Clock Skew Scheduling with Delay Padding for Prescribed Skew Domains*

Chuan Lin

Magma Design Automation
Santa Clara, CA 95054
clin@magma-da.com

Hai Zhou
EECS Department
Northwestern University
Evanston, IL 60208
haizhou@eecs.northwestern.edu

Abstract

Clock skew scheduling is a technique that intentionally introduces skews to memory elements to improve the performance of a sequential circuit. It was shown in [21] that the full optimization potential of clock skew scheduling can be reliably implemented using a few skew domains. In this paper we present an optimal skew scheduling algorithm for sequential circuits with flip-flops. Given a finite set of prescribed skew domains, the algorithm finds a domain assignment for each flip-flop such that the clock period is minimized with possible delay padding. Experimental results validate the efficiency of our algorithm and show 17% improvement on average in clock period.

1 Introduction

In a sequential circuit, due to the differences of interconnect delays in the clock distribution network, clock signals do not arrive at all flip-flops at the same time. The consequent differences in clock arrival times are also known as *clock skews*. Since the setup and hold constraints of a sequential circuit are complicated by clock skews, an approach that has been followed by [12, 13, 25, 19, 20] is to deliberately design the clock distribution network so as to ensure zero clock skew.

Clock skew scheduling [10], on the other hand, views clock skew as a manageable resource rather than a liability. It intentionally introduces skews to flip-flops to improve the circuit performance. The designated skews are then implemented by specific layout of the clock distribution network. However, in practice, a skew schedule with a large set of arbitrary values cannot be realized in a reliable manner. This is because the implementation of dedicated delays using additional buffers and interconnects is highly susceptible to intra-die variations of process parameters.

Instead of tuning clock skews of flip-flops, retiming [15] physically relocates flip-flops to balance the delays without changing the functionality of the circuit. It was observed in [10] that retiming and clock skew scheduling are discrete and continuous optimizations with the same effect. The equivalence between retiming and skew has been used in prior research [17, 16, 6, 22]. Although retiming is a powerful sequential optimization technique, its practical usage 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 may cause a steep increase in the number of flip-flops [9], requiring a larger effort for clock distribution and resulting in higher power consumption.

Recently, multi-domain clock skew scheduling was proposed in [21]. 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 flip-flops 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. The motivation behind the multi-domain skew scheduling 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" [5], 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 flip-flop. In [21], Ravindran *et al.* showed that a clock skew schedule using a few domains combined with a small within-domain latency can reliably implement the full optimization potential of clock skew scheduling. They proposed an algorithm based on a branch-and-bound search to assign flip-flops to clock domains and used a modified Burns's algorithm [4] to compute the skews.

For a user-given number of domains, the algorithm in [21] computed the optimal skew for each domain. However, the user had no control on the distribution of the domains. Furthermore, the algorithm did not consider delay padding [23], which is a technique that fixes hold violations by inserting extra delays on short paths without increasing the delay on any long path. In other words, the clock period obtained by the algorithm in [21] may be sub-optimal if delay padding is allowed, as demonstrated in [14, 8, 24].

In this paper we formulate the clock skew scheduling problem on a user-given finite set of prescribed clock domains. For example, one can require the skews of flip-flops to be either zero or half the clock period. One can also choose clock domains based on the results of the algorithm in [21]. We propose a polynomial-time algorithm that finds an optimal domain assignment for each flip-flop such that the clock period is minimized with possible delay padding. The obtained skew schedule respects the user requirement. We then consider how to insert extra paddings such that both setup and hold constraints are satisfied under the minimal clock period. We show that the existence of such a padding solution is guaranteed, and present an approach to find a padding by network flow technique. Experimental results confirm the efficiency of our algorithm.

The rest of the paper is organized as follows. Section 2 presents a motivation example and the problem formulation. Notations and constraints are explained in Section 3. Our algorithm is elaborated in Section 4. Experimental results are presented in Section 5, followed by conclusions in Sec-

*This work was done at Northwestern University and supported by the NSF under CCR-0238484.

tion 6.

2 Motivation and problem formulation

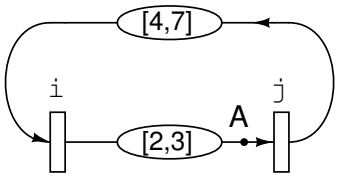

Figure 1: Effect of clock skew scheduling and delay padding on circuit performance.

We use Figure 1 as an example to illustrate the effect of clock skew scheduling and delay padding on circuit performance. In this example, we have two flip-flops i and j, both triggered at the falling edge of a clock. The ellipses between the flip-flops represent the combinational blocks with their minimum and maximum delays. Suppose that the setup and hold times are all zero, and that the skew to each flip-flop can be either zero or half the period. In order for the circuit to operate under a given period T, the following conditions must be satisfied, where the first two are from the setup constraint and the other two are from the hold constraint.

$$\begin{aligned} skew(i) + 3 - T &\le skew(j) \\ skew(j) + 7 - T &\le skew(i) \\ skew(i) + 2 &\ge skew(j) \\ skew(j) + 4 &\ge skew(i) \end{aligned}$$

Depending on the skew assignment, we have three cases. Firstly, $skew(i) = skew(j)$, which leads to $T \ge 7$. Secondly, $skew(i) = T/2$ and $skew(j) = 0$, we have $6 \le T \le 8$. For $skew(i) = 0$ and $skew(j) = T/2$, the setup constraint requires $T \ge 14$ while the hold constraint needs $T \le 4$. In other words, there is no such a T satisfying both constraints. However, if we insert an extra delay of 5 at point A, then the minimum and maximum delays from i to j become 7 and 8 respectively, which results in a feasible period of $T = 14$.

The above example reveals two things. Firstly, assigning skews to flip-flops may help to reduce the period of a circuit. On the other hand, cautions should be taken when choosing the skews since the circuit may end up having no feasible period at all. Secondly, delay padding can be used to remedy a skew assignment so that it permits feasible periods after the insertion of extra delays. In some cases, delay padding is required to reach a smaller period. For the above example, if the minimum delay from j to i is not 4 but 2, then a delay of 1 needs to be inserted on the minimum delay path to obtain the optimal period 6. This motivates us to solve a problem formulated as follows.

Problem 1 (Optimal Skew Scheduling)
Given a sequential circuit and a finite set of prescribed skew domains, find a domain assignment for each flip-flop such that the circuit satisfies both setup and hold constraints with possible delay padding under the minimal clock period.

For simplicity, we assume that flip-flops are triggered at the same clock edge of a single phase clock. However, the proposed approach can be extended to multiple clock phases.

3 Notations and constraints

Suppose that we are given N skew domains. Without loss of generality, we assume that the skew values of the N domains are $s_0T, s_1T, ..., s_{N-1}T$ with respect to the period T, where $s_0, s_1, ..., s_{N-1}$ are constants between 0 and 1 in the increasing order, i.e., $0 = s_0 < s_1 < ... < s_{N-1} < 1$. In particular, the domains are *evenly distributed* if $s_n = \frac{n}{N}T$, for all $0 \le n \le N-1$.

A directed graph $G = (V, E)$ is used to represent a sequential circuit, where V is the set of gates and flip-flops, and E is the set of interconnects. Each gate $v \in V$ has a maximum delay $D(v)$ and a minimum delay $d(v)$. Each interconnect $e \in E$ has a delay $w(e)$. Delay padding increases $w(e)$ by inserting extra delays on e. For any combinational path $p = u \rightsquigarrow v$, we use $D(p)$ to represent the maximum delay along p without padding, which is the sum of the constituent interconnect delays and maximum gate delays, except for $D(u)$. The minimum delay along p without padding is denoted by $d(p)$. With extra paddings on p, the maximum and minimum delays become $\Delta(p)$ and $\delta(p)$ respectively. Note that

$$\Delta(p) - \delta(p) = D(p) - d(p).$$

We also construct a timing graph $G_t = (V_t, E_t)$ of G as follows. Let $V_t \subset V$ be the set of flip-flops in the circuit. An edge (i, j) is introduced in E_t if there is a combinational path $p \in G$ from flip-flop i to flip-flop j. We define

$$D(i,j) \stackrel{\triangle}{=} \max_{p \in G: i \rightsquigarrow j} D(p), \qquad \Delta(i,j) \stackrel{\triangle}{=} \max_{p \in G: i \rightsquigarrow j} \Delta(p)$$

$$d(i,j) \stackrel{\triangle}{=} \min_{p \in G: i \rightsquigarrow j} d(p), \qquad \delta(i,j) \stackrel{\triangle}{=} \min_{p \in G: i \rightsquigarrow j} \delta(p)$$

In other words, $D(i, j)$ and $d(i, j)$ are the maximum and minimum combinational delays from i to j without padding, respectively. They become $\Delta(i, j)$ and $\delta(i, j)$ with padding. We say that p is a *long* path from i to j if $\Delta(p) = \Delta(i, j)$; p is a *short* path if $\delta(p) = \delta(i, j)$. For all $i \in V_t$, we use $X(i)$ and $H(i)$ to denote the setup and hold times at flip-flop i respectively. A label $l : V_t \to \{0, ..., N-1\}$ is introduced to represent the index of the domain that a flip-flop is assigned to.

Using these notations, the setup and hold constraints under a given period T can be formulated as follows.

$$0 \le l(i) \le N-1, \quad \forall i \in V_t \tag{1}$$

$$s_{l(i)}T + \Delta(i,j) - s_{l(j)}T \le T - X(j), \quad \forall (i,j) \in E_t \tag{2}$$

$$s_{l(i)}T + \delta(i,j) - s_{l(j)}T \ge H(j), \quad \forall (i,j) \in E_t \tag{3}$$

A partial order ($\le$) can be defined between two labels l and l' as follows.

$$l \le l' \stackrel{\triangle}{=} l(i) \le l'(i), \quad \forall i \in V_t$$

We say that T is a *feasible period* if and only if we can find an l and a delay padding such that (1)-(3) are satisfied under T. The optimal skew scheduling problem asks for the minimal feasible T, with possible padding insertion.

4 Algorithm

In order to find the minimum feasible T, we first compute a lower bound T_{lb} for it in Section 4.1. Then we observe that for any l satisfying (1), since $s_n < 1, \forall n \in [0, N-1]$, we have $1 + s_{l(j)} - s_{l(i)} > 0, \forall (i, j) \in E_t$. Thus, the setup constraint (2) can be written as $T \ge \big(\Delta(i,j) + X(j)\big) / \big(1 + s_{l(j)} - s_{l(i)}\big)$. Together with $\Delta(i, j) \ge D(i, j)$, it characterizes a minimal period T_S under setup constraint only. We propose an algorithm in Section 4.2 to compute $T^* = \max(T_{lb}, T_S)$ and the corresponding l^*. We then show in Section 4.3 that there always exists a padding solution such that both setup and hold constraints are satisfied under T^* and l^*. Therefore,

T^* and l^* are the solution to the optimal skew scheduling problem.

4.1 Lower bound for feasible period

Consider any combinational path p from $i \in V_t$ to $j \in V_t$. Since $\Delta(i,j) \geq \Delta(p)$ and $\delta(i,j) \leq \delta(p)$, from the definition of $\Delta(i,j)$ and $\delta(i,j)$, (2)-(3) imply that

$$s_{l(i)}T + \Delta(p) - s_{l(j)}T \leq T - X(j)$$
$$s_{l(i)}T + \delta(p) - s_{l(j)}T \geq H(j)$$

Subtracting the second one from the first yields $\Delta(p) - \delta(p) \leq T - X(j) - H(j)$. Since $\Delta(p) - \delta(p) = D(p) - d(p)$, we have a lower bound for T in the next lemma.

Lemma 1 *A feasible clock period T must satisfy*

$$T \quad \geq \quad T_{lb} \stackrel{\triangle}{=} \max_{(i,j)\in E_t, p\in E: i\rightsquigarrow j} \big(D(p) - d(p) + X(j) + H(j)\big)$$

To compute T_{lb}, let $\theta(v)$ be the difference between the maximum and the minimum delays at gate v, defined as

$$\theta(v) \stackrel{\triangle}{=} D(v) - d(v), \quad \forall v \in V - V_t \tag{4}$$

For flip-flop j, we define

$$\theta(j) \stackrel{\triangle}{=} X(j) + H(j), \quad \forall j \in V_t$$

Let $\Theta(v)$ be the length of the longest combinational path terminating at v with respect to θ, i.e.,

$$\Theta(v) \stackrel{\triangle}{=} \max_{combinational\ p\in E:\rightsquigarrow v} \theta(p), \qquad \forall v \in V \tag{5}$$

Then, finding T_{lb} is equivalent to computing the maximum $\Theta(j)$, $\forall j \in V_t$, which can be done by longest path computation in $O(|E| + |V|\log|V|)$ time [7].

4.2 Minimum period under setup constraint

We use T_S to denote the minimal period under which the setup constraint is satisfied without padding, i.e.,

$$0 \leq l(i) \leq N-1, \quad \forall i \in V_t \tag{1}$$
$$s_{l(i)}T + D(i,j) - s_{l(j)}T \leq T - X(j), \quad \forall (i,j) \in E_t \tag{6}$$

Let $T^* = \max(T_{lb}, T_S)$. We use l^* to denote a domain assignment satisfying (1) and (6) under T^*. For example, the corresponding domain assignment under T_S is such an l^*.

To find an l^*, we start with $l(i) = 0$, $\forall i \in V$, and obtain $T = \max_{(i,j)\in E_t} \frac{D(i,j)+X(j)}{1+s_{l(j)}-s_{l(i)}}$ since $1 + s_{l(j)} - s_{l(i)} > 0$, $\forall (i,j) \in E_t$. If $T \leq T_{lb}$, then we have $T^* = T_{lb}$, and thus the current l is an l^*. Otherwise, let $(x,y) \in E_t$ be the edge that determines T, i.e., $T = \frac{D(x,y)+X(y)}{1+s_{l(y)}-s_{l(x)}}$. Suppose $T > T^*$, it follows that $D(x,y)+X(y) > \big(1+s_{l(y)}-s_{l(x)}\big)T^*$. On the other hand, T^* and l^* should satisfy the setup constraint on (x,y), i.e., $\big(1 + s_{l^*(y)} - s_{l^*(x)}\big)T^* \geq D(x,y) + X(y)$. As a result, we have

$$s_{l^*(y)} - s_{l^*(x)} > s_{l(y)} - s_{l(x)} \tag{7}$$

We can move l closer to l^* by increasing $l(y)$. The amount of increase should only be 1 since we do not want to over-adjust l. This process is iterated until the optimality of T is certified. We present the pseudocode in Figure 2.

MinPeriod(G_t, T^*, l^*)

```
T*, l ← ∞, 0;
While ( (T* > T_lb  ∧  (∀i ∈ V_t : l(i) < N)) do
  Extract (x, y) from E_t with max (D(x,y)+X(y)) / (1+s_{l(y)}−s_{l(x)});
  T ← (D(x,y)+X(y)) / (1+s_{l(y)}−s_{l(x)});
  If (T < T*) then
      T*, l* ← max(T, T_lb), l;
  l(y) ← l(y) + 1;
```

Figure 2: Pseudocode of minimum period computation.

The next lemma states an invariant that is kept throughout "MinPeriod".

Lemma 2 *$l \leq l^*$ is kept during the execution of "MinPeriod" in Figure 2 before we reach an l^*.*

Proof: First of all, $l \leq l^*$ before we enter the while loop since we initialize $l(i) = 0$, $\forall i \in V_t$. What remains is to show that $l \leq l^*$ is preserved after $l(y)$ is increased by 1 for some $y \in V_t$ until T^* is reached.

Assume that $T > T^*$, thus (7) is true. Since $l(x) \leq l^*(x)$ and $l(y) \leq l^*(y)$ due to $l \leq l^*$, we have $l(y) < l^*(y)$, otherwise $l(y) = l^*(y)$, which leads to $l(x) > l^*(x)$, which is a contradiction. Therefore, $l \leq l^*$ is kept after the increase of $l(y)$. The lemma is true. □

The correctness and complexity of "MinPeriod" is given in the following theorem.

Theorem 1 *The procedure "MinPeriod" in Figure 2 will terminate in $O(N|V_t|B_t\log|E_t|)$ time, where B_t is the maximum incoming and outgoing degrees of the vertices in V_t. Upon termination, it gives $T^* = \max(T_{lb}, T_S)$, and an l^* satisfying (1) and (6) under T^*.*

Proof: The outer while loop cannot be executed more than $(N-1)|V_t|$ times since each traversal results in an increase in $l(y)$ for some $y \in V_t$. The complexity of extracting the edge (x,y) in E_t with the maximum $\frac{D(x,y)+X(y)}{1+s_{l(y)}-s_{l(x)}}$ is $O(\log|E_t|)$ if we choose Fibonacci heap [7]. After $l(y)$ is increased by 1, we need to adjust the values of $\frac{D(i,y)+X(y)}{1+s_{l(y)}-s_{l(i)}}$ for all incoming edges $(i,y) \in E_t$ to y, and the values of $\frac{D(y,j)+X(j)}{1+s_{l(j)}-s_{l(y)}}$ for all outgoing edges $(y,j) \in E_t$ from y, which takes $O(B_t\log|E_t|)$ time. Therefore, the overall complexity is $O(N|V_t|B_t\log|E_t|)$.

When it terminates, we have either $T^* = T_{lb}$ or $l(y) = N > l^*(y)$ for some $y \in V_t$. In the first case, we have $T_S \leq T_{lb}$, thus $T^* = \max(T_{lb}, T_S)$ is true. By Lemma 2, the second case implies that we have already reached an l^* and went beyond it. Therefore, the obtained T^* is T_S. Since $T^* > T_{lb}$, $T^* = \max(T_{lb}, T_S)$ is also true. In both cases, the obtained l^* satisfies (1) and (6) under T^*. □

Since padding increases $D(i,j)$, we have $\Delta(i,j) \geq D(i,j)$, $\forall (i,j) \in E_t$. Therefore, T^* is a lower bound for any feasible period with padding.

4.3 Padding for hold constraint

Given T^* and l^* from "MinPeriod", we will check the hold constraint (3) without padding. If we have a hold violation at some $j \in V_t$, it means that there is a short path p from $i \in V_t$ to j such that $s_{l^*(i)}T^* + d(p) - s_{l^*(j)}T^* < H(j)$. Intuitively, if p has an interconnect that does not lie on

any long path, then we can insert extra delays on it to fix the hold violation. The following lemma [23] provides the condition under which the existence of such an interconnect is guaranteed.

Lemma 3 *Let p be a short path to $j \in V_t$, where a hold violation occurs under T^* and l^*. There exists an edge e on p such that e does not lie on any long path if $D(p) - d(p) \leq T^* - X(j) - H(j)$.*

The next result is a corollary of the above lemma.

Corollary 3.1 *There exists a delay padding solution under T^* and l^* satisfying both setup and hold constraints.*

Proof: Since $T^* \geq T_{lb}$, the definition of T_{lb} in Lemma 1 implies that $D(p) - d(p) \leq T^* - X(j) - H(j)$ for all path p to j, $\forall j \in V_t$. Consequently, if a hold violation occurs at j under T^* and l^*, Lemma 3 ensures that we can successively identify interconnects for padding insertion without affecting any long path until the hold violation is fixed. □

Based on Corollary 3.1 and the fact that T^* is a lower bound for any feasible period with padding, we know that T^* and l^* are the solution to the optimal skew scheduling problem.

To find a padding solution, we can treat flip-flop outputs as primary inputs (PIs) and flip-flop inputs as primary outputs (POs), and find a padding for each individual combinational component. To ease the presentation, we will focus on padding for a combinational component $G_c = (V_c, E_c) \subseteq G$ under T^* and l^*.

For each gate $v \in G_c$, we use $A(v)$ and $a(v)$ to denote the latest and the earliest arrival times at the output of v, which are the longest and the shortest combinational delays from PIs to v, respectively. Let $p(u, v)$ be the padding on $(u, v) \in G_c$. The problem (denoted as MP) of finding a minimum delay padding under T^* and l^* can be formulated as follows [23].

$$
\begin{aligned}
MP: \quad & \text{Minimize} \sum_{(u,v)\in G_c} p(u,v) \\
a(i) &= s_{l^*(i)}T^*, \quad \forall i \in PI && (8)\\
a(v) &\leq a(u) + d(v) + w(u,v) + p(u,v), \\
&\qquad \forall (u,v) \in G_c && (9)\\
a(j) &\geq H(j) + s_{l^*(j)}T^*, \quad \forall j \in PO && (10)\\
A(i) &= s_{l^*(i)}T^*, \quad \forall i \in PI && (11)\\
A(v) &\geq A(u) + D(v) + w(u,v) + p(u,v), \\
&\qquad \forall (u,v) \in G_c && (12)\\
A(j) &\leq T^* - X(j) + s_{l^*(j)}T^*, \quad \forall j \in PO && (13)\\
p(u,v) &\geq 0, \quad \forall (u,v) \in G_c && (14)
\end{aligned}
$$

The conditions (8)-(9) characterize the earliest arrival times at gate outputs. (10) is the hold constraint at POs. The conditions (11)-(12) characterize the latest arrival times. (13) is the setup constraint. Inequality (14) enforces non-negative padding.

We say that p is a *feasible padding* if and only if there exist a and A such that (8)-(14) are satisfied under T^* and l^*. The *feasible region* of MP contains all the feasible paddings. Since both the objective and the constraints are linear, MP can be solved by any linear programming solver.

We next describe an approach to find a "reasonably good" padding using network flow technique, which is more efficient than linear programming.

First of all, we observe that subtracting (9) from (12) yields

$$A(v) - a(v) \geq A(u) - a(u) + \theta(v), \qquad \forall (u,v) \in G_c$$

where $\theta(v) = D(v) - d(v)$ is defined in (4). Therefore, $A(v) - a(v) \geq \Theta(v)$ by the definition of $\Theta(v)$ in (5). Given a feasible padding, we can insert extra delays on each edge such that (12) becomes an equality. When (12) is an equality, we have

$$A(v) - a(v) = \max_{(u,v)\in G_c} A(u) - a(u) + \theta(v), \quad \forall v \in G_c$$

As a result, there exists a combinational path p to v such that $A(v) - a(v) = \theta(p) \leq \Theta(v)$. Together with $A(v) - a(v) \geq \Theta(v)$, we have $A(v) - a(v) = \Theta(v)$, $\forall v \in G_c$. Since $\Theta(v)$ can be obtained by longest path computation with respect to θ, we can replace $A(v)$ by $\Theta(v) + a(v)$ in the conditions (11)-(13) to simplify the problem.

Lemma 4 *The minimum padding problem MP with (12) being an equality is equivalent to the following problem (EMP):*

$$
\begin{aligned}
EMP: \quad & \text{Minimize} \sum_{(u,v)\in G_c} p(u,v) \\
a(i) &= s_{l^*(i)}T^*, \quad \forall i \in PI, && (8)\\
a(v) + \Theta(v) &= a(u) + \Theta(u) + D(v) + w(u,v) + p(u,v), \\
&\qquad \forall (u,v) \in G_c && (15)\\
a(j) &\geq H(j) + s_{l^*(j)}T^*, \quad \forall j \in PO && (10)\\
a(j) + \Theta(j) &\leq T^* - X(j) + s_{l^*(j)}T^*, \quad \forall j \in PO && (16)\\
p(u,v) &\geq 0, \quad \forall (u,v) \in G_c && (14)
\end{aligned}
$$

Proof: Since both MP and EMP have the same objective, what remains is to show that they have the same feasible region when (12) is an equality.

Suppose that p is a feasible padding to EMP. Since $\Theta(v) \geq \Theta(u) + \theta(v)$ by (5), we know that (15) implies (9). For $i \in PI$, since $\Theta(i) = 0$ and (8), we have $A(i) = \Theta(i) + a(i) = s_{l^*(i)}T^*$, which is (11). Given that $A(v) = \Theta(v) + a(v)$, (15) is an equality form of (12). (16) is the same as (13) since $A(j) = \Theta(j) + a(j)$. Therefore, p is also a feasible padding to MP when (12) is an equality. Similarly, if p is feasible to MP with (12) being an equality, p is also feasible to EMP, which concludes our proof. □

Note that EMP is the dual of a min-cost flow problem [3]. Let $\bar{p}$ be an optimal solution to EMP. Since $\bar{p}$ satisfies the setup constraint, any p with $p(u, v) \leq \bar{p}(u, v)$, $\forall (u, v) \in G_c$, should also satisfy the setup constraint. Therefore, we use $\bar{p}$ as an upper bound for p and solve the minimum padding problem with the hold constraint only. This is formulated as the following problem (BMP).

$$
\begin{aligned}
BMP: \quad & \text{Minimize} \sum_{(u,v)\in G_c} p(u,v) \\
a(i) &= s_{l^*(i)}T^*, \quad \forall i \in PI && (8)\\
a(v) &\leq a(u) + d(v) + w(u,v) + p(u,v), \\
&\qquad \forall (u,v) \in G_c && (9)\\
a(j) &\geq H(j) + s_{l^*(j)}T^*, \quad \forall j \in PO && (10)\\
0 &\leq p(u,v) \leq \bar{p}(u,v), \quad \forall (u,v) \in G_c && (17)
\end{aligned}
$$

Note that BMP is the dual of a convex-cost flow problem [2]. Both EMP and BMP can be solved in polynomial time bounded by $O\big(|V_c||E_c| \log(|V_c|^2/|E_c|) \log(|V_c|T^*)\big)$. The next theorem provides the condition under which an optimal solution to BMP is also an optimal solution to MP.

Theorem 2 *If MP has an optimal solution p^* such that $p^*(u, v) \leq \bar{p}(u, v)$ for all $(u, v) \in G_c$, then an optimal solution to BMP is one such p^*.*

Proof: Since $p^* \leq \bar{p}$, p^* is feasible to BMP. Given that the feasible region of MP contains the feasible region of

BMP, p^* is an optimal solution to BMP. Therefore, any optimal solution to BMP has the same amount of padding as p^*, and hence is an optimal solution to MP. □

When the condition in Theorem 2 does not hold, solving BMP only gives a feasible padding. However, our experiments show that the feasible padding we obtained is close to the minimum padding.

Our algorithm for optimal skew scheduling is presented in Figure 3. It first applies "MinPeriod" to compute T^* and l^*. Then, it finds a padding solution for each combinational component under T^* and l^*. The overall complexity is $O\big(N|V_t|B_t \log|E_t| + |V||E| \log(|V|^2/|E|) \log(|V|T^*)\big)$ by Theorem 1 and the complexity for solving EMP and BMP.

```
Input : A circuit G = (V, E) and N skew domains.
Output: Optimal period T* under domain
        assignment l* and delay padding p.

   Construct timing graph G_t from G;
   MinPeriod(G_t, T*, l*);
   For each combinational component G_c ⊆ G do
      Find padding p by solving BMP for G_c;
   Return T*, l* and p;
```

Figure 3: Pseudocode of optimal skew scheduling algorithm.

5 Experimental results

We implemented the algorithm in a PC with a 2.4 GHz Xeon CPU, 512 KB 2nd level cache memory and 1GB RAM. We used the linear cost-scaling algorithm by Goldberg [11] to solve EMP. We also adapted it to convex cost case to solve BMP. The detailed procedure of adaption is shown in [2]. Our test files were generated from the ISCAS-89 benchmark suite using ASTRA [22]. Each gate was assigned a maximum delay equal to the number of fanouts or an upper bound 100, whichever is smaller. The minimum delay was equal to the maximum delay. To ease the presentation, interconnect delays were set to zero. Note that we did not ignore interconnect delays. They were handled uniformly in our algorithms. The circuits used are summarized in Table 1.

Table 1: Sequential circuits from ISCAS-89

Circuit	$\|V\|$	$\|E\|$	$\|V_t\|$	$\|E_t\|$
s838	618	959	172	3160
s1196	560	1053	31	94
s1423	896	1407	239	11320
s5378	3080	4561	301	3068
s9234	6198	8593	601	11035
s9234.1	6176	8588	579	10843
s13207.1	9337	12702	1386	9803
s15850	11449	15408	1677	63184
s15850.1	11348	15370	1576	52504
s35932	21880	34403	5815	32049
s38417	25058	35012	2879	59054
s38584	26651	40431	7398	101700

Without loss of generality, we used four evenly distributed skew domains, i.e., $s_n = nT/4$, $0 \le n \le 3$. Setup and hold times of each flip-flop were set to 2. Thus, $T_{lb} = 4$. The results are reported in Table 2. Column "$|s_n|$", $0 \le n \le 3$, lists the number of flip-flops that are assigned to domain s_n in the obtained optimal skew schedule. The sum of the setup time and the maximum combinational delay of the original circuit is listed in Column "T_{ub}". It is an upper bound for T^*. The computed minimal period is listed in column "T^*". The running time of "MinPeriod" for finding T^* is reported in column "t(sec)" in seconds. The improvement ratio $(T_{ub} - T^*)/T_{ub}$ for each circuit is listed in column "impr%". We obtain the arithmetic (geometric) mean of all the ratios in row "arith" ("geo"). Once T^* and l^* are obtained, we solve EMP and BMP to get a padding solution under T^* and l^*. We compare the solution with the minimum padding computed by MOSEK solver [1]. The amount of padding and running time are listed in column "padding" and "time(sec)" respectively.

We can see from Table 2 that, except for "s5378", all circuits have smaller periods after skew scheduling with possible padding. The improvement could be significant, e.g., 42.9% in "s15850". The average improvement is 16.9%. In addition, "MinPeriod" is efficient. It takes only 0.59 second for the largest circuit "s38584". Although the padding solution to BMP is about 1.5X the minimum padding, the actual area overhead will be reasonably small since the delay padding is amortized over the whole circuit. For running time comparison, solving BMP by network flow technique is much more efficient than solving MP by MOSEK. The average speed-up is more than one order of magnitude.

To see how skew scheduling helps to improve a retiming solution, we used the algorithm in [18] to compute a minimum period retiming under the setup and hold constraints, and then applied our algorithm on the retimed circuit. The results are reported in Table 3. We use "s838′" to denote the optimal retiming of "s838", and so on for other optimal retimings. Column "[18]" lists the minimal period computed by the retiming algorithm in [18], where we use "NO" to indicate that there is no feasible retiming. For circuits without feasible retiming, we obtained their min-period retimings under setup constraint only. We highlight the cases where the periods are further improved by our skew scheduling algorithm. Comparing Table 3 with Table 2, we observe the following results.

Firstly, half of the circuits have their minimal periods further reduced after skew scheduling with possible padding insertion. The improvement is 7.3% on average and up to 27.8%. This is significant considering that our algorithm is applied after a minimum period retiming. Two circuits "s13207.1" and "s38584" do not even have a feasible retiming due to the discrete nature of retiming. It happens when there exist reconvergent paths where the retiming requirements from different paths contradict each other. However, by skew scheduling and delay padding, we are able to find the minimal periods for them.

Secondly, it appears that the optimal skew schedule for the retimed circuit uses less number of domains. The number of flip-flops that are assigned to domains other than s_0 is also reduced. In other words, applying skew scheduling after retiming improves the implementability of the obtained optimal skew schedule.

Thirdly, in all test cases, except for "s838", applying skew scheduling on the retimed circuit results in less amount of delay padding than the original circuit. In addition, the solution to BMP is closer to (about 1.1X) the minimum padding of the retimed circuit.

6 Conclusion

We present a polynomial time algorithm that finds an optimal skew schedule over a finite set of prescribed skew domains such that the period is minimized with possible delay padding. We show that the existence of a padding solution under the minimal period is guaranteed and propose an approach to find a padding solution by network flow technique. Experimental results validate the efficiency of our algorithm.

Table 2: Optimal skew schedule with delay padding

Circuit	$\|s_0\|$	$\|s_1\|$	$\|s_2\|$	$\|s_3\|$	T_{ub}	T^*	impr%	t(sec)	padding		time(sec)	
									MOSEK	ours	MOSEK	ours
s838	94	28	43	7	96.0	65.6	31.6%	0.01	56.8	56.8	0.20	0.01
s1196	30	1	0	0	102.0	100.0	2.0%	0.00	25.0	87.0	0.45	0.01
s1423	150	70	19	0	334.0	256.0	23.4%	0.07	966.0	1425.4	0.49	0.03
s5378	301	0	0	0	94.0	94.0	0.0%	0.01	0.0	0.0	2.65	0.09
s9234	597	4	0	0	180.0	164.0	8.9%	0.04	38.0	75.0	5.57	0.37
s9234.1	575	4	0	0	180.0	164.0	8.9%	0.05	38.0	75.0	5.72	0.43
s13207.1	1381	5	0	0	288.0	272.0	5.6%	0.02	546.0	562.0	9.25	0.88
s15850	1102	291	158	126	374.0	213.7	42.9%	0.29	12441.4	14305.6	11.34	1.28
s15850.1	1362	210	4	0	374.0	292.0	21.9%	0.28	1862.0	2829.0	13.63	1.49
s35932	4951	864	0	0	140.0	126.0	10.0%	0.16	15840.0	22896.0	13.21	2.83
s38417	1945	430	230	274	222.0	128.0	42.3%	0.28	6612.0	8224.0	38.55	3.44
s38584	6042	1355	1	0	308.0	290.0	5.8%	0.59	22932.5	23920.0	19.19	5.14
arith							16.9%		1	1.53X	15.6X	1
geo							18.4%		1	1.42X	12.5X	1

Table 3: Effect of retiming on skew scheduling

Circuit	$\|s_0\|$	$\|s_1\|$	$\|s_2\|$	$\|s_3\|$	[18]	T^*	impr%	t(sec)	padding		time(sec)	
									MOSEK	ours	MOSEK	ours
s838′	54	38	40	34	82.0	**59.2**	27.8%	0.01	147.6	158.4	0.22	0.01
s1196′	35	0	0	0	100.0	100.0	0.0%	0.00	0.0	0.0	0.46	0.02
s1423′	239	17	0	0	282.0	**256.0**	9.2%	0.05	514.0	999.7	0.46	0.03
s5378′	301	0	0	0	94.0	94.0	0.0%	0.01	0.0	0.0	2.63	0.09
s9234′	603	0	0	0	164.0	164.0	0.0%	0.05	0.0	0.0	6.13	0.36
s9234.1′	581	0	0	0	164.0	164.0	0.0%	0.05	0.0	0.0	6.71	0.43
s13207.1′	1126	5	0	0	NO	**272.0**	n/a	0.02	36.0	52.0	9.17	0.79
s15850′	928	398	201	4	212.0	**156.0**	26.4%	0.10	9479.0	10783.8	9.80	1.40
s15850.1′	1515	0	0	0	292.0	292.0	0.0%	0.17	0.0	0.0	10.77	1.20
s35932′	4321	864	0	0	140.0	**126.0**	10.0%	0.14	14752.0	21808.0	13.83	2.69
s38417′	4688	0	0	0	122.0	122.0	0.0%	0.03	0.0	0.0	27.97	3.95
s38584′	4159	1355	1	0	NO	**290.0**	n/a	0.66	18284.0	19265.7	18.28	4.32
arith							7.3%		1	1.19X	13.9X	1
geo							8.0%		1	1.16X	11.7X	1

References

[1] The mosek optimization tools version 3.2 user's manual and reference. [online] http://www.mosek.com.

[2] R. K. Ahuja, D. S. Hochbaum, and J. B. Orlin. Solving the convex cost integer dual network flow problem. *Management Science*, 49(7):950–964, July 2003.

[3] R. K. Ahuja, T. L. Magnanti, and J. B. Orlin. *Network Flows: Theory, Algorithms, and Application*. Prentice Hall, 1993.

[4] S. M. Burns. *Performance analysis and optimization of asynchronous circuits*. PhD thesis, California Institute of Technology, Computer Science Department, 1991.

[5] K. M. Carrig. Chip clocking effect on performace for ibms sa-27e asic. *IBM Micronews*, 6(3):12–16, 2000.

[6] L.-F. Chao and E. H.-M. Sha. Retiming and clock skew for synchronous systems. In *ISCAS*, pages 1.283–1.286, 1994.

[7] T. H. Cormen, C. E. Leiserson, and R. H. Rivest. *Introduction to Algorithms*. MIT Press, 1989.

[8] D. Ernst, N. S. Kim, S. Das, S. Pant, R. Rao, T. Pham, C. Ziesler, D. Blaauw, T. Austin, K. Flautner, and T. Mudge. Razor: A low-power pipeline based on circuit-level timing speculation. In *International symposium of microarchitecture*, 2003.

[9] G. Even, I. Y. Spillinger, and L. Stok. Retiming revisited and reversed. *IEEE TCAD*, 15(3):348–357, March 1996.

[10] J. P. Fishburn. Clock skew optimization. *IEEE Transactions on Computers*, 39:945–951, July 1990.

[11] A. V. Goldberg. An efficient implementation of a scaling minimum-cost flow algorithm. *Journal of Algorithms*, 22:1–29, 1997.

[12] M. A. B. Jackson, A. Srinivasan, and E. S. Kuh. Clock routing for high-performance ic's. In *DAC*, 1990.

[13] A. Kahng, J. Cong, and G. Robins. High-performance clock routing based on recursive geometric matching. In *DAC*, 1991.

[14] Y. Kohira and A. Takahashi. Clock period minimization method of semi-synchronous circuits by delay insertion. *IEICE Trans. Fundamentals*, E88-A(4):892–898, April 2005.

[15] C. E. Leiserson, F. M. Rose, and J. B. Saxe. Optimizing Synchronous Circuitry by Retiming. In *Advanced Research in VLSI: Proc. of the Third Caltech Conf.*, pages 86–116, Rockville, MD, 1983. Computer Science Press.

[16] B. Lockyear and C. Ebeling. Minimizing the effect of clock skew via circuit retiming. Technical Report UW-CSE-93-05-04, Dept. of Computer Science and Engineering, University of Washington, Seattle, 1993.

[17] H.-G. Martin. Retiming by combination of relocation and clock delay adjustment. In *Euro-DAC*, pages 384–389, 1993.

[18] M. C. Papaefthymiou. Asymptotically efficient retiming under setup and hold constraints. In *ICCAD*, 1998.

[19] S. Pullela, N. Menezes, and L. T. Pillage. Reliable nonzero clock skew trees using wire width optimization. In *DAC*, 1993.

[20] S. Pullela, N. Menezes, and L. T. Pillage. Skew and delay optimization for reliable buffered clock trees. In *ICCAD*, 1993.

[21] K. Ravindran, A. Kuehlmann, and E. Sentovich. Multi-domain clock skew scheduling. In *ICCAD*, pages 801–808, 2003.

[22] S. S. Sapatnekar and R. B. Deokar. Utilizing the retiming-skew equivalence in a practical algorithm for retiming large circuits. *IEEE TCAD*, 15(10):1237–1248, October 1996.

[23] N. V. Shenoy, R. K. Brayton, and A. L. Sangiovanni-Vincentelli. Minimum padding to satisfy short path constraints. In *ICCAD*, 1993.

[24] B. Taskin and I. Kourtev. Delay insertion method in clock skew scheduling. In *ISPD*, pages 47–54, 2005.

[25] R.-S. Tsay. Exact zero skew. In *ICCAD*, pages 336–339, 1991.