Multi-Domain Clock Skew Scheduling

Kaushik Ravindran[1] Andreas Kuehlmann[1,2] Ellen Sentovich[2]

[1] University of California at Berkeley, CA, USA
[2] Cadence Berkeley Labs, Berkeley, CA, USA

Abstract

The application of general clock skew scheduling is practically limited due to the difficulties in implementing a wide spectrum of dedicated clock delays in a reliable manner. This results in a significant limitation of the optimization potential. As an alternative, the application of multiple clocking domains with dedicated phase shifts that are implemented by reliable, possibly expensive design structures can overcome these limitations and substantially increase the implementable optimization potential of clock adjustments. In this paper we present an algorithm for constrained clock skew scheduling which computes for a given number of clocking domains the optimal phase shifts for the domains and the assignment of the individual registers to the domains. For the within-domain latency values, the algorithm can assume a zero-skew clock delivery or apply a user-provided upper bound. Our experiments demonstrate that a constrained clock skew schedule using a few clocking domains combined with small within-domain latency can reliably implement the full sequential optimization potential to date only possible with an unconstrained clock schedule.

1 Introduction

Clock skew scheduling [1], often denoted as "cycle stealing", computes a set of individual delays for the clock signals of the registers and latches of synchronous circuits to minimize the clock period. The schedule globally tunes the latching of the state holding elements such that the delays of their incoming and outgoing paths are maximally balanced. The computed intentional differences in the clock arrival times, also referred to as "useful skew", are then implemented by designing dedicated delays into the clock distribution. In practice, a clock schedule with a large set of arbitrary delays cannot be realized in a reliable manner. This is because the implementation of dedicated delays using additional buffers and interconnections is highly susceptible to within-die variations of process parameters. As a consequence, the practically applicable maximum differences for the clock arrival times are typically restricted to less than 10% of the clock period, which limits the optimization potential of clock skew scheduling.

As an alternative to clock skew scheduling, retiming [2] balances the paths delays by relocating the registers. Although retiming provides a powerful sequential optimization method, its practical use is limited due to the impact on the verification methodology, i.e., equivalence checking and functional simulation. Furthermore, the use of retiming for maximum performance often causes a steep increase in the number of registers [3], requiring a larger effort for clock distribution and resulting in higher power consumption.

Multiple clocking domains are routinely applied in designs to realize several clocking frequencies and also to address specific timing requirements. For example, a special clocking domain that delivers a phase-shifted clock signal to the registers close to the chip inputs and outputs is regularly used to achieve timing closure for ports with extreme constraints on their arrival and required times. In principle, a multi-domain approach could also be used to realize larger clock latency variations for all registers. In combination with a within-domain clock skew scheduling algorithm, they could implement an aggressive sequential optimization that would be impractical with individual delays of register clocks. The motivation behind this approach is based on the fact that large phase shifts between clocking domains can be implemented reliably by using dedicated, possibly expensive circuit components such as "structured clock buffers" [4], adjustments to the PLL circuitry, or simply by deriving the set of phase-shifted domains from a higher frequency clock using different tapping points of a shift register.

In our terminology, we use the term *clock latency* of a register to denote its clock arrival time relative to a common origin of time. Note that the origin can be chosen arbitrarily; different origins simply correspond to different offsets added to all register latencies. *Clock skew* refers to the relative difference of the clock latencies of registers. We use the term *clock phase shift of a domain* to denote an offset of the latency common to all registers of that domain. The *within-domain latency* is defined as the difference between the clock latency of a register and the phase shift of its domain. Thus a zero within-domain latency means that all register latencies of a domain are equal to the phase shift of the domain.

In current design methodologies, the specification of multiple clocking domains is mostly done manually as no design automation support is available. In this paper we present an algorithm for *constrained clock skew scheduling* which computes for a user-given number of clocking domains the optimal phase shifts for the domain clocks and the assignment of the circuit registers to the domains. For the clock distribution within a domain, the algorithm can assume a zero-skew clock delivery or apply a user-provided upper bound for the within-domain latency. Our experiments demonstrate that a clock skew schedule using a few domains combined with a small within-domain latency can reliably implement the full optimization potential of an unconstrained clock schedule.

Our algorithm is based on a branch-and-bound search for the assignment of registers to clocking domains. We apply a satisfiability (SAT) solver based on a problem encoding in conjunctive normal form (CNF) to efficiently drive the search and compactly record parts of the solution space that are guaranteed to contain no solutions better than the current one. The combination of a modern SAT solver [5] with an underlying orthogonal optimization problem provides a powerful mechanism for a hybrid search that has significant potential for other applications in many domains.

ICCAD'03, November 11-13, 2003, San Jose, California, USA.
Copyright 2003 ACM 1-58113-762-1/03/0011 ...$5.00.

For simplicity, our description will be based on circuits which have initially a single clocking domain and include only registers that are triggered at the same clock edge. However, all presented concepts can be extended to more general cases including circuits which have initially multiple, possibly uncorrelated clocking domains and also include level-sensitive latches.

2 Unconstrained Clock Skew Scheduling

In this section, we revisit the algorithmic base for unconstrained clock skew scheduling which is extended to the constrained case in the following section. Given a sequential circuit, the objective of generic clock skew scheduling is to determine an assignment of latencies to registers in order to minimize the clock period, while avoiding clocking hazards [1].

Let $G = (V, E_{setup}, E_{hold})$ denote the timing graph for a sequential circuit. The set of vertices V corresponds to the registers in the circuit and includes a single vertex for all circuit ports. The sets $E_{setup} \subseteq V \times V$ and $E_{hold} \subseteq V \times V$ denote the setup edges and hold edges, respectively. E_{setup} contains for each set of combinational circuit paths between registers (or a port) u and v a directed edge $e = (u,v)$ with weight $w(u,v) = T_{cycle} - d_{max}(u,v) - d_{setup}(v)$, where $d_{max}(u,v)$ represents the longest combinational delay among all paths between u and v, $d_{setup}(v)$ denotes the setup time at v, and T_{cycle} is the cycle period. E_{hold} consists of a set of reversed edges $e_{hold} = (v,u)$ with weight $w(v,u) = d_{min}(u,v) - d_{hold}(v)$, where $d_{min}(u,v)$ is the shortest combinational delay among all paths between u and v and $d_{hold}(v)$ denotes the hold time at v.

By construction G is strongly connected and contains at least one setup edge. We assume that all weights of hold edges are non-negative, i.e., $\forall e \in E_{hold} : w(e) \geq 0$. This restriction simplifies the presentation, however, all algorithms can be extended easily for a relaxed condition that just prohibits negative hold time cycles.

Let $l : V \to \mathbb{R}$ assign a clock latency to each register and $E = E_{setup} \cup E_{hold}$. We want to determine an optimal clock skew schedule $l(v), v \in V$ such that:

$$\forall (u,v) \in E : \; l(v) - l(u) + w(u,v) \geq 0 \qquad (1)$$
$$T_{cycle} \to \min$$

The computed values l give for each register the additional delay (or advance if $l < 0$) of its clock signal such that the circuit can be clocked with the minimum cycle period T_{cycle}. Note that condition (1) ensures that the setup and hold constraints are satisfied as modeled by the edges E_{setup} and E_{hold}, respectively. Figure 1(a) gives an example of a circuit; the corresponding timing graph is given in part (b). The setup and hold times of registers and ports are assumed to be 0. The solid and dashed arcs correspond to the setup edges E_{setup} and hold edges E_{hold}, respectively.

Computation of the optimal clock schedule is closely related to detection of the *critical cycle* which is the structural cycle with the maximum value for $total\_delay/num\_registers$ (ignoring hold edges). Detecting the critical cycle is equivalent to computing the maximum mean cycle (MMC) of a weighted cyclic graph. Our approach is mainly based on Burn's work [6, pp. 42-56], which is to our knowledge one of the fastest practical algorithms for the MMC computation.

Algorithm 1 describes an adaptation of Burn's iterative MMC computation for the given problem. The basic idea is to iteratively decrease T_{cycle} and compute the corresponding clock schedule l at each step until a critical cycle is discovered. First, the algorithm initializes the schedule with all latencies set to 0 and T_{cycle} to the maximum edge delay plus the setup time. At each iteration, the set of edges critical under the current schedule form the critical sub-

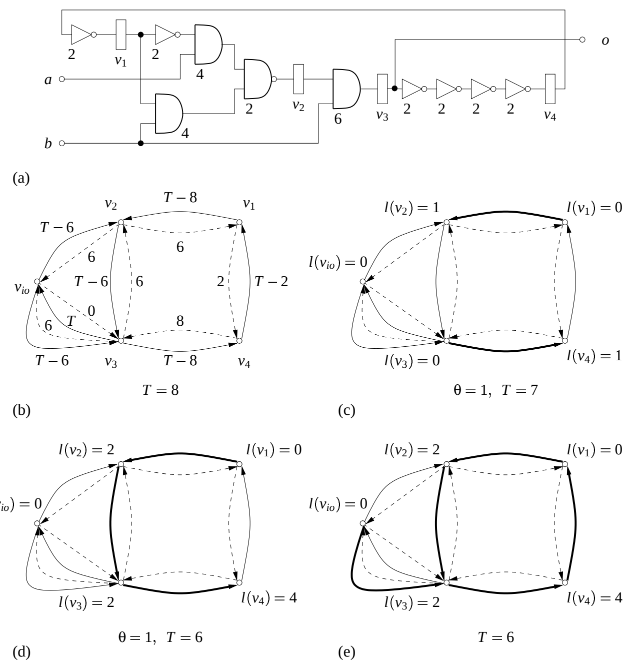

Figure 1: Example for unconstrained clock skew scheduling: (a) circuit structure with gate delays, (b) initial timing graph for $T_{cycle} = 8$, (c-e) timing graphs at several iterations leading to a critical cycle at $T_{cycle} = 6$.

Algorithm 1 UNCONSTRAINEDSKEWSCHEDULING (G)

```
1  foreach $v \in V : l(v) = 0$
2  $T_{cycle} = \max\{d_{max}(u,v) + d_{setup}(v) \mid (u,v) \in E_{setup}\}$
3  while (true)
      // compute critical edges of $G$ yielding critical graph $G'$
4     $E'_{setup} = \{(u,v) \mid (u,v) \in E_{setup} \wedge w(u,v) + l(v) - l(u) = 0\}$
5     $E'_{hold} = \{(u,v) \mid (u,v) \in E_{hold} \wedge w(u,v) + l(v) - l(u) = 0\}$
6     $G' = (V, E'_{setup}, E'_{hold})$
7     if ($G'$ contains cycle with at least one edge $e \in E'_{setup}$)
8        return $l, T_{cycle}$   // critical cycle found
      // compute for each vertex longest distance $\Delta$ from roots in $G'$
9     repeat foreach $v \in V$ until no change
10       $\Delta(v) = \begin{cases} 0 & : \text{if } v \text{ is root of } G' \\ \max\{\{\Delta(u)+1 \mid (u,v) \in E'_{setup}\}, \{\Delta(u) \mid (u,v) \in E'_{hold}\}\} & : \text{otherwise} \end{cases}$
      // compute conservative value for reducing $T_{cycle}$
11    $\theta = \infty$
12    foreach $(u,v) \in E_{setup}$
13       if ($\Delta(u) - \Delta(v) + 1 > 0$)
14          $\theta = \min\{\theta, \frac{w(u,v) - l(u) + l(v)}{\Delta(u) - \Delta(v) + 1}\}$
15    foreach $(u,v) \in E_{hold}$
16       if ($\Delta(u) - \Delta(v) > 0$)
17          $\theta = \min\{\theta, \frac{w(u,v) - l(u) + l(v)}{\Delta(u) - \Delta(v)}\}$
      // update values for $l$ and $T_{cycle}$
18    foreach $v \in V : l(v) = l(v) + \theta \cdot \Delta(v)$
19    $T_{cycle} = T_{cycle} - \theta$
```

graph G' (lines 4–6). If G' contains a cycle with at least one setup edge, the critical cycle has been found and the schedule l and best T_{cycle} are returned (line 8). Otherwise, a conservative decrement θ for the cycle period is computed based on a one-step lookahead from the ends of the critical subgraph (lines 12–17). This calculation and the fast update of the schedule l uses the longest distance $\Delta(v)$ of vertex v from any root of G' (line 10). Note that G' may contain cycles formed by hold edges only. However, the increments of the Δ values along such cycles are 0 and thus convergence is guaranteed. At the end of each iteration the schedule l and T_{cycle} are updated (lines 18,19). Note that when algorithm UNCONSTRAINEDSKEWSCHEDULING terminates, the sum of the edge weights w of the critical cycles is equal to zero.

For the given example in Figure 1, the first iteration of Algorithm 1 results in the graph depicted in part (c) where the two critical edges (v_1,v_2) and (v_3,v_4) with delays equal to the current clock period are highlighted. Now, $\Delta(v_2)=\Delta(v_4)=1$, and $\Delta(v_1)=\Delta(v_3)=0$. The edge (v_2,v_3) determines $\theta=1$ as the maximum amount by which T_{cycle} can be reduced. Thus $T_{cycle}=7$ at the end of the iteration and the vertex latencies are as shown in part (c). The following iteration (d) adds one new critical edge, and $\theta=1$ results in $T_{cycle}=6$. The next iteration (e) finds a critical cycle and returns $T_{cycle}=6$ as the best possible cycle time.

3 Multi-Domain Clock Skew Scheduling

3.1 Problem Formulation

Multi-domain clock skew scheduling of a timing graph $G=(V,E_{setup},E_{hold})$ for a small number of domains imposes additional constraints on the values for clock latencies. For a given number of clocking domains n and a maximum permissible within-domain latency δ, all clock latencies must fit into n value ranges

$$(l(d_1),\ l(d_1)+\delta)$$
$$\dots$$
$$(l(d_n),\ l(d_n)+\delta)$$

where $l(d_i)$ denotes the phase shift of domain i. The objective of multi-domain clock skew scheduling is to determine domain phase shifts $l(d)$ and register latencies $l(v)$ that satisfy the above range constraints and minimize the period T_{cycle}.

For a formal model we extend the definition of the timing graph by introducing a set of domain vertices and conditional edges between registers and domains. Let $G=(V,D,E_{setup},E_{hold},E_{cond})$ denote a multi-domain timing graph where the set of vertices V and sets of edges E_{setup} and E_{hold} have the same definition as before. D represents a set of vertices that correspond to the clocking domains and $E_{cond}=(V\times D)\cup(D\times V)$ are conditional edges associating the registers to the domains. For each pair $v\in V, d\in D$ two locking edges (v,d) and (d,v) are included in E_{cond} with the conditional weights $w(v,d)$ and $w(d,v)$, respectively. Using a set of Boolean variables $x(v,d)\in\{0,1\}$ the weights are defined as follows:

$$w(v,d)=\begin{cases}\delta & : \text{if } x(v,d)=1\\ \infty & : \text{otherwise}\end{cases}$$

$$w(d,v)=\begin{cases}0 & : \text{if } x(v,d)=1\\ \infty & : \text{otherwise}\end{cases}$$

The Boolean attribute $x(v,d)$ is true if register v is assigned to domain d. Let $l(d)$ be the phase shift of domain d. The conditional weights on the edges of E_{cond} ensure that the latency $l(v)$ of register v is bound by $l(d)\le l(v)\le l(d)+\delta$ if v is assigned to d.

Let $E=E_{setup}\cup E_{hold}\cup E_{cond}$. For constrained clock skew scheduling we want to determine a set of register clock latencies $l(v), v\in V$, domain phase shifts $l(d), d\in D$, and assignments of registers to domains $x(v,d)$ such that:

$$\forall(u,v)\in E:\ l(v)-l(u)+w(u,v)\ge 0 \tag{2}$$

$$\forall v\in V:\ \sum_{\forall d}x(v,d)=1 \tag{3}$$

$$T_{cycle}\to\min$$

Similar to the unconstrained case, constraint (2) ensures that all setup and hold time constraints are satisfied and furthermore that all registers assigned to a domain do not exceed the specified maximum within-domain latency. Condition (3) specifies that each register has to be assigned to exactly one domain.

3.2 Base Algorithm

The problem formulation for constrained clock skew scheduling presented in the previous section establishes a Mixed Integer Linear Program (MILP). Unfortunately, the size of practical problem instances involving thousands of registers makes their solution intractable for generic MILP solvers.

Our objective is to efficiently solve the constrained clock skew scheduling problem for a small number of domains. We use a hybrid approach combining a CNF-based SAT solver with a modified version of the scheduling algorithm used in the unconstrained case. We use the SAT solver for enumerating the assignments of registers to domains based on the presented encoding with the Boolean variables x. Boolean constraints are applied to restrict the search to valid assignments according to condition (3) and to incrementally record parts of the solution space that do not contain solutions that are better than the best found thus far. This recording is done by adding conflict clauses to the SAT problem which prevent the solver from revisiting symmetric parts of the solution space.

The basic flow of our approach is shown in Algorithm 2. After initialization on lines 1 and 2, an empty CNF formula ϕ is created with a set of variables for the registers and clocking domains. The procedure INITIALCONSTRAINTS then adds an initial set of Boolean constraints to ϕ that encode valid register-to-domain assignments and represent necessary conditions for the optimization problem. Next the SAT solver is called iteratively to find a complete satisfying assignment x_{SAT} with respect to ϕ. For each generated satisfying assignment, one of the following applies: (1) if the minimum possible period for the configuration is greater than the current best value for T_{cycle}, then this can be detected by a negative cycle in the graph configured by x_{SAT}, or (2) if there are no negative cycles, then T_{cycle} can be further improved using Burn's algorithm.

In the first case the procedure NEGCYCLECONSTRAINTS learns the negative cycles by adding corresponding CNF constraints to ϕ. In the second case the modified critical cycle analysis shown in Algorithm 3 is invoked to further improve T_{cycle} until a tighter critical cycle is reached. Following this optimization step, the procedure TIGHTENINGCONSTRAINTS adds a set of new CNF constraints to ϕ which encode the critical cycles in G and other conditions that are necessary for improving the solution.

The negative and critical cycle constraints jointly ensure that no configuration with previously encountered cycles is revisited. The iteration between the SAT solver and the critical cycle analysis is continued until no new solution can be found. At this point, the values for the last T_{cycle} and l presents the optimal solution for the constrained clock skew scheduling problem.

Algorithm 2 ConstrainedSkewScheduling (G)

```
1  $T_{cycle} = \max\{d_{max}(u,v) + d_{setup}(v) \mid (u,v) \in E_{setup}\}$
2  $\phi$ = empty CNF formula with variables $\{x(v,d) \mid v \in V, d \in D\}$
3  InitialConstraints ($G, \phi$)
4  while (true)
5     $x_{SAT}$ = SatSolve ($\phi$)
6     if ($x_{SAT}$ = UNSAT)
7        return $l, T_{cycle}$
8     if ($G$ contains negative weighted cycle)
9        NegCycleConstraints ($G, T_{cycle}, \phi, x_{SAT}$)
10    else
11       $l, T_{cycle}$ = ConditionalSchedule ($G, x_{SAT}$)
12       TighteningConstraints ($G, T_{cycle}, \phi, x_{SAT}$)
```

The following sections describe the details of the invidual procedures used in Algorithm 2. Section 3.3 outlines how the basic algorithm presented in this section can be further improved.

3.2.1 Algorithm ConditionalSchedule

Algorithm 3 gives the pseudo-code for the modified critical cycle analysis. The assignment to x_{SAT} is used to "activate" some of the conditional edges of E_{cond}, which are then treated in the same way as the edges of E_{hold} in Algorithm 1.

To simplify the presentation of the algorithmic flow, we show all register latencies initialized to 0 and T_{cycle} is set to the maximum combinational delay each time Algorithm 3 is invoked. This ensures a valid starting point for Burn's algorithm. Furthermore, ConditionalSchedule is only applied if G does not contain any negative cycle for the current T_{cycle} – thus it is guaranteed that a schedule with an equal or smaller value for T_{cycle} can be found.

In the actual implementation, the detection of negative cycles on line 8 of Algorithm 2 and the computation of valid register latencies for the given best T_{cycle} is combined using a single analysis run.

Algorithm 3 ConditionalSchedule (G, x_{SAT})

```
1  foreach $v \in V : l(v) = 0$
2  $T_{cycle} = \max\{d_{max}(u,v) + d_{setup}(v) \mid (u,v) \in E_{setup}\}$
3  while (true)
4     $E'_{setup} = \{(u,v) \mid (u,v) \in E_{setup} \wedge w(u,v) + l(v) - l(u) = 0\}$
5     $E'_{hold} = \{(u,v) \mid (u,v) \in E_{hold} \wedge w(u,v) + l(v) - l(u) = 0\}$
6     $E'_{cond} = \{(u,v) \mid (u,v) \in E_{cond} \wedge w(u,v) + l(v) - l(u) = 0\}$
7     $G' = (V, E'_{setup}, E'_{hold}, E'_{cond})$
8     if ($G'$ contains cycle with at least one edge $e \in E'_{setup}$)
9        return $l, T_{cycle}$   // critical cycle found
10    repeat foreach $v \in V$ until no change
11       $\Delta(v) = \begin{cases} 0 & : \text{if } v \text{ is root of } G' \\ \max\{\{\Delta(u)+1 \mid (u,v) \in E'_{setup}\}, \\ \quad \{\Delta(u) \mid (u,v) \in E'_{hold} \cup E'_{cond}\}\} & : \text{otherwise} \end{cases}$
12    $\theta = \infty$
13    foreach $(u,v) \in E_{setup}$
14       if $(\Delta(u) - \Delta(v) + 1 > 0)$
15          $\theta = \min\{\theta, \frac{w(u,v)-l(u)+l(v)}{\Delta(u)-\Delta(v)+1}\}$
16    foreach $(u,v) \in E_{hold} \cup E_{cond}$
17       if $(\Delta(u) - \Delta(v) > 0)$
18          $\theta = \min\{\theta, \frac{w(u,v)-l(u)+l(v)}{\Delta(u)-\Delta(v)}\}$
19    foreach $v \in V : l(v) = l(v) + \theta \cdot \Delta(v)$
20    $T_{cycle} = T_{cycle} - \theta$
```

This provides a good starting point for tightening the critical cycle and thus avoids unnecessary iterations of Burn's algorithm.

3.2.2 Algorithm InitalConstraints

There are two sets of initial constraints for the SAT solver. The first set ensures that each register is assigned to exactly one domain. This is encoded by the following set of CNF clauses for all $v \in V$ and all $d_i, d_j \in D, i \neq j$:

$$\bigvee_{\forall d} x(v,d)$$

$$\overline{x(v,d_i)} \vee \overline{x(v,d_j)}$$

To avoid visiting symmetric domain assignments, one can either encode a corresponding set of CNF constraints that exclude these cases, or define a total ordering of the phase shifts of the individual domains such that:

$$i < j \;\Rightarrow\; l(d_i) \leq l(d_j)$$

In our approach we chose the latter method which can be enforced by adding an edge (d_i, d_{i+1}) to the timing graph with weight $w(d_i, d_{i+1}) = 0$. Algorithm 4 summarizes the generation of initial constraints.

Algorithm 4 InitialConstraints (G, ϕ)

```
1  foreach $v \in V$
2     $\phi = \phi \;\cup\; \{(\bigvee_{\forall d} x(v,d))\}$
3     foreach $d_i, d_j \in D, d_i \neq d_j$
4        $\phi = \phi \;\cup\; \{(\overline{x(v,d_i)} \vee \overline{x(v,d_j)})\}$
5  foreach $i : 0 < i < |D| - 1$
6     $E_{hold} = E_{hold} \;\cup\; \{(d_i, d_{i+1})\}$
7     $w(d_i, d_{i+1}) = 0$
```

In the actual implementation, the edge weights are set to a slightly tighter value $w(d_i, d_{i+1}) = -\delta$ excluding "overlapping" solutions which can occur due to the within-domain latency of up to δ. However, using negative weights for the domain-to-domain edges requires special care for the initialization of the schedule l for Burn's algorithm.

3.2.3 Algorithm NegCycleConstraints

Algorithm NegCycleConstraints is invoked if the graph currently configured cannot implement the best cycle time T_{cycle} found thus far. This situation is detected by finding a cycle in G that contains at least one setup edge and has a non-positive cycle weight. Clearly, any such cycle must contain at least one pair of "active" conditional edges from E_{cond}. This is because a negative cycle just consisting of edges from $E_{setup} \cup E_{hold}$ constrains the minimum value of T_{cycle} independently of the domain assignment and hence would have been detected earlier.

The negative or zero weighted cycles are encoded as CNF conflict clauses and added to ϕ. For example, if a cycle contains the two conditional edges (v_1, d_1) and (v_2, d_2), the clause $\overline{x(v_1,d_1)} \vee \overline{x(v_2,d_2)}$ is added which ensures that in the future both edges are not activated at the same time. Since the number of cycles is generally exponential, our implementation uses a greedy heuristic which encounters all cycles up to four conditional edges. Our experiments show that this scheme provides an efficient means to keep the number of learned clauses small and at the same time ensure quick convergence. Algorithm 5 summarizes the learning of negative cycle constraints.

Algorithm 5 NEGCYCLECONSTRAINTS $(G, T_{cycle}, \phi, x_{SAT})$

```
1  foreach cycle $E_{cycle} \subseteq E_{setup} \cup E_{hold} \cup E_{cond}$
          with $\sum_{e \in E_{cycle}} w(e) \leq 0$ and at least one edge $e \in E_{setup}$
2      $\phi = \phi \cup \{(\bigvee_{\forall e \in E_{cycle} \cap E_{cond}} \overline{x(e)})\}$
```

3.2.4 Algorithm TIGHTENINGCONSTRAINTS

If no negative cycles are encountered algorithm CONDITIONALSCHEDULE is invoked to improve the clock period T_{cycle} and calculate a corresponding schedule l. After this computation, a set of constraints encoding the zero-weight critical cycles are added which prevent revisiting a configuration with an identical critical cycle.

Algorithm 6 TIGHTENINGCONSTRAINTS $(G, T_{cycle}, \phi, x_{SAT})$

```
   // Critical cycle constraints
1  foreach cycle $E_{cycle} \subseteq E_{setup} \cup E_{hold} \cup E_{cond}$
          with $\sum_{e \in E_{cycle}} w(e) = 0$ and at least one edge $e \in E_{setup}$
2      $\phi = \phi \cup \{(\bigvee_{\forall e \in E_{cycle} \cap E_{cond}} \overline{x(e)})\}$
   // Precedence constraints
3  foreach $d_i : 0 < i < |D|$
4      foreach $d_j : i \leq j < |D|$
5          foreach $u, v \in V, u \neq v$
6              $w^P(u,v) = \text{SHORTESTPATHLENGTH}(G, u, v)$
7              if $(w^P(u,v) + (1 + i - j) \cdot \delta \leq 0)$
8                  $\phi = \phi \cup \{(\overline{x(v,d_i)} \vee \overline{x(u,d_j)})\}$
```

Algorithm 6 gives the general computation of the tightening constraints learned when T_{cycle} is improved. The critical cycle constraints are computed on lines 1–2. Lines 3–8 determine the precedence constraints which arise due to the enforceable value ordering of phase shifts between individual domains, similar to the ones generated in procedure INITIALCONSTRAINTS. For example, if the weight of an edge $(u,v) \in E_{setup} \cup E_{hold}$ is less than or equal to 0, condition (2) in the MILP formulation implies $l(v) \geq l(u)$. Because of the assumed ordering of domains this inequality can be learned through the following set of clauses generated for all $d_i, d_j \in D : i < j$:

$$\overline{x(v,d_i)} \vee \overline{x(u,d_j)}$$

These clauses effectively capture the constraint that any satisfying configuration x_{SAT} can only allow assignments $x(u,d_i)$ and $x(v,d_j)$ where $i \leq j$. The condition can be applied more generally by including any path from u to v formed by edges of $E_{setup} \cup E_{hold}$ with negative path weight. When T_{cycle} is decreased, all edges in E_{setup} decrease in weight. The precedence constraint can then be implied on a subset of paths in $G = (V, E_{setup}, E_{hold})$ whose weights become negative. Again, overlapping solutions can be avoided by tightening these constraints by the sum of the bounds on the within-domain latencies. For an efficient generation of precedence constraints, an incremental *All-Pairs-Shortest-Path* algorithm [7] is used to update the shortest path delays between any pair of nodes in G whenever T_{cycle} is improved.

3.2.5 Example

Figure 2 shows the multi-domain timing graphs for two configurations for the example of Figure 1 with two clocking domains and within-domain latency $\delta = 0$. The minimum period with two domains ($T_{cycle} = 7$) is achieved by configuration (b). Note that with three domains the minimum clock period is 6, which is just the solution for the unconstrained case as derived in Figure 1. Indeed, the optimum clock period achieved in the unconstrained case provides a lower bound for the optimum period when the number of domains is constrained.

For the constrained clock skew scheduling example in Figure 2, there are at most $|D|^{|V|} = 2^5 = 32$ different configurations to explore in order to compute the smallest period with two domains. The key for efficiently pruning the search is based on the observation that the period of a particular configuration is limited only by the subset of the register-domain assignments that correspond to critical cycles in the timing graph. For example, after the SAT solver generates the configuration in Figure 2(a), we can avoid any other configuration with either the assignments $x(v_1,d_1) = x(v_2,d_1) = 1$ or $x(v_3,d_2) = x(v_4,d_2) = 1$, since the corresponding critical cycles always limit T_{cycle} to 8. This is encoded by adding the following two CNF conflict clauses to ϕ: $(\overline{x(v_1,d_1)} \vee \overline{x(v_2,d_1)})$ and $(\overline{x(v_3,d_2)} \vee \overline{x(v_4,d_2)})$.

When the configuration in Figure 2(b) is visited, T_{cycle} is updated to 7 and the corresponding critical cycles are learned. In this manner, the algorithm continuously generates valid configurations, prunes the remaining search space by learning critical cycles, and improves T_{cycle} until the SAT solver is unable to find another satisfying register-domain assignment.

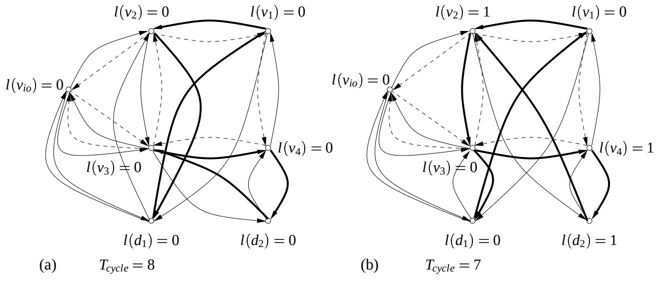

Figure 2: Two register-domain assignments for the circuit from Figure 1 optimized for two clocking domains: (a) 1. configuration: $\{x(v_1,d_1) = x(v_2,d_1) = x(v_{io},d_1) = 1, x(v_3,d_2) = x(v_4,d_2) = 1\}$; Critical cycles: (d_1,v_1,v_2), (d_2,v_3,v_4); $T_{cycle} = 8$, (b) 2. configuration: $\{x(v_1,d_1) = x(v_3,d_1) = x(v_{io},d_1) = 1, x(v_2,d_2) = x(v_4,d_2) = 1\}$; Critical cycles: (d_1,v_1,v_2,v_3), (d_2,v_2,v_3,v_4); $T_{cycle} = 7$.

3.3 Further Algorithmic Improvements

The base algorithm works efficiently for larger circuits up to three clocking domains. However, in the case of more clocking domains, the exponential nature of the problem may cause long runtimes. Note that the search can be interrupted at any point; all encountered solutions are valid; thus the last one can serve as suboptimal schedule.

We observed that the runtime can be reduced significantly when the search is composed of the following three phases: (1) initial estimation of a good solution based on binning of the unconstrained clock schedule, (2) gradual improvement of this solution based on a limited search space that preserves the ordering of the unconstrained schedule and (3) final full search with temporary limitation removed. When artificially over-constraining the search during the first two phases, the solver converged significantly faster. Furthermore, many negative cycle and tightening constraints can be added for the final full search which in turn improves its run time.

Algorithm 7 gives an overview of this refined algorithm; the following sections elaborate on the details of the first two phases.

Algorithm 7 REFINEDCONSTRAINEDSKEWSCHEDULING (G)

```
1  $T_{cycle} = \max\{w(u,v) \mid (u,v) \in E_{setup}\}$
   // Phase 1: Use constraints from binning unconstrained schedule
2  $T_{cycle} =$ INITIALSOLUTION ($G, T_{cycle}$)
   // Phase 2: Use constraints to preserve partial ordering of
   // unconstrained schedule
3  $T_{cycle} =$ PARTIALORDERSOLUTION ($G, T_{cycle}$)
   // Phase 3: Full search
4  $l, T_{cycle} =$ FULLSOLUTION ($G, T_{cycle}$)
5  **return** $l, T_{cycle}$
```

Algorithm 8 PARTIALORDERINGCONSTRAINTS (G, ϕ)

```
1 $l, T'_{cycle} =$ UNCONSTRAINEDSKEWSCHEDULING ($G$)
2 **foreach** $(u,v) \in E$
3   **if** $l(u) > l(v)$
4     **foreach** $d_i, d_j \in D,\ i < j$
5       $\phi = \phi \ \cup \ \{(\overline{x(u,d_i)} \vee \overline{x(v,d_j)})\}$
```

3.3.1 Initial Solution

A simple approach to derive a good initial value for T_{cycle} is to solve the unconstrained clock skew scheduling problem for G using Algorithm 1 and then distribute the resulting latencies greedily into $|D|$ bins of size $\frac{l_{max}-l_{min}}{|D|}$, where l_{max} and l_{min} represent the maximum and minimum latency of the unconstrained schedule, respectively. The actual clock period for this solution is computed by translating the latency binning into corresponding register-domain edges in G followed by single run of Algorithm 3.

Furthermore, the best solution for $|D|-1$ domains provides an upper bound for T_{cycle} with $|D|$ domains. Since the algorithm runs significantly faster for fewer clocking domains, a previously computed solution for fewer domains can be used as an alternative starting point if it's value for T_{cycle} is smaller than the one from binning.

3.3.2 Partial Ordering Heuristic

After the initialization step, we can introduce a set of partial ordering constraints on the domain assignments of registers. The partial ordering helps in trimming the search space, but may in turn also exclude the optimum solution. The heuristic assumes that if in the unconstrained skew schedule register u has a latency greater than that of register v, then there exists an optimum constrained skew schedule that has u assigned to a domain equal to or higher than v. The constraint generation for this heuristic is detailed in Algorithm 8. The SAT-based search is then applied to this over-constrained problem. The resulting clock period is a good starting point for the final run of the solver to compute the exact optimum.

The partial ordering heuristic appears to be exact for small circuits; however, one can show that the ordering constraints may exclude better solutions as illustrated by a simple counter-example given in Figure 3. For this graph, the optimum T_{cycle} is 4 for the unconstrained case. The latencies at each vertex to achieve this period are shown in the figure. Note that the constrained version of the problem will require at least 8 clocking domains to achieve this period.

Let d_v denote the domain that vertex v is assigned to. Allowing only two clocking domains and zero within-domain latency (i.e., $\delta = 0$), the path from v_1 to v_4 restricts the optimal period achievable with two domains to 8. The phase shifts of the individual domains are $l(d_1) = 0$ and $l(d_2) = 2$. The domain assignments of the individual vertices are $d_{v_1} = d_{v_3} = d_1$ and $d_{v_2} = d_{v_4} = d_2$, with the remaining vertices assigned to either domain.

To preserve the latency ordering of the unconstrained schedule, the partial ordering heuristic requires the constraints $d_{v_1} \leq d_{v_2} \leq d_{v_3} \leq d_{v_4}$. However, the constraint $d_{v_2} \leq d_{v_3}$ clearly violates all optimal domain-register assignments. The application of partial ordering constraints results in a period of 10, which is sub-optimal.

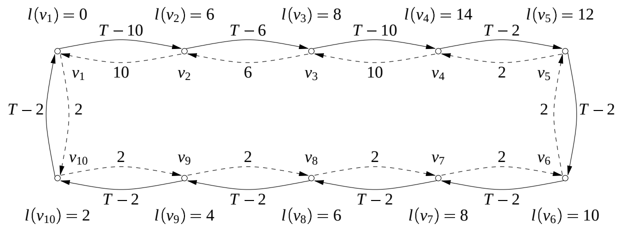

Figure 3: Example to demonstrate that the partial ordering heuristic may over-constrain the solution space and thus lead to a suboptimal schedule and cycle period.

4 Previous Work

The original definition of the clock skew optimization problem was given by Fishburn [1]. He formulated the unconstrained clock skew scheduling problem as a linear program, similar to the specification given in Equation (1) earlier in this paper. Deokar and Sapatnekar [8] translated this problem to a graph-theoretical setting. The idea of their approach is to perform a binary search for the smallest period T_{cycle} characterized by the absence of a negative weight cycle in the constraint graph.

Albrecht, et al. [9] proposed a solution for a concurrent cycle time and slack optimization based on the parametric shortest path algorithm [10]. They extend the basic framework to balance slacks on all circuit paths in order to restrict uncertainties in the implementation of delay buffers and clock tree synthesis.

To our knowledge, the practically fastest algorithm applicable for unconstrained clock skew scheduling has been published by Burns [6]. Here, the computation of the optimal clock schedule is related to the detection of the critical cycle. Burn's algorithm provides a fast method to identify the critical cycle and distribute the smallest amount of latency necessary for the registers to attain the optimal cycle period. It is this algorithm that we use in the inner loop of our approach for constrained clock skew scheduling.

There is no work that proposes a solution to the constrained clock skew scheduling problem that is considered in this paper. Toyonaga, et al. [11] proposed an algorithm based on simulated annealing to synthesize a semi-synchronous clock tree optimizing a function of clock period and area. However, the focus there was on generating a feasible clock tree as opposed to finding an optimal solution for the cycle time in the clock skew scheduling problem.

The work that comes closest to our problem of constrained clock skew scheduling was published by Singh and Brown [12]. The authors consider the problem of clock skew scheduling using a fixed, small set of clocking domains with pre-determined phase shifts to be implemented in FPGA's. The solution is a slight modification of the unconstrained clock skew scheduling method and uses an iterative Bellman-Ford algorithm. However, the quality of their results is only as good as the predefined phase shift values. In contrast, our work considers the more general problem of multi-domain clock skew scheduling where the phase shifts of the domain can be modified for optimal performance.

5 Experimental Results

To evaluate the algorithm and observe its performance on practical designs, we have created a prototype implementation using the presented methods on top of the SAT solver Chaff [5]. Our benchmark suite consisted of the 31 ISCAS89 sequential circuits and 8 industrial designs. The ISCAS benchmarks were technology mapped through SIS [13] using the library *lib2.genlib*. The industrial circuits were generated by a commercial logic synthesis tool using industrial ASIC libraries. We applied the REFINEDCONSTRAINEDSKEWSCHEDULING algorithm to determine the minimum feasible clock period with up to four clocking domains and a within-domain latency of up to 10% of the initial cycle period corresponding to the longest combinational delay including setup time. The experiments were conducted on a PentiumIII 2GHz processor with 2GB RAM running Linux. The results are reported in Tables 1 and 2. Table 3 presents the run times and the number of SAT solver iterations for the industrial circuits.

Columns 2 and 3 in Tables 1 and 2 give the number of vertices and edges in the timing graph. Column 4 reports the optimal clock period T^{∞}_{cycle} achievable through clock skew scheduling with an unconstrained number of domains. This is a lower bound. Column 5 shows the initial cycle time for the circuit corresponding to a zero-skew schedule which is simply the longest combinational path delay. This is an upper bound and corresponds to a configuration with one domain and zero within-domain latency, denoted as $T^{1,0}_{cycle}$. The subsequent columns report the optimum clock period computed by our algorithm for a bounded number of domains and within-domain latency of 0%, 5%, and 10% of $T^{1,0}_{cycle}$. The numbers reported in a column with a label of $T^{x,y}_{cycle}$ indicate the optimum cycle time for x clock domains and a within-domain latency of $\delta = y\% \cdot T^{1,0}_{cycle}$. We highlighted all dominating solutions, i.e., the non-bold entries reflect solutions for which there exist an equivalent or better one with fewer domains or a smaller value for the within-domain latency.

The algorithm easily optimized all ISCAS benchmarks – for a majority of instances, the optimum was achieved with less than three domains. The total run time on the first 27 ISCAS benchmarks was less than a minute. The last four circuits took only slightly longer. The results reported in Table 2 indicate a considerable cycle time improvement in most of the industrial circuits. Even with two domains and a within-domain latency of $\delta = 5\% \times T^{1,0}_{cycle}$, the industrial benchmarks achieved on average 90% of the optimum cycle time (T_{∞}) possible. With three domains and $5\% \times T^{1,0}_{cycle}$ latency, these benchmarks come as close as 95% of the optimum solution. In fact, for six of the eight industrial benchmarks, we achieve the lowest clock period possible through clock skew scheduling with four domains; four among these reached the optimum with three domains. The run times were reasonable, given the high complexity of the problem. For design D2, with four domains and no within-domain skew, we terminated the algorithm after 20 hours; it had achieved a cycle time of 15.89 as shown. We re-ran that case with a tight initial guess (from a previous run) and the algorithm terminated in 17 hours with the optimum cycle time, which for that case is 15.41.

Figure 4 tracks the progress of the three phases of the algorithm over time for seven industrial designs constrained by four clocking domains and zero within-domain latency. Circuit $D4$ is not included because the optimum period was trivially computed and there was no iterative improvement. The execution time and clock period have been normalized: 100% corresponds to the clock period of the zero-skew schedule $T^{1,0}_{cycle}$. The curves are not a comparison of relative progress - rather, they capture the rate at which T_{cycle} is improved. The three phases of the REFINEDCONSTRAINEDSKEWSCHEDULING algorithm are indicated by a dotted segment denoting the initial solution, a solid line representing phase 2 where partial ordering constraints are introduced and a dashed line denoting the last phase where a full search is performed. From the graph, it can be observed that the cycle times improve most dramatically early in the algorithm. Hence, with limited CPU time, one can stop the algorithm shortly into phase 3 and still expect very good improvements in cycle time.

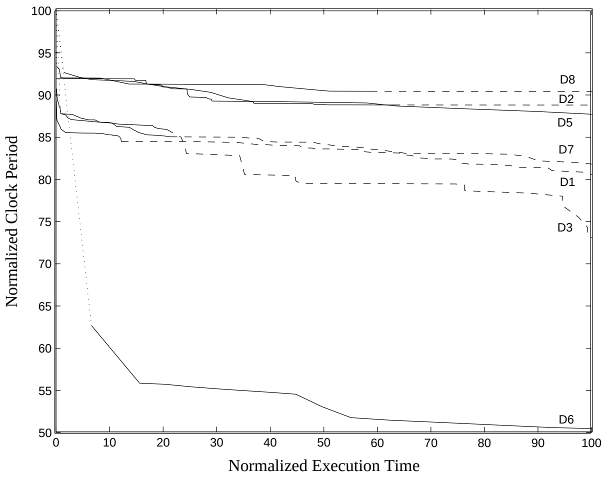

Figure 4: Graphical tracking of the algorithm's progress over time for seven industrial designs optmized for four clocking domains and zero within-domain latency. The dotted, solid, and dashed segments denote phases 1, 2, and 3, respectively of Algorithm 7.

6 Conclusions

In this paper we presented an algorithm for constrained clock skew scheduling which computes for a fixed number of clocking domains the optimal phase shifts for the domains and the assignment of the individual registers to the domains. For the within-domain latency values, the algorithm can assume a zero-skew clock delivery or apply a user-provided upper bound. Our algorithm is based on a branch-and-bound enumeration of the register-to-domain assignments. We apply a CNF-based SAT solver for the enumeration process and use learning of CNF constraints to prevent invalid register assignments and to record sets of inferior solutions which should not be revisited. The actual evaluation of each assignment is performed by an incremental maximum mean cycle analysis on the constraint graph.

Our experiments indicate, that despite the potential complexity of the enumeration process, the presented algorithm is efficient for modestly sized circuits and works even for circuits with several thousand registers reasonably fast. Furthermore, our results show that a constrained clock skew schedule with few clocking domains and zero or 5% within-domain latency can in most cases achieve the optimal cycle time dictated by the critical cycle of the circuit. The resulting multi-domain solution provides a significant advantage over the corresponding unconstrained clock skew schedule which typically has large variations of register latencies and thus cannot be implemented in a reliable manner.

Design	# FF	# Edges	T^{∞}_{cycle}	$\delta = 0\% \times T^{1,0}_{cycle}$				$\delta = 5\% \times T^{1,0}_{cycle}$				$\delta = 10\% \times T^{1,0}_{cycle}$			
				$T^{1,0}_{cycle}$	$T^{2,0}_{cycle}$	$T^{3,0}_{cycle}$	$T^{4,0}_{cycle}$	$T^{1,5}_{cycle}$	$T^{2,5}_{cycle}$	$T^{3,5}_{cycle}$	$T^{4,5}_{cycle}$	$T^{1,10}_{cycle}$	$T^{2,10}_{cycle}$	$T^{3,10}_{cycle}$	$T^{4,10}_{cycle}$
s1196	19	365	22.28	**22.28**	22.28	22.28	22.28	22.28	22.28	22.28	22.28	22.28	22.28	22.28	22.28
s1423	76	2235	73.13	**79.04**	**75.35**	**73.82**	**73.13**	**75.08**	**73.13**	73.13	73.13	**73.13**	73.13	73.13	73.13
s298	16	86	10.79	**13.05**	**11.36**	**10.79**	10.79	**12.40**	**10.79**	10.79	10.79	**11.75**	10.79	10.79	10.79
s420	18	146	21.13	**22.06**	**21.13**	21.13	21.13	**21.13**	21.13	21.13	21.13	21.13	21.13	21.13	21.13
s526	23	167	11.22	**13.48**	**11.79**	**11.22**	11.22	**12.81**	**11.22**	11.22	11.22	**12.13**	11.22	11.22	11.22
s641	21	486	29.51	**29.98**	**29.51**	29.51	29.51	**29.51**	29.51	29.51	29.51	29.51	29.51	29.51	29.51
s832	7	213	16.22	**16.22**	16.22	16.22	16.22	16.22	16.22	16.22	16.22	16.22	16.22	16.22	16.22
s953	8	94	15.36	**17.32**	**15.77**	**15.36**	15.36	**16.48**	**15.36**	15.36	15.36	**15.59**	15.36	15.36	15.36
s1238	19	365	24.33	**26.15**	**24.36**	**24.33**	24.33	**24.84**	**24.33**	24.33	24.33	**24.33**	24.33	24.33	24.33
s1488	8	266	23.18	**23.58**	**23.18**	23.18	23.18	**23.18**	23.18	23.18	23.18	23.18	23.18	23.18	23.18
s208	10	70	9.91	**10.84**	**9.91**	9.91	9.91	**10.30**	9.91	9.91	9.91	**9.91**	9.91	9.91	9.91
s344	17	121	13.14	**15.57**	**14.32**	**13.19**	**13.14**	**14.79**	**13.54**	**13.14**	13.14	**14.16**	**13.14**	13.14	13.14
s382	23	175	9.63	**14.06**	**11.55**	**9.77**	**9.63**	**13.36**	**10.71**	**9.63**	9.63	**12.66**	**10.01**	9.63	9.63
s386	8	129	9.60	**10.56**	**9.97**	**9.60**	9.60	**10.03**	**9.62**	9.60	9.60	**9.74**	**9.60**	9.60	9.60
s444	23	175	8.10	**13.92**	**10.84**	**9.55**	**8.88**	**13.22**	**10.14**	**8.86**	**8.18**	**12.53**	**9.45**	**8.18**	**8.15**
s526n	23	167	11.31	**13.57**	**11.91**	**11.31**	11.31	**12.89**	**11.31**	11.31	11.31	**12.21**	11.31	11.31	11.31
s713	21	486	30.58	**30.58**	30.58	30.58	30.58	30.58	30.58	30.58	30.58	30.58	30.58	30.58	30.58
s838	34	298	44.66	**45.59**	**44.66**	44.66	44.66	**44.66**	44.66	44.66	44.66	44.66	44.66	44.66	44.66
s1494	8	266	23.85	**24.71**	**23.85**	23.85	23.85	**23.85**	23.85	23.85	23.85	23.85	23.85	23.85	23.85
s27	5	21	5.06	**6.58**	**5.75**	**5.06**	5.06	**6.25**	**5.42**	5.06	5.06	**5.92**	**5.09**	5.06	5.06
s349	17	121	13.51	**15.89**	**14.72**	**13.60**	**13.51**	**15.09**	**13.93**	**13.51**	13.51	**14.51**	**13.51**	13.51	13.51
s400	23	175	9.89	**14.59**	**11.62**	**10.15**	**10.08**	**13.86**	**10.89**	**9.89**	9.89	**13.13**	**10.16**	9.89	9.89
s510	8	103	14.29	**14.29**	14.29	14.29	14.29	14.29	14.29	14.29	14.29	14.29	14.29	14.29	14.29
s5378	165	2180	22.89	**28.84**	**25.93**	**23.17**	**22.94**	**27.40**	**24.21**	**22.89**	22.89	**25.96**	**22.89**	22.88	22.88
s820	7	213	16.74	**16.74**	16.74	16.74	16.74	16.74	16.74	16.74	16.74	16.74	16.74	16.74	16.74
s9234	140	2226	33.77	**34.75**	**33.96**	**33.77**	33.77	**33.77**	33.77	33.77	33.77	33.77	33.77	33.77	33.77
s13207	471	3885	53.36	**57.35**	**55.13**	**53.46**	**53.36**	**54.48**	**53.36**	53.36	53.36	**53.36**	53.36	53.36	53.36
s15850	565	16375	85.27	**98.38**	**92.24**	**88.70**	**88.42**	**93.46**	**87.04**	**85.27**	85.27	**88.54**	**85.27**	85.27	85.27
s35932	1442	6128	286.32	**289.47**	**286.32**	286.32	286.32	**286.32**	286.32	286.32	286.32	286.32	286.32	286.32	286.32
s38584	1451	17900	286.62	**288.60**	**287.04**	**286.62**	286.62	**286.62**	286.62	286.62	286.62	286.62	286.62	286.62	286.62
s38417	1465	31980	86.19	**87.76**	**86.19**	86.19	86.19	**86.19**	86.19	86.19	86.19	86.19	86.19	86.19	86.19

Table 1: Results of multiple-domain clock skew optimization on ISCAS89 sequential benchmark circuits.

Design	# FF	# Edges	T^{∞}_{cycle}	$\delta = 0\% \times T^{1,0}_{cycle}$				$\delta = 5\% \times T^{1,0}_{cycle}$				$\delta = 10\% \times T^{1,0}_{cycle}$			
				$T^{1,0}_{cycle}$	$T^{2,0}_{cycle}$	$T^{3,0}_{cycle}$	$T^{4,0}_{cycle}$	$T^{1,5}_{cycle}$	$T^{2,5}_{cycle}$	$T^{3,5}_{cycle}$	$T^{4,5}_{cycle}$	$T^{1,10}_{cycle}$	$T^{2,10}_{cycle}$	$T^{3,10}_{cycle}$	$T^{4,10}_{cycle}$
D1	2245	46048	2.79	**3.78**	**3.67**	**3.22**	**3.04**	**3.65**	**3.53**	**3.08**	**2.98**	**3.56**	**3.34**	**2.90**	**2.82**
D2	2921	250737	15.26	**17.94**	**17.04**	**16.46**	**15.89**	**17.05**	**15.62**	**15.26**	15.26	**16.14**	**15.26**	15.26	15.26
D3	6316	21006	4.41	**6.48**	**5.95**	**5.58**	**4.73**	**6.16**	**5.66**	**4.98**	**4.43**	**5.89**	**5.30**	**4.74**	**4.42**
D4	2694	16518	3.69	**3.69**	3.69	3.69	3.69	3.69	3.69	3.69	3.69	3.69	3.69	3.69	3.69
D5	3065	18030	2.95	**3.86**	**3.72**	**3.60**	**3.38**	**3.72**	**3.55**	**3.43**	**3.21**	**3.61**	**3.47**	**3.27**	**3.00**
D6	574	2294	4.55	**9.03**	**5.06**	**4.67**	**4.55**	**8.58**	**4.84**	**4.55**	4.55	**8.16**	**4.63**	4.55	4.55
D7	852	47370	16.34	**20.47**	**18.92**	**17.61**	**16.71**	**19.45**	**17.66**	**16.37**	**16.34**	**18.43**	**16.34**	16.34	16.34
D8	2368	9181	1.74	**2.03**	**1.97**	**1.84**	**1.83**	**1.96**	**1.90**	**1.80**	**1.78**	**1.90**	**1.79**	**1.78**	**1.74**

Table 2: Results of multiple-domain clock skew optimization on some industrial circuits.

Design	# FF	# Edges	T^{∞}_{cycle}	$\delta = 0\% \times T^{1,0}_{cycle}$				$\delta = 5\% \times T^{1,0}_{cycle}$				$\delta = 10\% \times T^{1,0}_{cycle}$			
				$T^{1,0}_{cycle}$	$T^{2,0}_{cycle}$	$T^{3,0}_{cycle}$	$T^{4,0}_{cycle}$	$T^{1,5}_{cycle}$	$T^{2,5}_{cycle}$	$T^{3,5}_{cycle}$	$T^{4,5}_{cycle}$	$T^{1,10}_{cycle}$	$T^{2,10}_{cycle}$	$T^{3,10}_{cycle}$	$T^{4,10}_{cycle}$
D1	2245	46048	5s	1s/2	5s/6	4m/50	200m/321	1s/2	5s/7	5m/44	355m/854	1s/2	20s/7	8m/39	372m/839
D2	2921	250737	1m	20s/2	6m/56	120m/1735	1200m/2327	21s/2	28s/2	2m/3	2m/2	20s/3	35s/2	2m/2	2m/2
D3	6316	21006	45s	5s/2	33s/25	2m/36	450m/2234	6s/2	16s/5	4m/19	195m/622	6s/2	20s/5	1m/6	4m/8
D4	2694	16518	1s	1s/2	1s/2	1s/2	1s/2	1s/2	1s/2	1s/2	1s/2	1s/2	1s/2	1s/2	1s/2
D5	3065	18030	1m	3s/2	4s/3	7s/5	2m/52	3s/2	4s/3	14s/8	5m/69	3s/2	6s/3	40s/6	10m/71
D6	574	2294	5s	1s/2	1s/3	2s/12	4s/17	1s/2	1s/3	2s/7	2s/3	1s/2	1s/3	2s/4	2s/3
D7	852	47370	1m	7s/2	19s/16	2m/128	10m/224	7s/2	12s/2	1m/4	3m/4	9s/2	45s/2	1m/2	1m/2
D8	2368	9181	25s	1s/2	2s/3	30s/37	30m/902	1s/2	2s/5	30s/37	2m/69	1s/2	3s/5	5s/6	15s/7

Table 3: Program run times / Number of SAT solver invocations for the industrial circuits in Table 2.

Acknowledgments

The authors would like to thank Marc Heyberger for providing the delay data for the industrial circuits and Christoph Albrecht for valuable feedback during the preparation of the final manuscript.

References

[1] J. P. Fishburn, "Clock skew optimization," *IEEE Transactions on Computers*, vol. 39, pp. 945–951, July 1990.

[2] C. Leiserson and J. Saxe, "Optimizing synchronous systems," *Journal of VLSI and Computer Systems*, vol. 1, pp. 41–67, January 1983.

[3] G. Even, I. Y. Spillinger, and L. Stok, "Retiming revisited and reversed," *IEEE Transactions on Computer-Aided Design*, vol. 15, pp. 348–357, March 1996.

[4] K. M. Carrig, "Chip clocking effect on performace for IBM's SA-27E ASIC technology," *IBM Micronews*, vol. 6, no. 3, pp. 12–16, 2000.

[5] M. W. Moskewicz, C. F. Madigan, Y. Zhao, L. Zhang, and S. Malik, "Chaff: Engineering an efficient SAT solver," in *Proceedings of the 38th ACM/IEEE Design Automation Conference*, (Las Vegas, Nevada), pp. 530–535, June 2001.

[6] S. M. Burns, *Performance Analysis and Optimization of Asynchronous Circuits*. PhD thesis, California Institute of Technology, Pasadena, CA, December 1991.

[7] T. H. Cormen, C. E. Leiserson, and R. L. Rivest, *Introduction to Algorithms*. The MIT Electr. Eng. and Computer Science Series, MIT Press/McGraw Hill, 1990.

[8] R. B. Deokar and S. S. Sapatnekar, "A graph-theoretic approach to clock skew optimization," in *Proceedings of the IEEE International Symposium on Circuits and Systems*, pp. 407–410, 1994.

[9] C. Albrecht, B. Korte, J. Schietke, and J. Vygen, "Cycle time and slack optimization for vlsi-chips," in *Digest of Technical Papers of the IEEE International Conference on Computer-Aided Design*, pp. 232–238, November 1999.

[10] N. E. Young, R. E. Tarjan, and J. B. Orlin, "Faster parametric shortest path and minimum balance algorithms," *Networks*, vol. 21, no. 2, pp. 205–221, 1991.

[11] M. Toyonaga, K. Kurokawa, T. Yasui, and A. Takahashi, "A practical clock tree synthesis for semi-synchronous circuits," in *Proceedings of the 2000 International Symposium on Physical Design*, pp. 159–164, 2000.

[12] D. P. Singh and S. D. Brown, "Constrained clock shifting for field programmable gate arrays," in *Proc. of the ACM Internat. Symp. on FPGAs*, pp. 121–126, 2002.

[13] E. Sentovich, K. Singh, C. Moon, H. Savoj, R. Brayton, and A. Sangiovanni-Vincentelli, "Sequential circuit design using synthesis and optimization," in *Proc. of the Internat. Conf. on Computer Design (ICCD'92)*, pp. 328–333, 1992.