edge $(v, u) \in E_b$ asserts that the signal from u does not overwrite the previous one before v stores it:

$$l(u) + d(u, v) \geq l(v) + d_{hold}(v), \quad (2)$$

where $d(u, v)$ is the minimum combinational delay from u to v , and $d_{hold}(v)$ is the hold time for register v . The sets of forward and backward edges constitute the edge set of G , that is, $E = E_f \cup E_b$. Figure 1 gives a simple example of a sequential circuit adapted from [4] and its constraint graph, assuming zero setup and hold times for all registers. In Figure 1(b), solid edges represent setup constraints

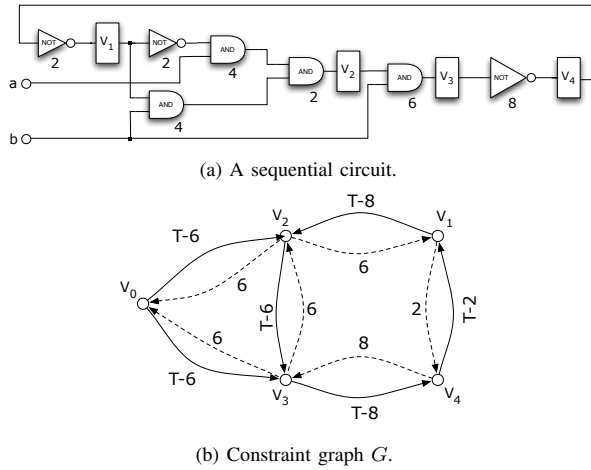


Fig. 1. An example of sequential circuit and its constraint graph G .

while dashed edges represent hold constraints.

Given the number k of clock domains, the objective of multi-domain clock skew scheduling (MDCSS) is to decide the k domain values $\{l_1, l_2, \dots, l_k\}$ as well as to assign each register one domain value such that the clock period T is minimized while the setup and hold constraints given in the above inequalities are satisfied. Since only the differences between the clock latencies matter, without loss of generality, we assume $l_1 = 0$ in the rest of the paper. The MDCSS problem can be formally formulated as follows:

$$\begin{aligned} \min \quad & T \\ \text{s.t.} \quad & l(u) \leq l(v) + w(u, v), \quad \forall (u, v) \in E \\ & l(u) \in \{l_1 = 0, l_2, \dots, l_k\}, \quad \forall u \in V \\ & l_i \in [0, T) \end{aligned} \quad (3)$$

where

$$w(u, v) = \begin{cases} T - D(u, v) - d_{setup}(v), & (u, v) \in E_f \\ d(v, u) - d_{hold}(u), & (u, v) \in E_b \end{cases} \quad (4)$$

III. NP-COMPLETENESS FOR MANY DOMAINS

As the multi-domain clock scheduling problem is formally stated by (3), we study the open problem of its hardness in this section. First, we convert the optimization problem into a decision problem, which can be stated as follows:

Problem 1: Given a clock period T , does there exist a feasible multi-domain clock skew schedule within k domains?

If this decision problem can be solved in polynomial time, due to the monotonicity of feasible T in the MDCSS problem, we can carry out a binary search and solve the optimization problem efficiently with $O(\log(T))$ calls to the decision procedure.

We find out that the complexity of the decision problem relies highly on the domain number k . Interestingly, when $k \geq |V|$, the MDCSS problem becomes the unconstrained clock skew scheduling problem [1]. On the other hand, when k is constant, the problem is also polynomially solvable as shown in the next section. However, when k is smaller than $|V|$ but large and comparable to $|V|$, the problem is NP-complete.

Theorem 1: Problem 1 is NP-complete when $k = |V|/2$.

Proof: A solution of k -domain clock skew scheduling can be verified in $O(|E|)$ time by checking each of the difference inequalities in (3). Thus Problem 1 is in NP.

Next, we prove its NP-completeness by a reduction from *Equal Subset Sum* which has been proved to be NP-complete [12].

Problem 2 (Equal Subset Sum): A finite set S of positive integers, do there exist two disjoint nonempty subsets S_1 and S_2 of S whose sums are equal to each other?

Let $S = \{a_1, \dots, a_n\}$ be a set of positive integers. We construct a directed graph G_S with $2n$ vertices $\{u_1, \dots, u_n, v_1, \dots, v_n\}$ by connecting u_i and v_i in both directions for each i . Then we assign the weight a_i to the forward edge (u_i, v_i) and $-a_i$ to the backward edge (v_i, u_i) . The constructed G_S is shown in Figure 2(a). Note that a domain assignment is feasible if and only if $l(v_i) = l(u_i) + a_i$ for each i . Based on the constructed instance G_S , we prove the following claim: G_S has a feasible assignment with at most n distinct domains if and only if there exist two disjoint nonempty subsets S_1 and S_2 of S whose sums are equal to each other.

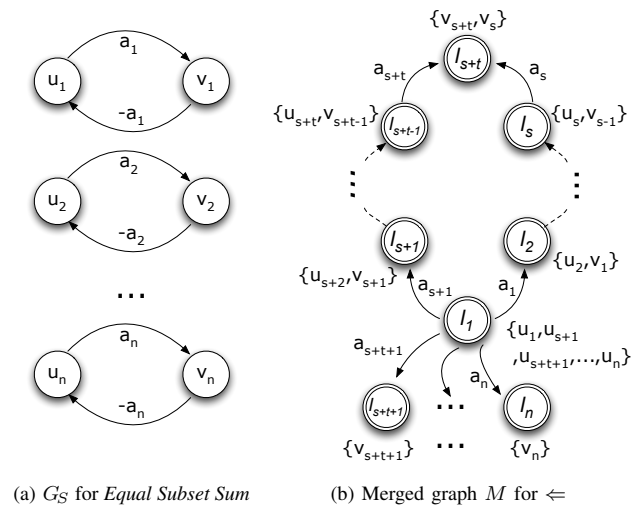


Fig. 2. Graph construction for *Equal Subset Sum* and the merged graph.

$\Rightarrow$: Given a feasible assignment with at most n distinct domains, we build a merged graph M from G_S by merging all vertices with the same domain in G_S while removing all the negative edges. Figure 2(b) gives an example of merged graph. Since the resulting M has at most n vertices and exactly n edges, it contains a cycle C ignoring the direction of edges. (Notice that an acyclic graph with n vertices have at most $n - 1$ edges). Let S_1 be the set of weights of the edges appearing in C in one direction, and S_2 be the set of weights of the edges appearing in C in the other direction. It is easy to see that the sum of S_1 is equal to the that of S_2 .

$\Leftarrow$: Let S_1 and S_2 be the subsets of S satisfying the condition. Without loss of generality, assume that $S_1 = \{a_1, a_2, \dots, a_s\}$ and $S_2 = \{a_{s+1}, a_{s+2}, \dots, a_{s+t}\}$. It can be verified that the following domain assignment of G_S is valid:

$$\begin{aligned} l(u_1) &= l(u_{s+1}) = 0 \\ l(u_i) &= l(v_{i-1}), & \forall 2 \leq i \leq s \vee s+2 \leq i \leq s+t \\ l(u_i) &= 0, & \forall s+t+1 \leq i \leq n \\ l(v_i) &= l(u_i) + a_i, & \forall 1 \leq i \leq n \end{aligned}$$

Figure 2(b) shows the resulting merged graph M corresponding to the above assignment. The number of nodes in M is n , indicating a valid n -domain assignment. ■

IV. EASINESS FOR FEW DOMAINS

Although the MDCSS problem is NP-complete when the domain number is $|V|/2$, the problem is not that hard when the domain number is small, which is the normal case, for exactly the same reason of introducing MDCSS (that is, it is exponentially hard to precisely implement many domains) [4], [6]. In this section, we show that the MDCSS problem is polynomially solvable if the domain number is a constant. This provides the basis for our efficient algorithm in the next section.

Since restricting the clock latencies to k domains does not change the monotonicity of feasibility on clock period T , we can find the optimal clock period by a binary search on T with iteratively solving Problem 1. As the final solution of our problem, a feasible domain distribution is composed of a feasible domain assignment and a set of domain values. We define a domain assignment to be a partitioning of $v \in V$ to the k domains. A feasible domain assignment is the one that satisfies all the difference inequality constraints in (3) under the given T with properly chosen domain values. Problem 1 can be split into two parts. First, we need to generate proper domain candidates $\{l_1, l_2, \dots, l_k\}_{i=1}^M$. Next, for each domain candidate $\{l_1, l_2, \dots, l_k\}_i$, we check whether there is a valid clock skew assignment under current period T .

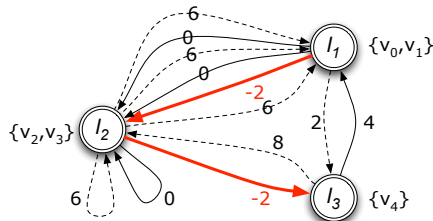


Fig. 3. An example of 3-domain-graph with $T = 6$. Notice that red edges constitute the shortest path tree of the given M_3 .

In order to better illustrate the effect of constraining clock latency $l(u)$ to k domains, we introduce a k -domain graph M_k , which consists of k vertices $\{s_1, s_2, \dots, s_k\}$ that represent the distinct k domains. M_k is built by merging the nodes with the same latency

l_i in G into a single node s_i , $i = 1, \dots, k$, while maintaining all the edges. Figure 3 gives an example of a 3-domain graph of G from Figure 1(b) with $T = 6$. Note that M_3 in Figure 3 corresponds to one possible domain assignment, which is annotated beside the nodes. According to different domain assignments, the domain graph will vary.

For a clock period T , M_k with a feasible domain assignment must have no negative cycles. On the other hand, if the domain assignment is feasible, we have the following lemma:

Lemma 1: For a clock period T and a feasible domain assignment, a feasible set of domain values can be decided from the shortest path tree in M_k .

Proof: Since M_k has no negative cycles, there exists a shortest path tree of M_k rooted at s_1 with $l_1 = 0$. A feasible set of domain values can be obtained by computing the shortest distance from s_1 to all other s_i 's. Suppose the shortest distance from s_1 to s_i is p_i , then a feasible set of domain values is $\{l_i = \max(0, -p_i)\}$. ■

The above lemma is very useful because its contrapositive tells us that if a set of domain value $\{l_1, l_2, \dots, l_k\}$ derived from the shortest path tree of M_k is infeasible, the corresponding domain assignment is also infeasible.

Next, we show that with a given T , although the possible number domain assignment is prohibitively huge, the number of distinct shortest path trees are much smaller when $k \ll |V|$.

Lemma 2: Given a clock period T , there are at most $O((k-1)!|E|^{k-1})$ distinct shortest path trees of k -domain graphs for all possible $k^{|V|}$ domain assignments.

Proof: It is obvious that the number of distinct M_k is $k^{|V|}$, which is the number of ways to partition all $|V|$ nodes in G_T into k domains. We now demonstrate that the number of underlying shortest path trees of M_k can be much smaller through the process of recursively constructing an arbitrary shortest path tree of M_k . We start from zero domain s_1 . At step j , we pick one domain s from the j domain nodes in the current shortest path tree and grow an edge from it to a newly selected domain t . The constructing process terminates after $(k-1)$ steps. The number of possible choices in step j is at most $j(|E| - j + 1)$. So the total number of distinct shortest path trees should be $M \leq \prod_{j=1}^{k-1} j(|E| - j + 1) = O((k-1)!|E|^{k-1})$. ■

Although there may be infinite number of potential domain candidates, the $O((k-1)!|E|^{k-1})$ candidates generated by corresponding shortest path trees are all that we need to check. Because these candidates have covered all $k^{|V|}$ possible assignments, if none of them is feasible, then no assignment can be feasible under the given T . Furthermore, note that Lemma 2 only gives an upper bound of the number of domain candidates. In practical circuits, the actual number of distinct shortest path trees and thus that of the domain candidates are mostly far smaller than the given upper bound.

Finally, we discuss the feasibility check problem. With a fixed domain candidate $\{l_1, l_2, \dots, l_k\}$, the domain feasibility check problem is actually a system of difference inequalities with variable values restricted to the given set, which can be solved with modified Bellman-Ford algorithm in $O(|V||E|)$ time [13]. Upon finish, this algorithm also assigns domain value to each node $v \in V$.

Summarizing the above discussion, we arrive at the following conclusion:

Theorem 2: Given a clock period T , whether there exists a feasible k -domain clock skew schedule can be decided in $O((k-1)!|V||E|^k)$ time. The optimal clock period under k domains can be decided in $O((k-1)!|V||E|^k \log T)$ time.

V. PRACTICALLY EFFICIENT ALGORITHM

Theorem 2 provides a basis to build an optimal algorithm by incrementally growing all distinct shortest path trees on M_k and checking its feasibility. As is discussed above and will be evident in the experiments, in real circuits, the number of distinct shortest path trees is far less than the $O((k-1)!|E|^{(k-1)})$ bound, which is already a significant reduction from $k^{|V|}$ by the effort from last section.

Admittedly, the raw algorithm developed straightforwardly from Theorem 2 may still suffer from high computational cost. To further improve the performance of the optimal algorithm, we introduce two efficient pruning techniques. First, we develop a necessary condition for clock feasibility with given T . If the necessary condition is not satisfied, none of the domain assignments for T is feasible and thus can be pruned. Secondly, if the necessary condition is satisfied, we identify false domain candidate by examining the weights of the edges on the corresponding shortest path tree. Such candidates can also be pruned to save the effort of feasibility check. We will elaborate the two pruning techniques as follows.

A. Pruning Based on Ordering Graph

With clock period T , we construct an *ordering graph* $O_T = (V, E_O)$ for G , in which each vertex represents a register and edge (u, v) indicates that $p(u, v) < 0$, where $p(u, v)$ is the shortest path distance from u to v in G . Figure 4 gives an example of ordering graph for G in Figure 1(b) with $T = 6$, where negative $p(u, v)$'s are annotated on the edges and $tp(u)$ is the topological depth of vertex u in O_T . Since O_T represents the ordering sequence of clock latency

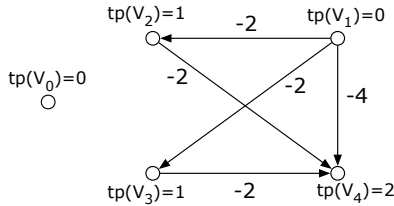


Fig. 4. Example of Ordering Graph for Figure 1(b) with $T = 6$.

among all the vertices, it is very useful for analyzing the domain dependencies between different vertices in G .

Lemma 3: $\forall (u, v) \in E_O$, suppose $p(u, v)$ is the shortest path distance from vertex u to vertex v in G ; $l(u)$ and $l(v)$ are the clock latencies of u and v in a feasible k domain distribution under clock period T . Then $l(u) < l(v)$ if $p(u, v) < 0$.

Proof: Suppose the shortest path from u to v is $\{u = t_0, t_1, \dots, t_{p-1}, t_p = v\}$. Because $l(t_{i-1}) \leq l(t_i) + w(t_{i-1}, t_i)$ and $p(u, v) = \sum_{i=1}^p w(t_{i-1}, t_i)$ for $i = 1, \dots, p$, we have $l(u) \leq l(v) + p(u, v)$. So if $p(u, v) < 0$, then $l(u) < l(v)$. ■

Let $K_T = \max_{u \in V} tp(u)$. With Lemma 3, it is not hard to find that K_T is the lower bound of the domain number needed under a given clock period T . So we obtain the following necessary condition for the existence of a k -domain solution under T :

Theorem 3: There exists a k -domain feasible distribution only if $K_T \leq k$.

By checking the satisfiability of Theorem 3, we can prune all the domain candidates under T when $K_T > k$. This pruning technique will greatly improve the performance of our algorithm.

However, naïve construction of O_T requires the knowledge of all-pair shortest paths, which will introduce great computational overhead. Notice that we are actually interested in the negativity of

Algorithm 1 GetCriticalClockMatrix($G = (V, E)$)

```

1: for each  $u, v \in V$  do
2:    $C_{uv} = T_{\text{opt}}$ ;  $bp(u, v) = 0$ ;
3: end for
4: for each vertex  $u \in V$  do
5:   for each vertex  $v \in V$  do
6:     Initialize  $p_{\text{ref}}(u, v)$  and  $t(u, v)$ ;
7:   end for
8:   Initialize NODE-KEY, ARC-KEY, PRIORITY-Q;
9:   while true do
10:     $\langle r, (s, v) \rangle = \text{PRIORITY-Q-FIND-MIN}$ ;
11:    if  $r = \infty \vee s \in \text{subtree}(v)$  then
12:      for each vertex  $v \in V$  do
13:        if  $p_{\text{ref}}(u, v) - r \cdot t(u, v) \leq 0 \wedge p_{\text{ref}}(u, v) - t(u, v) \cdot bp(u, v) > 0$  then
14:           $C_{uv} = T_{\text{init}} - p_{\text{ref}}(u, v)/t(u, v)$ ;
15:        end if
16:      end for
17:    end if
18:    for each vertex  $x \in \text{subtree}(v)$  do
19:      if  $p_{\text{ref}}(u, x) - r \cdot t(u, x) \leq 0$  then
20:         $C_{ux} = T_{\text{init}} - p_{\text{ref}}(u, x)/t(u, x)$ ;
21:         $bp(u, v) = r$ ;
22:      end if
23:      Update  $p_{\text{ref}}(u, v), t(u, v)$ ;
24:    end for
25:    for each vertex  $x \in \text{subtree}(v)$  do
26:      Update ARC-KEY and NODE-KEY of all edges entering or leaving  $x$ ;
27:    end for
28:  end while
29: end for

```

$p(u, v)$ rather than its specific value. If we know the critical clock period C_{uv} such that $p(u, v)|_{T=C_{uv}} = 0$, we can avoid repeated computation of all-pair shortest paths during each binary search iteration on T . We define a $|V| \times |V|$ *critical-clock matrix* C as the collection of all such critical clock periods C_{uv} . Each element C_{uv} satisfies the following condition:

$$T_{\text{opt}} \leq C_{uv} \leq T_{\text{init}}, \quad \forall u, v \in V, \quad (5)$$

where T_{opt} is the optimal clock period under unconstrained skew scheduling and T_{init} is the maximum path delay in G . T_{opt} and T_{init} constitute the lower bound and upper bound of T . Given a clock period T and C , O_T can be obtained by topological sort of G in $O(|V| + |E|)$ time. During this process, edge (u, v) is added to O_T if $C_{uv} > T$.

Next, we focus on how to compute the critical-clock matrix C . We propose an efficient algorithm to do so by leveraging Young's approach [14], which is one of the most efficient algorithms for min-cycle ratio problem [15]. The basic idea behind Young's algorithm (YTO) is to incrementally decrease the clock period T starting from T_{init} , while dynamically maintaining the corresponding shortest path trees. The algorithm terminates when a zero loop is detected. Given a clock period T , the shortest path distance we are interested in can be obtained in YTO as $p(u, v) = p_{\text{ref}}(u, v) - r \cdot t(u, v)$, where $p_{\text{ref}}(u, v)$ is the reference distance from u to v under T_{init} , $t(u, v)$ is the number of setup time edges on the shortest path from u to v under T and $r = T_{\text{init}} - T$. Consequently, our algorithm for computing the critical-clock matrix C can be embedded in YTO and is given in Algorithm 1. The whole flow and rest of the notations follow YTO in [15] with two modifications.

1) Since we are interested in the shortest distance from all nodes to

each other, instead of introducing a dummy root and keeping a global shortest path tree, each node is used in turn as the root of shortest path tree (Line 4). Single-source shortest path algorithm is used to get the initial shortest path information (Line 6);

- 2) In each iteration, we monitor vertex v whose p_{ref} and t has changed and check whether there is a change of sign of $p_{\text{ref}}(u, v) - r \cdot t(u, v)$. If such change of sign is detected, the critical clock period C_{uv} which causes zero shortest distance from current root u to v can be decided as $T_{\text{init}} - p_{\text{ref}}(u, v)/t(u, v)$. (Line 12-16, Line 18-24)

The detail of YTO is omitted here since we only use it to gather the shortest path distance information. Interested reader can refer to [14], [15].

B. False Case Pruning

As is mentioned in Section IV, checking the feasibility of a single domain candidate takes $O(|V||E|)$ time, which is unaffordable in large circuits because of the large number of candidates. Fortunately, many domain candidates can be ruled out by simple inspection of their value, saving the computational cost of feasibility check. Recall the process of constructing a shortest path tree and generating the corresponding domain values in Lemma 2 and Lemma 1. Without loss of generality, we assume root s_1 to be zero domain and all other domains have a value large than zero. In other words, the shortest path distance p_i from root s_1 to s_i in M_k must satisfy $p_i < 0$. Any shortest path tree violating this negative distance constraint can be regarded as a *false case* and thus can be pruned. More importantly, since we grow the shortest path tree and compute p_i incrementally, it is not always necessary to enumerate all shortest path trees. During the construction of M_k , if a new branch e starts from node s_t , then the range of $w(e)$ must lie in $(-\infty, -p_t)$ in order to satisfy the negative distance constraint. In implementation, we use an index array to store all the edges. Instead of enumerating all edges in each iteration of shortest path tree construction, we only enumerate edges with weight less than $-p_t$. The effect of our range-confining approach is similar to the branch-and-bound method.

C. Algorithm Flow

We are now ready to present the proposed optimal algorithm for multi-domain clock skew scheduling as Algorithm 2. The algorithm framework is binary search on T with feasibility check for all domain candidates, as is suggested by Theorem 2. Ordering graph based pruning in the outer loop (Line 5) and false case pruning in the inner loop (Line 11) are used to drastically reduce the computational effort of the feasibility checking process.

VI. EXPERIMENTAL RESULTS

We implemented the proposed optimal multi-domain clock skew scheduling algorithm in C++ and tested it on a Linux machine with an Intel quad-core 2.0GHz CPU and 2GB memory. Note that only one core is used for our experiments. The performance and solution quality are evaluated on ISCAS89 sequential benchmarks obtained from authors of [6]. Table I shows the results in comparison to those in [4] and [6]. Columns “ $|V|$ ” and “ $|E|$ ” give the number of vertices and edges in the constraint graph. Column “ T_{opt} ” gives the optimal cycle period from the unconstrained clock skew scheduling, which also acts as the lower bound for MDCSS. Column “CPU Time” reports the runtime of our algorithm. The results of cycle period for $k = 2, 3, 4$ domains from [4], [6] and our algorithm are shown in columns “SAT-based”, “M-clustering” and “Proposed” respectively.

Algorithm 2 Proposed Optimal Algorithm for MDCSS

```

1:  $C = \text{GetCriticalClockMatrix}(G = (V, E));$ 
2:  $T_l = T_{\text{opt}}, T_r = T_{\text{init}}, T = T_l + T_r;$ 
3: while  $(T_r - T_l)/T_l > \epsilon$  do
4:   Construct  $O_T$  from  $C$  and get  $K_T$ ;
5:   if  $K_T > k$  then
6:      $T_l = T, T = T_l + T_r;$ 
7:     CONTINUE;
8:   end if
9:   while Not all possibilities enumerated do
10:    get a feasible set  $\{l_1, \dots, l_k\}$  of domain values;
11:    if NOT  $\{0 = l_1 < l_2 < \dots < l_k\}$  then
12:      CONTINUE;
13:    else
14:      if  $\{l_1, \dots, l_k\}$  is feasible then
15:         $T_r = T, T = T_l + T_r;$ 
16:        BREAK;
17:      end if
18:    end if
19:  end while
20: end while

```

For convenience of comparison, all the cycle periods are normalized against T_{opt} as is in [6]. Results of multi-level clustering on 11 circuits are missing as they are not reported in [6].

First, we analyze the optimality and runtime of the proposed algorithm. As is shown by Table I, the proposed algorithm delivers identical results to those of SAT-based optimal algorithm on all the circuits for $k = 2, 3, 4$, which verifies the its optimality. As for the runtime, the proposed algorithm finishes each case within 4 seconds, with most of them less than 0.1 second. Column “#Prune” lists the percentage of feasibility checking saved from the two pruning techniques described in Section V. Since the two pruning techniques act on different levels, the total percentage of pruned cases is computed with the following formula:

$$\#Prune = (1 - (1 - P_{O_T})(1 - P_{\text{False}})) \times 100\%,$$

where P_{O_T} and P_{False} are percentage of pruned cases by ordering graph based and false case based pruning techniques respectively. Except for the five trivial cases ($T_{\text{init}} = T_{\text{opt}}$) which do not require pruning, on average 92% of the feasibility checking process can be eliminated, which demonstrates the effectiveness of our pruning techniques in helping achieve a practically efficient optimal algorithm. Circuits *s400*, *s444* and *s15850* are identified as hard cases due to their relatively longer runtime than all the other circuits. For these three cases, due to their internal circuit structures, the proposed algorithm have to go through many shortest tree scenarios in order to completely check the feasibility of domain distribution for a fixed T . The relatively lower pruning percentage in Column “#Prune” confirms the above argument. However, even for these hard cases, the proposed algorithm is still able to prune most of the false domain candidates and yields the optimal solutions within a few seconds.

Next, we compare the propose algorithm with previous work [4], [6]. In [4], the exact runtime of SAT-based algorithm for ISCAS89 cases were not shown. Instead, the author claimed that it took less than one minute for a server with Intel Pentium III 2.0GHz CPU and 2GB memory to solve the MDCSS for 27 of circuits, while for others it took longer. Even regarding the advance of microprocessors, we argue that the proposed optimal algorithm is still very competitive against the SAT-based method. For the multi-level clustering algorithm [6], it is reported that it took less than 2 seconds to finish each of the circuits listed with a result in Table I, which

TABLE I
RESULTS OF MULTI-DOMAIN CLOCK SKEW SCHEDULING ON ISCAS89 SEQUENTIAL BENCHMARKS

Design	V	E	T_{opt}	#Prune (%)			CPU Time (s)			T/T_{opt}								
				$k=2$	$k=3$	$k=4$	$k=2$	$k=3$	$k=4$	SAT-based[4]			M-clustering[6]			Proposed		
										$k=2$	$k=3$	$k=4$	$k=2$	$k=3$	$k=4$	$k=2$	$k=3$	$k=4$
s27	5	21	5.06	90.5	89.9	88.7	< 0.1	< 0.1	< 0.1	1.14	1.00	1.00	-	-	-	1.14	1.00	1.00
s208	10	70	9.91	97.3	96.7	96.7				1.00	1.00	1.00	-	-	-	1.00	1.00	1.00
s298	16	86	10.79	90.8	85.5	85.5				1.05	1.00	1.00	1.07	1.07	1.05	1.05	1.00	1.00
s344	17	121	13.15	94.3	77.2	74.2				1.09	1.00	1.00	1.09	1.00	1.00	1.09	1.00	1.00
s349	17	121	13.51	96.7	76.3	69.5				1.09	1.01	1.00	1.09	1.00	1.00	1.09	1.01	1.00
s382	23	175	9.63	74.6	74.9	89.9				1.20	1.01	1.00	1.20	1.03	1.03	1.20	1.01	1.00
s386	8	129	9.61	97.3	91.6	91.3				1.04	1.00	1.00	1.04	1.00	1.00	1.04	1.00	1.00
s400	23	175	9.89	79.0	72.1	85.8				1.17	1.03	1.02	1.18	1.03	1.03	1.17	1.03	1.02
s420	18	146	21.13	99.1	98.0	98.0				< 0.1	1.00	1.00	1.00	-	-	1.00	1.00	1.00
s444	23	175	8.10	73.3	73.6	73.1				3.19	1.34	1.18	1.10	1.39	1.18	1.15	1.34	1.18
s510	8	103	14.29	-	-	-	< 0.1	< 0.1	< 0.1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
s526	23	167	11.22	84.1	92.2	92.6				1.05	1.00	1.00	1.07	1.07	1.05	1.05	1.00	1.00
s526n	23	167	11.31	84.3	96.1	96.0				1.05	1.00	1.00	-	-	-	1.05	1.00	1.00
s641	21	486	29.51	99.2	98.9	98.8				1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
s713	21	486	30.58	-	-	-				1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
s820	7	213	16.74	-	-	-				1.00	1.00	1.00	-	-	-	1.00	1.00	1.00
s832	7	213	16.22	-	-	-				1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
s838	34	298	44.66	99.3	98.6	98.6				1.00	1.00	1.00	-	-	-	1.00	1.00	1.00
s953	8	94	14.38	97.2	90.3	90.3				1.03	1.00	1.00	-	-	-	1.03	1.00	1.00
s1196	19	365	22.28	-	-	-				1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
s1238	19	365	24.33	98.9	92.0	92.0	< 0.1	< 0.1	< 0.1	1.00	1.00	1.00	1.03	1.03	1.01	1.00	1.00	1.00
s1423	76	2235	73.13	99.8	99.7	97.5				1.03	1.01	1.00	1.05	1.04	1.04	1.03	1.01	1.00
s1488	8	266	23.18	98.9	97.3	96.7				1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
s1494	8	266	23.85	98.9	95.7	95.7				1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
s5378	165	2180	22.88	99.4	90.2	90.3				1.13	1.01	1.00	-	-	-	1.13	1.01	1.00
s9234	140	2226	33.76	99.6	99.6	99.6				1.01	1.00	1.00	-	-	-	1.01	1.00	1.00
s13207	471	3885	53.36	99.6	99.6	99.6				1.03	1.00	1.00	-	-	-	1.03	1.00	1.00
s15850	565	16375	85.27	84.0	98.1	75.2				3.79	1.08	1.04	1.04	-	-	1.08	1.04	1.04
s35932	1442	6128	286.24	95.4	95.4	95.4				< 0.1	1.00	1.00	1.00	1.01	1.01	1.01	1.00	1.00
s38417	1465	31980	86.19	99.9	99.9	99.9				0.14	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
s38584	1451	17900	286.62	99.9	99.9	99.9				< 0.1	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00

is faster than the propose algorithm. However, multi-level clustering algorithm does not theoretically have optimality guarantee. Its results in Table I show a degradation of 2% – 7% against the optimal solution. On the other hand, the proposed algorithm yields optimal solutions within comparable amount of time.

VII. CONCLUSION

In this paper, we thoroughly studied the multi-domain clock skew scheduling problem. We proved that the problem is NP-complete if the domain number is large and showed that it is polynomial time tractable if the domain number is a small constant. Based on the theoretical analysis, we developed an optimal algorithm with efficient pruning techniques for MDCSS for small domain number, which is the normal case in practice. Experimental results confirmed the optimality of the proposed algorithm and demonstrated its efficiency.

ACKNOWLEDGEMENT

We would like to thank Tsuyoshi Ito from the University of Waterloo for his help on the NP-completeness proof, and Yanling Zhi from Fudan University for inspirations and discussions. This work is supported by the NSF under CCF-0811270.

REFERENCES

- [1] J. P. Fishburn, "Clock skew optimization," *IEEE Transactions on Computers*, vol. 39, no. 7, pp. 945–951, 1990.
- [2] R. B. Deokar and S. S. Sapatnekar, "A graph-theoretic approach to clockskew optimization," in *Proc. Intl. Symposium on Circuits and Systems*, 1994, pp. 407–410.
- [3] A. Vittal, H. Ha, F. Brewer, and M. Marek-Sadowska, "Clock skew optimization for ground bounce control," in *Proc. Intl. Conf. on Computer-Aided Design*, 1996, pp. 395–399.
- [4] K. Ravindran, A. Kuehlmann, and E. Sentovich, "Multi-domain clock skew scheduling," in *Proc. Intl. Conf. on Computer-Aided Design*, 2003, pp. 801–808.
- [5] Y. Zhi, H. Zhou, and X. Zeng, "A practical method for multi-domain clock skew optimization," in *Proc. Asian and South Pacific Design Automation Conference*, 2011, pp. 521–526.
- [6] J. Casanova and J. Cortadella, "Multi-level clustering for clock skew optimization," in *Proc. Intl. Conf. on Computer-Aided Design*, 2009, pp. 547–554.
- [7] Y. Zhi, W.-S. Luk, H. Zhou, C. Yan, H. Zhu, and X. Zeng, "An efficient algorithm for multi-domain clock skew scheduling," in *Proc. DATE: Design Automation and Test in Europe*, 2011.
- [8] M. Ni and S. O. Memik, "A fast heuristic algorithm for multidomain clock skew scheduling," *IEEE Transactions on Very Large-Scale Integrated Systems*, vol. 18, no. 4, pp. 630–637, 2010.
- [9] N. MohammadZadeh, M. Mirsaedi, A. Jahanian, and M. S. Zamani, "Multi-domain clock skew scheduling-aware register placement to optimize clock distribution network," in *Proc. DATE: Design Automation and Test in Europe*, April 2009, pp. 833–838.
- [10] S.-H. Huang, C.-M. Chang, and Y.-T. Nieh, "Fast multi-domain clock skew scheduling for peak current reduction," in *Proc. Asian and South Pacific Design Automation Conference*, 2006, pp. 254–259.
- [11] C. Lin and H. Zhou, "Clock skew scheduling with delay padding for prescribed skew domains," in *Proc. Asian and South Pacific Design Automation Conference*, 2007, pp. 541–546.
- [12] G. J. Woeginger and Z. Yu, "On the equal-subset-sum problem," *Inf. Process. Lett.*, vol. 42, no. 6, pp. 299–302, 1992.
- [13] J. P. Fishburn, "Solving a system of difference constraints with variables restricted to a finite set," *Inf. Process. Lett.*, vol. 82, no. 3, pp. 143–144, 2002.
- [14] N. E. Young, R. E. Tarjan, and J. B. Orlin, "Faster parametric shortest path and minimum balance algorithms," *IEEE Transactions on Computers*, vol. 21, pp. 205–221, March 1991.
- [15] A. Dasdan, "Experimental analysis of the fastest optimum cycle ratio and mean algorithms," *ACM Transactions on Design Automation of Electronic Systems*, vol. 9, no. 4, pp. 385–418, October 2004.