An Efficient Algorithm for Multi-Domain Clock Skew Scheduling

Yanling Zhi[1], Wai-Shing Luk[1], Hai Zhou[1,2], Changhao Yan[1*], Hengliang Zhu[1], Xuan Zeng[1*]
[1]State Key Lab. of ASIC & System, Microelectronics Department, Fudan University, China
[2]Department of EECS, Northwestern University, U.S.A.

***Abstract*— Conventional clock skew scheduling for sequential circuits can be formulated as a minimum cycle ratio (MCR) problem, and hence can be solved effectively by methods such as Howard's algorithm. However, its application is practically limited due to the difficulties in reliably implementing a large set of arbitrary dedicated clock delays for the flip-flops. Multi-domain clock skew scheduling was proposed to tackle this impracticality by constraining the total number of clock delays. Even though this problem can be formulated as a mixed integer linear programming (MILP), it is expensive to solve optimally in general. In this paper, we show that, under mild restrictions, the underlying domain assignment problem can be formulated as a special MILP that can be solved effectively using similar techniques for the MCR problem. In particular, we design a generalized Howard's algorithm for solving this problem efficiently. We also develop a critical-cycle-oriented refinement algorithm to further improve the results. The experimental results on ISCAS89 benchmarks show both the accuracy and efficiency of our algorithm. For example, only 4.3% of the tests have larger than 1% degradation (3% in the worst case), and all the tests finish in less than 0.7 seconds on a laptop with a 2.1GHz processor.**

I. Introduction

The performance of a sequential circuit is determined by the longest combinational logic path between flip-flops. The clock arrival time to a flip-flop is known as its *clock latency*, and the relative differences in latencies between flip-flops are referred to as *clock skews*. Clock skew scheduling [1] optimizes the performance of a circuit by deliberately assigning different clock latencies to the flip-flops, so as to "borrow" time from paths with larger slacks and use it in the more critical ones. This scheduling problem can be formulated as a minimum cycle ratio (MCR) problem, which can be solved efficiently by many different algorithms [2], such as Howard's algorithm [3]. The computed clock latencies are then implemented by a specific clock tree network. However, in practice, a clock schedule with a wide spectrum of arbitrary values cannot be realized in a reliable manner, because the implementation of dedicated delays using interconnects and additional buffers is highly susceptible to intra-die process variations.

Ravindran et al. proposed multi-domain clock skew scheduling [4] to tackle this impracticality. Instead of delivering arbitrary clock latencies in a precise manner, multi-domain clock skew scheduling only delivers a constant number of latencies called *clocking domains*. The problem has been formulated as a mixed integer linear programming (MILP) problem, and a SAT-based algorithm has been proposed, where a SAT solver [5] is used to enumerate the assignment of flip-flops to clocking domains. The algorithm is exact, but too expensive for real designs. For example, it took the algorithm twenty hours to compute the latencies of a circuit with 2921 flip-flops in four domains.

Casanova et al. later proposed a multi-level clustering algorithm [6] to tackle the problem. The algorithm progressively groups flip-flops with *skew affinity*, quantified by the overlap of their *slack intervals* for a given cycle period. Here the slack interval of a flip-flop is the latency range that it can have without violating any timing constraints. Slack intervals are distributed by slack optimization algorithms [7]. Compared with the work in [4], this algorithm is significantly faster, but the solution quality is also substantially sacrificed. For example, the solutions in 25% of the circuits have greater than 1% degradation compared with the optimal solution, and several solutions even have 7% degradation. This is mainly due to the local decision nature of the clustering operation.

It was shown in [4] and [6] that the optimization potential of clock skew scheduling can be reliably implemented using only a few clocking domains. Ni et. al proposed to minimize the total number of domains for the optimal cycle period [8]. Although it was found that, in some cases, many domains may be needed to achieve the optimal cycle period, it was also shown that most of the circuits need no more than six domains. Moreover, given the expense of reliably implementing more domains in the clock tree network, it is usually not wise to increase the domain number just for tiny improvement in the cycle period.

In this paper, we propose a better algorithm for the multi-domain clock scheduling problem as defined in [4] and [6]. We discover that if imposed mild restrictions on the clocking domains, the multi-domain clock skew scheduling problem can be transformed into a special MILP that can be solved optimally and efficiently using similar techniques for MCR problems. It is due to the monotonicity nature of the transformed problem. The main contributions of the paper are as follows.

1) An MILP formulation is proposed for obtaining the clocking domain assignment for flip-flops. This formulation is capable of giving a good domain assignment as it only mildly tightens the constraints on the clocking domains. Moreover, it can be solved optimally once and for all, thus avoids the shortcomings of the multi-level clustering algorithm.
2) A generalized Howard's algorithm is proposed to efficiently solve the MILP problem. The algorithm takes advantage of the monotonicity in the timing constraints, and has a similar performance as the original Howard's algorithm [3][2].
3) A critical-cycle-oriented refinement algorithm is developed to further improve the result. Instead of iterating over each flip-flop and determining whether the result can be improved by assigning it to other clocking domains, our algorithm focuses on the domain assignment edges in the critical cycle. The efficiency of this technique has been validated in the experiments.

The rest of the paper is organized as follows. In Section II, the multi-domain clock skew scheduling problem is formally formulated. The algorithmic ideas are discussed in Section III, while the details are described in Section IV. Experimental results are shown in Section V. Finally the conclusions are given in Section VI.

*Corresponding authors. E-mails: yanch@fudan.edu.cn, xzeng@fudan.edu.cn.
978-3-9810801-7-9/DATE11/©2011 EDAA

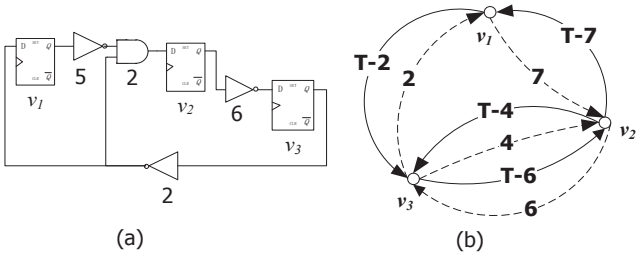

Fig. 1. Example for timing constraint model: (a) a sequential circuit with gate delays; (b) timing constraint graph

II. Problem formulation

A. Timing Constraint Model

A sequential circuit must satisfy both setup and hold time constraints to work correctly. Let u, v denote two consecutive flip-flops connected by combinational paths in a sequential circuit, and $d_{\max}(u,v), d_{\min}(u,v)$ denote the maximum and minimum delays from u to v. We use $l(u), l(v)$ to denote the clock latencies of u and v respectively, and $d_s(v), d_h(v)$ to denote the setup time and hold time of v respectively. Furthermore, T denotes the clock period.

Setup time constraints make sure that the signal from u to v has enough time to stabilize its value before v stores it:

$$l(u) + d_{\max}(u,v) \leq l(v) + T - d_s(v). \tag{1}$$

On the other hand, *hold time constraints* make sure that the signal from u does not overwrite the previous data before v stores it:

$$l(u) + d_{\min}(u,v) \geq l(v) + d_h(v). \tag{2}$$

Setup and hold time constraints can be represented in a timing constraint graph. Let $G = (V, E_s, E_h)$ denote the graph, in which the set of vertices V corresponds to the flip-flops, and the sets $E_s \subseteq V \times V$ and $E_h \subseteq V \times V$ correspond to the setup and hold edges respectively. The setup and hold edges are constructed as follows. For the setup timing constraint in (1), a directed edge from v to u with weight $w(v,u) = T - d_{\max}(u,v) - d_s(v)$ is included in G, and for the hold timing constraints in (2), a directed edge from u to v with weight $w(u,v) = d_{\min}(u,v) - d_h(v)$ is included. It is worth noting that primary inputs and outputs are represented as a single vertex in the timing constraint graph.

Figure 1 shows an example of the construction of setup edges and hold edges. A sequential circuit with gate delays are shown in Figure 1(a). For simplicity, setup time and hold time for each flip-flop are assumed to be zero. The solid and dashed lines in 1(b) correspond to the setup edges E_s and hold edges E_h respectively.

B. Multi-domain Clock Skew Scheduling

Given a sequential circuit, the objective of conventional clock skew scheduling [1] is to minimize the clock period while satisfying the constraints in (1) and (2). This problem can be formulated as a minimum cycle ratio (MCR) problem. For this we associate each edge (u,v) in G with *transit time* $\tau(u,v)$:

$$\tau(u,v) = \begin{cases} 1, & \text{if } (u,v) \in E_s; \\ 0, & \text{if } (u,v) \in E_h. \end{cases} \tag{3}$$

An initial cycle period T_{init} is also provided, and the weight associated with each edge (u,v) in G is rewritten as:

$$w(u,v) = \begin{cases} T_{init} - d_{\max}(v,u) - d_s(u), \text{if } (u,v) \in E_s; \\ d_{\min}(u,v) - d_h(v), \text{if } (u,v) \in E_h. \end{cases} \tag{4}$$

Now the clock skew scheduling problem is formulated as the following MCR problem:

$$\begin{aligned} &\max \quad r \\ &s.t.\ \ l(v) - l(u) \leq w(u,v) - r \cdot \tau(u,v), \forall (u,v) \in E_s \cup E_h, \end{aligned} \tag{5}$$

where r is the cycle ratio variable. Let r^* be the optimal solution of Problem (5), then the optimal cycle period is $T^* = T_{init} - r^*$. Therefore in the sequel the objective of our algorithm is always maximizing r instead of minimizing T.

Multi-domain clock skew scheduling [4] imposes additional constraints on the clock latencies of flip-flops. Let the number of skew domains be k, and their clock latencies be $d_1, d_2, \cdots, d_k$ whose values are unknown, then the latency of each flip-flop must be one of d_i's ($i = 1, 2, \cdots, k$). The problem is formally stated as:

$$\begin{aligned} &\max \quad r \\ &s.t.\ \ l(v) - l(u) \leq w(u,v) - r \cdot \tau(u,v), \forall (u,v) \in E_s \cup E_h, \\ &\qquad l(u) \in \{d_1, d_2, \cdots, d_k\}, \forall u \in V. \end{aligned} \tag{6}$$

Note that the problem reduces to the conventional clock skew scheduling problem if $k \geq |V|$, whereas typically $k \ll |V|$.

III. Algorithmic Ideas

Multi-domain clock skew scheduling in (6) is hard to solve optimally in general. We tackle it by imposing further mild restrictions on the clocking domains and formulating it as a special MILP problem. In this section, we will first present the formulation of MILP and how to solve it. Then the derivation of critical-cycle-oriented refinement algorithm will be described.

A. Mixed Integer Linear Programming Formulation

The complexity of multi-domain clock skew scheduling is essentially due to the unknown latency values of the clocking domains. In order to make this problem solvable, we formulate an MILP problem by restricting the domain values to be integers. To facilitate this, scaling is first imposed on the weight and transit time on each edge as well as the clock latencies of each vertex in the timing constraint graph. Let c be a constant scaling factor. The new weight and transit time associated with each edge in G, and new clock latency for each vertex are defined as:

$$\hat{w}(u,v) = w(u,v)/c \quad \forall (u,v) \in E_s \cup E_h \tag{7a}$$

$$\hat{\tau}(u,v) = \tau(u,v)/c, \quad \forall (u,v) \in E_s \cup E_h. \tag{7b}$$

$$\hat{l}(u) = l(u)/c, \qquad \forall u \in V \tag{7c}$$

Let the new clocking domains be $\{\hat{d}_i | \hat{d}_i = d_i/c, i = 1, 2, \cdots, k\}$. Then Problem (6) is equivalent to:

$$\begin{aligned} &\max \quad r \\ &s.t.\ \ \hat{l}(v) - \hat{l}(u) \leq \hat{w}(u,v) - r \cdot \hat{\tau}(u,v), \forall (u,v) \in E_s \cup E_h \\ &\qquad \hat{l}(u) \in \{\hat{d}_1, \hat{d}_2, \cdots, \hat{d}_k\}, \forall u \in V. \end{aligned} \tag{8}$$

By selecting a proper scaling factor c, we can mildly tighten the domain constraints in (8) by restricting the values of $\hat{d}_i$'s to be integers. Here c is always selected in a way that the latencies of all the flip-flops are within k integers. Then the problem is transformed into an MILP problem:

$$\begin{aligned} &\max \quad r \\ &s.t.\ \ \hat{l}(v) - \hat{l}(u) \leq \hat{w}(u,v) - r \cdot \hat{\tau}(u,v), \forall (u,v) \in E_s \cup E_h \\ &\qquad \hat{l}(u) \in \mathbb{Z}, \forall u \in V. \end{aligned} \tag{9}$$

Note that the constraints of integer domain values in Problem (9) distribute the clock latencies of the flip-flops in a uniform manner. If the domains in the optimal solution of Problem (8) are distributed uniformly, solving Problem (9) can obtain the same optimal solution, otherwise the result may be suboptimal for (8). Nonetheless, the result can still be close. Moreover, the integer clock latencies can be treated as a domain assignment for the flip-flops, in which the flip-flops having the same latencies are assigned to the same domain. Then real clock latencies for the flip-flops are obtained by solving the clock skew scheduling under this domain assignment, and the result r can be improved further. The result should become more accurate if more domains are given, as it is easier to distribute the clock latencies uniformly when more domains are allowed.

The complexity of a general MILP problem is NP-hard, but it turns out that Problem (9) is a special case that can be solved efficiently. To see this, notice that Problem (9) is equivalent to:

$$\begin{aligned} &\max \quad r \\ &s.t.\ \ \hat{l}(v) - \hat{l}(u) \leq \lfloor \hat{w}(u,v) - r \cdot \hat{\tau}(u,v) \rfloor, \forall (u,v) \in E_s \cup E_h. \end{aligned} \tag{10}$$

This is the same as the MCR problems in (5) except the right hand side of the constraints. Nonetheless, they share the same property that the expression on the right hand side of the constraints is *monotonically decreasing* with respect to r. Recall that a function $f(r)$ is called monotonically decreasing if and only if $f(r_1) \leq f(r_2)$ for all $r_1 \geq r_2$. A generalization of such problem has been shown in [9], whereas we develop a more efficient algorithm to solve it. Let $f_{uv}(r) = \lfloor \hat{w}(u,v) - r \cdot \hat{\tau}(u,v) \rfloor$. Initially an upper bound is assigned to r. If r is not the solution of (10), which means we can not find such $\hat{l}(u)$'s that satisfy the constraints, then there must exist at least one negative cycle C such that $\sum_{(u,v)\in C} f_{uv}(r) < 0$. We can minimally decrease r and "zero out" C by solving equation $\sum_{(u,v)\in C} f_{uv}(r) = 0$. In this way, r is iteratively decreased until there is no negative cycle, and thus becomes the optimal one (the solution). This algorithm can be easily extended to solving other similar problems as long as $f_{uv}(r)$ is monotonically decreasing with respect to r. For example, it reduces to original Howard's algorithm if $f_{uv}(r) = w(u,v) - r \cdot \tau(u,v)$. Hence it is called generalized Howard's algorithm in the sequel.

B. Critical-Cycle-Oriented Refinement

Although a good clocking domain assignment can be obtained by solving the MILP problem formulated in the previous section, it can still be further improved by adjusting the domain assignments.

To represent the multi-domain constraints, the timing constraint graph G is extended by introducing a set of domain vertices and domain assignment edges between flip-flops and domains [4]. Let $G = (V, D, E_s, E_h, E_a)$ denote the multi-domain timing graph, where the set of vertices V and the set of edges E_s and E_h have the same definition as before. D represents the set of vertices corresponding to the clocking domains and E_a is the set of edges representing assignments of flip-flops to domains. For any $v \in V$, if v is assigned to Domain $d \in D$, then two edges (d, v) and (v, d) are included in E_a with weights and transit times all equal to zero. Now the clock skew scheduling under a given domain assignment is transformed into the same form as conventional clock skew scheduling, and can be solved using the same MCR algorithms. There are three observations:

1) The MCR in unconstrained clock skew scheduling is the upper bound of that in multi-domain case.
2) The *critical cycle* (of which the cycle ratio is minimal) in multi-domain clock skew scheduling must contain at least one edge in E_a, unless the result is the same as that in unconstrained clock skew scheduling, in which case there is no need to refine.
3) Only adjustments on the domain assignments related with at least one of the domain assignment edges in the critical cycle can improve the result. Changes only on other domain assignment will never refine the result, as the same critical cycle still exists.

Based on the observations above, an efficient refinement algorithm can be derived. Instead of iterating over each flip-flop and checking whether the result can be improved by adjusting its domain assignment, the algorithm focuses on the domain assignment edges in the critical cycle. In each iteration, some domain assignments corresponding to the domain assignment edges in the critical cycle are adjusted to improve the result. Such strategy guarantees that the result is improved in every iteration, which is a great advantage for efficiency.

IV. Algorithm Details

Algorithm 1 SmipRef(G, k)

```
1: assignment := φ
2: r := −∞;
3: r* := CSSUnconstrained(G);
4: latencies := distributeLatencies(G, r*);
5: c := calculateInitialScalingFactor(latencies, k);
6: while true do
7:    scaleAndFormulateMILP(G, c);
8:    r_m := solveMILP(G, r*);
9:    latencies := distributeIntegerLatencies(G, r_m);
10:   if #values in latencies > k then
11:      break;
12:   end if
13:   if r < r_m then
14:      r := r_m;
15:      assignment := assignDomains(latencies);
16:   end if
17:   updateScalingFactor(c);
18: end while
19: r := CSSUnderDomainAssignment(G, assignment);
20: r := refine(G, assignment, k);
21: return r;
```

The main flow of our algorithm SmipRef for multi-domain clock skew scheduling problem is shown in Algorithm 1. The domain assignment $assignment$ and r are initialized in Lines 1-2. Then the initial scaling factor c is calculated in Lines 3-5. In our implementation, the clock skew scheduling (i.e. the MCR problem) is solved by Howard's algorithm [3]. Clock latencies of the flip-flops are distributed after the optimal solution in the unconstrained case r^* is calculated. The latency distribution subroutine is shown in Algorithm 2, which is a modified version of Bellman-Ford algorithm [10]. The latencies are computed by first adding a virtual source vertex s and zero-weighted edges from s to all the vertices in G, and then calculating the shortest path tree rooted at s. The latency of any flip-flop v is the shortest distance from s to v. This algorithm is guaranteed to obtain latencies that satisfy the timing constraints as long as $r \leq r^*$.

During the iterations in Lines 6-18, the algorithm progressively searches for a proper scaling factor c. At each iteration, the MILP

problem is constructed from a specific c and solved in Lines 7-8. Then the solution r_m is used to distribute the integer latencies in Line 9. The distribution of integer latencies is the same as Algorithm 2 except that the edge weight $w(u,v) - r \cdot \tau(u,v)$ is replaced by $\lfloor w(u,v) - r \cdot \tau(u,v) \rfloor$. Solution r_m is *feasible* if the integer latencies of all the flip-flops are within k integers, where k is the number of domains. The iterations stop whenever an infeasible solution r_m is found for current c, otherwise c is updated for next iteration and the search continues. The MILP problem formulated during the iterations that has the largest feasible solution r_m will be selected, and a domain assignment is constructed from the integer latencies distributed for r_m. In line 19, MCR for the clock skew scheduling under the domain assignment is calculated. After that, the critical-cycle-oriented refinement is invoked to further improve the result.

Algorithm 2 distributeLatencies(G, r)

```
1: latencies[u] := 0, ∀u ∈ V;
2: for i = 1 to |V| − 1 do
3:    changed := false;
4:    for each edge (u, v) in G do
5:       if latencies[v] > latencies[u] + w(u, v) − r·τ(u, v) then
6:          latencies[v] := latencies[u] + w(u, v) − r · τ(u, v);
7:          changed := true;
8:       end if
9:    end for
10:   if not changed then
11:      break;
12:   end if
13: end for
14: return latencies;
```

Important subroutines in Algorithm SmipRef are described in details below, then the complexity analysis of the overall algorithm is given, and finally an example is provided to demonstrate the flow of our algorithm.

A. Search for a Proper Scaling Factor

Let $l_{\min}(r)$ and $l_{\max}(r)$ denote the minimum and maximum latencies, respectively, for a given MCR r. A rough estimation for c is to partition the latency range $[l_{\min}(r), l_{\max}(r)]$ into $k-1$ intervals, and make the endpoints of the intervals be integers after scaling. Thus we have:

$$c = (l_{\max}(r) - l_{\min}(r))/(k-1), k \geq 2 \tag{11}$$

If r is the optimal solution of Problem (6), the MILP problem formulated with such a c should almost always obtain a feasible solution. In case any clock latency lies outside the range of the k integers, it can be adjusted to the closest valid one. However, the optimal solution is not known so far, and therefore we use $r = r^*$ as a starting point to search, where r^* is the optimal solution for unconstrained clock skew scheduling. As we mentioned, r^* is the upper bound for multi-domain clock skew scheduling. With the decrease of r, the latency range $[l_{\min}(r), l_{\max}(r)]$ becomes smaller, and c should be smaller as well. Therefore we can gradually decrease c to find its proper value. In our implementation, we do this by increasing the denominator in Equation (11) by a step of 0.1. Although there are many ways to search for a good scaling factor, this heuristic strategy makes a good tradeoff between accuracy and performance. Note that the "most" proper value of c is not necessary since we only treat the integer latencies for the flip-flops as their domain assignment, and moreover a refinement subroutine will be invoked to improve the result later.

B. MILP Solver

Algorithm 3 solveMILP(G, r^*)

```
1: r_m := r*;
2: while there exists a negative cycle C under r_m do
3:    r_m := Σ_{(u,v)∈C} ŵ(u, v) / Σ_{(u,v)∈C} τ̂(u, v);
4:    while Σ_{(u,v)∈C} ⌊ŵ(u, v) − r_m · τ̂(u, v)⌋ < 0 do
5:       find the largest r_1 below r_m such that:
         ∃(u, v) ∈ C, ŵ(u, v) − r_1 · τ̂(u, v) ∈ ℤ;
6:       r_m := r_1;
7:    end while
8: end while
9: return r_m
```

The generalized Howard's algorithm for MILP problem (10) is shown in Algorithm 3. The solution for the unconstrained clock skew scheduling problem r^* is used as the initial estimate for r_m. In Lines 2-8, negative cycles are progressively "zeroed out" by decreasing r_m. In the actual implementation, the algorithm flow is much similar to the improved version of Howard's algorithm in [2], where a successor graph and vertex distance labels are maintained during the iterations. To simplify the presentation, we omit the description of these improvement techniques.

In Lines 3-6, equation $\sum_{(u,v)\in C} f_{uv}(r_m) = 0$ is solved, where $f_{uv}(r_m) = \lfloor \hat{w}(u,v) - r_m \hat{\tau}(u,v) \rfloor$ and C is a negative cycle. Since $f_{uv}(r_m)$ is a staircase function whose turning points are all at r_m's such that $\hat{w}(u,v) - r_m \cdot \hat{\tau}(u,v) \in \mathbb{Z}$, $\sum_{(u,v)\in C} f_{uv}(r_m)$ is also a staircase function of r_m, in which any turning point must be at an r_m such that $\exists (u,v) \in C, \hat{w}(u,v) - r_m \cdot \hat{\tau}(u,v) \in \mathbb{Z}$. Based on this observation, we solve the equation in the following way. An upper bound for r_m is first calculated as $r_m = \sum_{(u,v)\in C} \hat{w}(u,v) / \sum_{(u,v)\in C} \hat{\tau}(u,v)$. Then the turning points below the upper bound are tested in decreasing order until the one making the cycle nonnegative. In the worst case, it takes $|C|$ (the number of edges in C) iterations to converge. However, the solution is normally very close to the initial upper bound and $|C|$ is very small. Thus it takes only a few iterations to converge, and the overhead for solving $\sum_{(u,v)\in C} f_{uv}(r_m) = 0$ is negligible. Therefore, the performance of our generalized Howard's algorithm is similar to the original Howard's algorithm, which is experimentally one of the fastest algorithms for MCR problem.

C. Critical-Cycle-Oriented Refinement

The critical-cycle-oriented refinement algorithm is shown in Algorithm 4. In Lines 2-17, the result is iteratively improved by adjusting domain assignments corresponding to the domain assignment edges in the critical cycle C. For efficiency, we limit the adjustment to only one edge in each iteration. At each iteration, the flip-flop whose domain assignment can be modified to obtain the largest improvement is selected and assigned to its "best" domain. In finding best domain assignment for a flip-flop, Howard's algorithm is invoked to compute the new MCR after adjusting domain assignment of the flip-flop. The iterations stop if no improvement can be made by adjusting any domain assignment edge in the critical cycle.

Since a good solution is already obtained before refinement, and improvement can be made in every iteration, the convergence of Algorithm 4 is very fast.

D. Complexity Analysis

The overall SimpRef algorithm consists of two stages: search for a proper scaling factor and critical-cycle-oriented refinement. The

Algorithm 4 refine(G, $assignment$)

```
1: (v_ch, d_ch) := φ;
2: while true do
3:    (r,C):=CSSUnderDomainConstraints(G,assignment);
4:    improved := false;
5:    for each domain assignment edge (v, d) (or (d, v)) in critical
      cycle C do
6:       (r_imp, d_imp) := findBestDomainAssignment(v);
7:       if r_imp > r then
8:          r := r_imp;
9:          (v_ch, d_ch) := (v, d_imp);
10:         improved := true;
11:      end if
12:   end for
13:   if not improved then
14:      break;
15:   end if
16:   updateDomainAssignment(G, assignment, v_ch, d_ch);
17: end while
18: return r
```

former requires a few invocations of generalized Howard's algorithm (whose complexity is similar to Howard's algorithm), while the latter is mostly finding best domain assignment subroutine for critical flip-flops, whose amount is not large due to the fast convergence of the iterations. As we mentioned, the subroutine of finding best domain assignment is implemented with Howard's algorithm. Therefore, the performance of our algorithm is practically asymptotic to a few invocations of Howard's algorithm.

E. Example

Figure 2 shows the working of our algorithm SmipRef on the circuit in Figure 1 for two clocking domains. The corresponding MCR problem is constructed with $T_{init} = 7$ as shown in Figure 2(a). Similar to the timing constraint graph, here a directed edge from v_i to v_j with weight $w(i, j)$ represents the constraint $l(v_i) + w(i, j) \geq l(v_j)$. The solution here is $r^* = 2$, and the latency range after distribution is $[-3, 0]$. Then the initial scaling factor is calculated as $c = 3$. Now the weight and transit time associated with each edge are scaled, and the MILP problem is constructed as shown in Figure 2(b). After solving the MILP problem using Algorithm 3, we obtain the result $r_m = 1$ and the integer clocking latencies of the flip-flops: $\hat{l}(v_1) = -1$, $\hat{l}(v_2) = 0$ and $\hat{l}(v_3) = 0$. Thus the clocking domains can be assigned as $D(v_1) = 1$, $D(v_2) = 2$ and $D(v_3) = 2$, where $D(v_i)$ is the domain of v_i. During the search for a proper scaling factor, $c = 3$ is selected. Later the clock skew scheduling problem under this domain assignment is solved, and the result is $r = 1$. In the refinement phase, no improvement can be made by adjusting any domain assignment edge in the critical cycle. In fact, this domain assignment is the optimal one for the circuit with two domains, in which the minimum cycle period is achieved.

V. Experimental Results

We implemented our algorithm SmipRef in C++ and experimented on a laptop with an Intel dual-core 2.1GHz processor and 4GB memory. The accuracy and performance were evaluated on ISCAS89 sequential benchmarks, for which the timing data were provided by the authors of [6]. Due to the unavailability of the industrial circuits in [4] and [6], we did not test our program on those circuits.

Table I shows the results on the 31 ISCAS89 benchmarks in comparison to SAT-based algorithm in [4] and multi-level clustering algorithm in [6]. Columns "#Vertices" and "#Edges" show

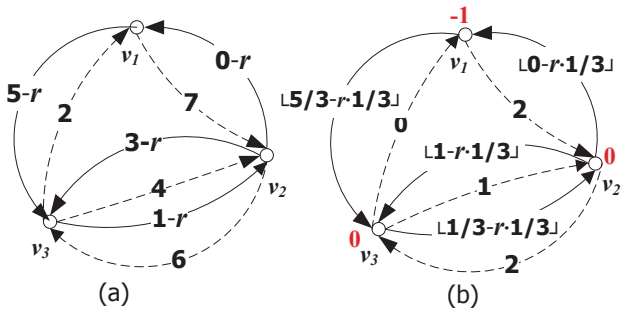

Fig. 2. Multi-domain clock skew scheduling for the circuit from Figure 1 with 2 domains: (a) MCR problem with $T_{init} = 7$; (b) MILP problem after scaling with $c = 3$ and the integer clock latencies (in red) for the solution $r = 1$.

the number of vertices and edges in the timing constraint graph respectively, while Column "T^{∞}_{cycle}" reports the optimal cycle period achievable through unconstrained clock skew scheduling. The results for $k = 2, 3$ and 4 domains from SAT-based algorithm, multi-level clustering and our algorithm are shown in Columns "SAT-based", "Multi-level clustering" and "SmipRef (ours)" respectively. Note that in multi-level clustering algorithm, the results of 11 circuits are missed in the original paper for unknown reasons. The cycle periods are all normalized to T^{∞}_{cycle} as in [6] for convenience of comparison.

The results from SAT-based algorithm are optimal and thus are used as reference for accuracy. Note that the differences on the normalized cycle periods between SAT-based algorithm and other algorithms are regarded as degradation. We compare our algorithm with multi-level clustering algorithm on accuracy in three aspects:

- tests that have greater than 1% degradation. In multi-level clustering algorithm, 15 of the 60 tests ($= 25\%$) have degradation greater than 1%, whereas in our algorithm, only 4 of the 93 tests ($= 4.3\%$) have.
- degradation in the worst case. In multi-level clustering algorithm, the degradation in the worst case is 7%, while it is only 3% in our algorithm.
- the average degradation. In multi-level clustering algorithm, the average degradation for 2, 3 and 4 domains are $1.2\%, 1.4\%$ and 1.4% respectively, whereas in our algorithm they are only $0.2\%, 0.1\%$ and 0.1%.

It is obvious that our algorithm is more accurate than multi-level clustering algorithm.

For the evaluation of the performance of our algorithm, Table II shows the runtimes in seconds on ISCAS89 benchmarks for 4 domains. The runtimes for 2 to 3 domains are omitted as they are very close to those for four domains. It is claimed in [6] that it takes multi-level clustering algorithm 2 seconds to test the largest circuit, whereas our algorithm finishes in less than 0.7 seconds on all the circuits. Obviously our algorithm is faster than multi-level clustering algorithm, not to mention SAT-based algorithm, and we believe that this advantage can be observed more clearly in even larger circuits.

So far as we know, there is no existing algorithms that solve the multi-domain clock skew scheduling problem as efficiently as our algorithm while obtaining solutions of better quality. The accuracy and efficiency of our algorithm make it very practical for real circuits.

TABLE I

RESULTS OF SIMPREF ON ISCAS89 SEQUENTIAL BENCHMARKS

Design	#Vertices	#Edges	T^{∞}_{cycle}	$T_{cycle}/T^{\infty}_{cycle}$								
				SAT-based [4]			Multi-level clustering [6]			SimpRef (ours)		
				$k=2$	$k=3$	$k=4$	$k=2$	$k=3$	$k=4$	$k=2$	$k=3$	$k=4$
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	1.00	1.00	1.00	1.03	1.03	1.01	1.00	1.00	1.00
s13207	471	3885	53.36	1.03	1.00	1.00	-	-	-	1.03	1.00	1.00
s1423	76	2235	73.13	1.03	1.01	1.00	1.05	1.04	1.04	1.03	1.01	1.00
s1488	8	266	23.18	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
s1494	8	266	23.85	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
s15850	565	16375	85.27	1.08	1.04	1.04	-	-	-	1.08	1.06	1.04
s208	10	70	9.91	1.00	1.00	1.00	-	-	-	1.00	1.00	1.00
s27	5	21	5.06	1.14	1.00	1.00	-	-	-	1.14	1.00	1.00
s298	16	86	10.79	1.05	1.00	1.00	1.07	1.07	1.05	1.07	1.00	1.00
s344	17	121	13.15	1.09	1.00	1.00	1.09	1.00	1.00	1.09	1.00	1.01
s349	17	121	13.51	1.09	1.01	1.00	1.09	1.00	1.00	1.09	1.01	1.01
s35932	1442	6128	286.24	1.00	1.00	1.00	1.01	1.01	1.01	1.00	1.00	1.00
s382	23	175	9.63	1.20	1.01	1.00	1.20	1.03	1.03	1.20	1.01	1.00
s38417	1465	31980	86.19	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
s38584	1451	17900	286.62	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
s386	8	129	9.61	1.04	1.00	1.00	1.04	1.00	1.00	1.04	1.00	1.00
s400	23	175	9.89	1.17	1.03	1.02	1.18	1.03	1.03	1.17	1.03	1.03
s420	18	146	21.13	1.00	1.00	1.00	-	-	-	1.00	1.00	1.00
s444	23	175	8.10	1.34	1.18	1.10	1.39	1.18	1.15	1.34	1.18	1.10
s510	8	103	14.29	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00	1.00
s526	23	167	11.22	1.05	1.00	1.00	1.07	1.07	1.05	1.05	1.00	1.00
s526n	23	167	11.31	1.05	1.00	1.00	-	-	-	1.05	1.00	1.00
s5378	165	2180	22.88	1.13	1.01	1.00	-	-	-	1.13	1.02	1.02
s641	21	486	29.51	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	1.00	1.00	1.00	-	-	-	1.00	1.00	1.00
s9234	140	2226	33.76	1.01	1.00	1.00	-	-	-	1.01	1.00	1.00
s953	8	94	14.38	1.03	1.00	1.00	-	-	-	1.06	1.00	1.00
Average				1.049	1.009	1.005	1.061	1.023	1.019	1.051	1.010	1.006

TABLE II

RUNTIMES FOR THE CIRCUITS IN ISCAS89 BENCHMARKS

Design	#Vertices	#Edges	Runtime(s) for $k=4$
s13207	470	6552	0.05
s15850	564	31138	0.69
s35932	1729	13304	0.05
s38417	1464	63662	0.22
s38584	1450	34668	0.08
s5378	164	2626	0.07
others			< 0.04

VI. CONCLUSIONS

In this paper, we presented an efficient algorithm for multi-domain clock skew scheduling. Under mild restrictions, the problem was formulated as a special MILP problem that can be solved efficiently using our generalized Howard's algorithm. A critical-cycle-oriented refinement algorithm was also developed to further improve the result. The experimental results on ISCAS89 sequential benchmarks validate the accuracy and efficiency of our algorithm. All the tests for 2 to 4 clocking domains finish in less than 0.7 seconds on a laptop with a 2.1GHz processor, with no more than 0.2% degradation on average in the results. The combination of efficient MILP solver and refinement procedure has made our algorithm very practical for real designs.

VII. ACKNOWLEDGEMENTS

This work was supported in part by the NSFC Research Projects 60976034 and 61076033, the National Basic Research Program of China under Grant 2005CB321701, the National Major Science and Technology Special Projects 2008ZX01035-001-06 and 2009ZX02023-4-3 of China during the 11th Five Year Plan Period, the Doctoral Program Foundation of the Ministry of Education of China 200802460068, the Program for Outstanding Academic Leader of Shanghai and NSF under CCF-0238484 and CCF-0811270.

The authors would like to thank Jonas Casanova and Jordi Cortadella for providing the timing data for ISCAS89 benchmarks.

REFERENCES

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

[2] A. Dasdan, "Experimental analysis of the fastest optimum cycle ratio and mean algorithms," *ACM Trans. on Design Automation of Electronic Systems*, vol. 9, no. 4, pp. 385–418, October 2004.

[3] R. A. Howard, *Dynamic Programming and Markov Process*. MIT Press, June 1960.

[4] K. Ravindran, A. Kuehlmann, and E. Sentovich, "Multi-domain clock skew scheduling," in *IEEE Proc. ICCAD*, 2003, pp. 801–808.

[5] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *IEEE Proc. DAC*, 2001, pp. 530–535.

[6] J. Casanova and J. Cortadella, "Multi-level clustering for clock skew optimization," in *IEEE Proc. ICCAD*, 2009, pp. 547–554.

[7] C. Albrecht, B. Korte, J. Schietke, and J. Vygen, "Cycle time and slack optimization for VLSI-chips," in *IEEE/ACM Proc. Digest of Technical Papers Compuater-Aided Design*, 1999, pp. 232–238.

[8] M. Ni and S. O. Memik, "A fast heuristic algorithm for multidomain clock skew scheduling," *IEEE Trans. on VLSI*, vol. 18, no. 4, pp. 630–637, 2010.

[9] Y. Wang, W.-S. Luk, X. Zeng, J. Tao, C. Yan, J. Tong, W. Cai, and J. Ni, "Timing yield driven clock skew scheduling considering non-gaussian distributions of critical path delays," in *IEEE Proc. DAC*, June 2008, pp. 223–226.

[10] T. H. Cormen, C. E. Leiserson, R. L. Rivest, and C. Stein, *Introduction to algorithms (Second Edition)*. The MIT press, 2001.