第18回　回路とシステム
軽井沢ワークショップ
The 18th Workshop on Circuits and Systems
in Karuizawa, April 25-26, 2005

Practical Fast Clock-Schedule Design Algorithms

Atsushi Takahashi
Department of Communications and Integrated Systems, Tokyo Institute of Technology

Abstract: This paper introduces a practical clock-scheduling engine. The minimum feasible clock-period is obtained by using a modified Bellman-Ford shortest path algorithm. Then an optimum cost clock-schedule is obtained by using a bipartite matching algorithm. It also provides useful information to circuit synthesis tools. The experiment to a circuit with about 10000 registers and 100000 signal paths shows that a result is obtained within a few minutes. The computation time is almost linear to the circuit size in practice.

1 Introduction

Due to the difficulty of clock-distribution and the increase of instantaneous power consumption and noise, the circuit design methodology without zero clock-skew is becoming popular. However, a layout-aware clock-schedule design methodology that takes the feasibility and efficiency of layout into account is not established yet. In this paper, we propose two efficient algorithms that determine the minimum feasible clock-period and a minimum cost schedule which are essential to obtain a layout-aware clock-schedule.

In the circuit synthesis tools, clock-scheduling is executed many times as a subroutine. Therefore, the computation time is the prime subject. Moreover, it should handle various constraints derived from various design styles such as the minimum and maximum clock-timing for each register, latch based circuitry, and multi-clock-cycle signal paths, etc.

The minimum feasible clock-period in terms of clock-scheduling is better to be known in clock-scheduling, since it is an important index of the constraints. The minimum feasible clock-period is obtained by linear programming [4, 13], by graph-theoretic approaches with binary search [15, 14, 3, 11], or by graph-theoretic approaches without binary search [16, 1]. However, the method using linear programming which can handle at most several thousands of registers is not practical. While, graph-theoretic approaches which construct a constraint graph that represents various constraints can handle a circuit of practical size. The constraints are feasible if and only if the constraint graph contains no negative cycle. Let n and m be the number of registers and the number of register pairs with signal path, respectively. The minimum clock-period is obtained by using $O(nm)$ Bellman-Ford shortest path algorithm with binary search [15, 14, 3]. The exact minimum clock-period is obtained in $O(n^2m)$ by using the minimum cycle Z-mean algorithm [16] which is a generalization of the maximum cycle-mean algorithm [6], and obtained in $O(nm + n^2 \log n)$ [1] by using parameterized shortest path algorithm [19].

In this paper, first, we propose a more efficient Bellman-Ford based algorithm that decides whether the graph contains a negative cycle or not. The original Bellman-Ford shortest algorithm is very time-consuming if the graph contains a negative cycle since it requires n repetitions of update procedure. Therefore, negative cycle detection methods are required to handle a graph that corresponds to a larger circuit [2]. In our proposed algorithm, the simple negative cycle detection method is adopted. The overhead when there is no negative cycle in a graph is negligible. The time complexity of our proposed algorithm is $O(km + k^2n)$ where k is the maximum number of distinct edges in a shortest trail in the graph which can be regarded as a small constant in most cases. By using this decision algorithm, the minimum clock-period with certain precision is obtained by binary search.

The second algorithm is a minimum cost scheduling algorithm. In general, there are a lot of clock-schedules that satisfy the constraints. Therefore, clock-scheduling can be used to improve the circuit reliability [11, 17, 16, 1, 7, 10] and other objectives such as area, wire length and power. In order to use the design margin for other objectives, we assume that the target clock-timing is given for each register. We define the sum of difference between the obtained clock-timing in the clock-schedule and the target clock-timing for every register as the cost of clock-scheduling as in [18]. The algorithm in [18] computes repeatedly a maximum matching of a bipartite graph and obtains a minimum cost clock-schedule efficiently. However, in [18], how to handle the maximum and minimum clock-timing constraints is not discussed. We extend their algorithm to handle these timing constraints. Although the total time complexity is $O(n^2m)$, the efficiency of the algorithm is shown empirically.

Our clock-scheduling engine computes the minimum feasible clock-period and obtains a minimum cost clock-schedule by using proposed algorithms. Then, our engine defines the range of clock-timing of each register so that every constraint is satisfied whenever the clock-timing is realized within each defined range. The engine also provides useful information to the circuit synthesis tools. In experiments to circuits with up to about ten thousand registers, we show that the minimum feasible clock-period and a minimum cost schedule are obtained within a few minutes by our engine.

2 Preliminaries

We consider a circuit with a single clock consisting of registers and combinatorial circuits between them. The *clock-timing* $s(v)$ of register v is the difference in clock arrival time between v and an arbitrary chosen (perhaps hypothetical) reference register. The set of clock-timings is called a *clock-schedule*.

We assume the framework that a circuit works correctly if the following two types of constraints are satisfied for each register pair with signal propagation [4]:

Hold Const. : $s(v) - s(u) \leq d_{\min}(u,v)$

Setup Const. : $s(u) - s(v) \leq T - d_{\max}(u,v)$

where T is the clock-period and $d_{\max}(u,v)$ ($d_{\min}(u,v)$) is intuitively the maximum (minimum) propagation delay from register u to register v along a combinatorial circuit. More precisely, $d_{\max}(u,v)$ is defined as $d'_{\max}(u,v) + \text{setup}(v) + \text{margin}_s(v)$ where $d'_{\max}(u,v)$ (≥ 0), $\text{setup}(v)$ (≥ 0), and $\text{margin}_s(v)$ (≥ 0) are the maximum propagation delay from u to v, the setup-time of register v, and the predefined timing margin of v, respectively. Similarly, $d_{\min}(u,v)$ is defined as $d'_{\min}(u,v) - \text{hold}(v) - \text{margin}_h(v)$ where $d'_{\min}(u,v)$ (≥ 0), $\text{hold}(v)$ (≥ 0), and $\text{margin}_h(v)$ (≥ 0) are the minimum propagation delay from u to v, the hold-time of register v, and the predefined timing margin. The timing margin is used to secure the feasibility in realizing clock-timing and to secure the reliability of the circuit. Note that $d_{\min}(u,v)$ could be negative if the minimum propagation delay $d'(u,v)$ is small.

Multi-clock-cycle signal paths are often used in a conventional zero clock-skew design in order to avoid the performance limitation caused by the maximum signal delay between registers. The constraints for a register pair with signal propagation defined as above are extended so that they correspond to general situations [12]:

Hold Const. : $s(v) - s(u) \leq -\alpha_{u,v}T + d_{\min}(u,v)$

Setup Const. : $s(u) - s(v) \leq \beta_{u,v}T - d_{\max}(u,v)$

where $\alpha_{u,v}$ and $\beta_{u,v}$ are given constants for each pair ($0 \leq \alpha_{u,v} < \beta_{u,v}$). Note that, for a pair with single-clock-cycle signal path, $\alpha_{u,v}$ and $\beta_{u,v}$ are given as 0 and 1, respectively. This formulation enables us to handle multi-clock-cycle signal paths, mixture of positive-edge and negative-edge registers, latch based circuitry, and multi-clocks that have different periods, etc. Note that the feasible clock-period has no upper bound if $\alpha_{u,v}$ is 0 for every pair. That is, if T is feasible then for any T' ($\geq T$) is feasible. However, the feasible clock-period has an upper bound if $\alpha_{u,v}$ is not 0 for some pairs.

In designing a block, a subcircuit, the input and output timings of I/O pins of the block are fixed or constrained by the other parts. An arbitrary large difference of clock-timings in clock-schedule is impractical. Therefore, we are often requested to handle the upper bound $u(v)$ and the lower bound $l(v)$ of clock-timing of register v. In such cases, we prepare the virtual register whose clock-timing is fixed to 0 as used in [3, 1]. The constraints on the upper and lower bounds of clock-timing are described in terms of the virtual signal path from/to the virtual register.

These constraints are represented by the *constraint graph*. The constraint graph $G(V,E)$ is defined as follows: a vertex $v \in V$ corresponds to a register, a directed edge $(u,v) \in E$ corresponds to either type of constraints; edge (u,v) corresponding to the hold (setup) constraint is called hold-edge (setup-edge), and the weight $w(u,v)$ of (u,v) is $-\alpha_{u,v}T + d_{\min}(u,v)$ ($\beta_{u,v}T - d_{\max}(v,u)$). Let E_{hold} and E_{setup} be the set of hold-edges and the set of setup-edges, respectively. The constraint graph G when the clock-period is t is denoted by G_t. Similarly, weight $w(u,v)$ when the clock-period is t is denoted by $w_t(u,v)$.

For clock-schedule s in clock-period t, edge (u,v) in G_t is said to be *legal* if $s(v) - s(u) \leq w_t(u,v)$, *illegal* otherwise. A clock-schedule is called *feasible* in clock-period t if there is no illegal edge in G_t. The slack of edge (u,v) for clock-schedule s in clock-period t is defined as

$$\Delta_{s,t}(u,v) = s(u) + w_t(u,v) - s(v).$$

If the slack of an edge is 0, the edge is said to be *critical*. A cycle (path) consisting of critical edges is called a critical cycle (path). Note that the constraints can be satisfied if and only if G contains no negative cycle [8, 3, 16]. The smallest clock-period t such that G contains no negative cycle is denoted by $T(G)$. Note that there exists a critical cycle in G_t for a feasible clock-schedule if and only if the clock-period t is $T(G)$.

3 Minimum Feasible Clock-Period

By using constraint graph, we can decide whether the given clock-period is feasible or not. The clock-period t is feasible if and only if the constraint graph G_t contains no negative cycle. The detection of negative cycle in a directed graph is possible by using Bellman-Ford shortest path algorithm. The *distance* of a vertex from an arbitrary chosen vertex r, called *root*, is defined as the weight of a shortest trail whose weight is minimum among all trails that connect between root r and the vertex. The weight of a trail is the sum of edge weights in the trail in which the multiple occurrences of edges are allowed. If the distance of every vertex is finite, then there is no negative cycle in the graph and Bellman-Ford shortest path algorithm converges. The famous Dijkstra shortest path algorithm is not applicable since the graph contains edges with negative weight.

If there is no negative cycle in the constraint graph, Bellman-Ford algorithm converges in early stages since the number of distinct edges in a shortest trail is much smaller than the number of vertices in the constraint graph extracted from a circuit. However, if there are negative cycles, the update procedure in Bellman-Ford algorithm is repeated as many times as the number of vertices since Bellman-Ford algorithm does not converge. Thus, a negative cycle detection strategy is required to detect a negative cycle in early stages. There are several negative cycle detection strategies [2]. Among them, we adopt the simplest one called "walk to the root", though it was concluded as a slow one compared to other strategies in [2] because its time complexity is $O(n)$.

In the strategy "walk to the root", a negative cycle is detected if root r is not reachable from a vertex by following parent pointers. In [2], the checking from vertex u that follows parent pointers is executed before labeling operation with respect to edge (u,v). If vertex v is contained in the route from u to r, then labeling operation to v from u creates a cycle and a negative cycle

```
Procedure IsNoNegCycle( $G(V,E)$, $T$ )
Input: constraint graph $G$, target clock-period $T$.
Output: Yes or No.
 1. Construct $G_T$, choose $r \in V$, $Q_1 \leftarrow \emptyset$, $Q_2 \leftarrow \emptyset$. /* init */
 2. for all $v \in V$ let $d(v) := \infty$. /* distance from $r$ */
 3. $d(r) := 0$, push $r$ to $Q_1$.
    /* update procedure (steps 4–15) */
 4. while $Q_1$ is not empty do
 5.   $u \leftarrow pop(Q_1)$.
 6.   for all $v$ adjacent to $u$ in $G$ do
 7.     if $d(v) > d(u) + w_T(u,v)$ then
          /* labeling operation (steps 8–10) */
 8.       $d(v) := d(u) + w_T(u,v)$.
 9.       set parent pointer from $v$ to $u$.
10.       push $v$ to $Q_2$ if $v$ is not in $Q_2$.
11.     endif
12.   endfor
13.   if $r$ is in $Q_2$ return "No". (negative cycle exists)
14.   if reach $u$ by following parent pointer from $u$
                    return "No". (negative cycle exists)
15. endwhile
16. if $Q_2$ is empty return "Yes". (no negative cycle exists)
17. $Q_1 \leftarrow Q_2$, $Q_2 \leftarrow \emptyset$, goto step 4.
```

Figure 1: Negative Cycle Detection by Bellman-Ford with Walk to the Root (Decision Algorithm).

is detected. Otherwise the labeling operation does not create a cycle. In our proposed algorithm, the checking from u is executed after all labeling operations to the neighbors of u are done. If a cycle is created by these operations, the cycle contains u, that is, u is contained in the route from u. Otherwise, r is reachable from u. Moreover, the our algorithm detects a negative cycle if the labeling operation to root r is done. Although it is possible to execute the checking after each update procedure is finished, the algorithm executes the checking after labeling operations to the neighbors are finished since it is empirically fast.

Our proposed negative cycle detection algorithm is shown in Fig. 1. Let n, m, and k be the number of vertices, the number of edges, and the number of distinct edges in a shortest trail, respectively. The time complexity of initialization (steps 1, 2, and 3), the labeling operation and "walk to the root" (steps 13 and 14), are $O(n)$, $O(1)$, and $O(k)$, respectively. In each update procedure, the labeling operation and step 14 are executed at most m times and at most n times, respectively. The update procedure is repeated k times. Therefore, the time complexity of the algorithm is $O(km + k^2 n)$.

By using proposed negative cycle detection algorithm, we can check whether the given clock-period is feasible or not. In order to use this algorithm in binary search, we need to find lower and upper bounds of the minimum feasible clock-period. The maximum signal propagation delay from a register to the same register gives a lower bound of feasible clock-period. The difference of the maximum and minimum signal propagation delays from a register to another register gives also a lower bound of clock-period. We adopt the larger of these two

```
Procedure MinClock( $G(V,E)$ )
Input: constraint graph $G(V, E_{\text{hold}} \cup E_{\text{setup}})$.
Output: minimum feasible clock-period $T$.
 1. $L_{\text{self}} := \max_{(u,u)\in E_{\text{hold}}} d_{\max}(u,u)$.
 2. $L_{\text{diff}} := \max_{(u,v)\in E_{\text{hold}}} (d_{\max}(u,v) - d_{\min}(u,v))$.
 3. $L := \max\{L_{\text{self}}, L_{\text{diff}}\}$.
 4. $T := \max_{(u,v)\in E_{\text{hold}}} d_{\max}(u,v)$.
 5. if IsNoNegCycle( $G$, $L$ ) = "Yes" return $L$.
 6. if IsNoNegCycle( $G$, $T$ ) = "No" return $\infty$.
 7. while $T - L > \varepsilon$ do
 8.   $M := (T + L)/2$.
 9.   if IsNoNegCycle( $G$, $M$ ) = "Yes" then $T := M$,
10.   else $L := M$ endif
11. endwhile
12. return $T$.
```

Figure 2: Minimum Feasible Clock-Period Algorithm.

```
Procedure MinCost( $G_T(V,E_T)$, $l$, $u$, $o$ )
Input: constraint graph $G_T$, lower bound $l(v)$, upper bound
       $u(v)$, and target timing $o(v)$ for $v \in V$.
Output: clock-timing $s(v)$ for $v \in V$.
 1. Get an initial clock-schedule $s$.
 2. do
 3.   do $s :=$ TryDec( $G_T$, $l$, $o$, $s$ ) while $s$ is modified.
 4.   do $s :=$ TryInc( $G_T$, $u$, $o$, $s$ ) while $s$ is modified.
 5. while $s$ is modified.
 6. return $s$.
```

Figure 3: Minimum Cost Scheduling Algorithm.

lower bounds as the lower bound of the binary search. In case $\alpha_{u,v}$ is 0 for each pair, there is no upper bound in feasible clock-period. We adopt the maximum signal propagation delay between registers as an upper bound since it gives a feasible clock-period even in zero clock-skew framework if there is no negative cycle consisting of hold-edges. In case $\alpha_{u,v}$ is not 0 for some pairs, the above approach might miss the feasible clock-period range. If the maximum feasible clock-period is less than the maximum signal propagation delay, then we fail to find the feasible clock-period range. However, we do not take these cases into consideration at this point since they seldom happen.

The algorithm to determine the minimum feasible clock-period is shown in Fig. 2. The above mentioned lower and upper bounds of the minimum feasible clock-period are determined in $O(m)$. The time complexity of the algorithm is $O(\gamma(km + k^2 n))$ where γ is the number of repetitions of the decision algorithm. In experiments, k and γ are less than 100 and 20, respectively.

4 Minimum Cost Scheduling

The minimum cost scheduling algorithm is shown in Figs. 3, 4, and 5. The cost of a clock-schedule s is defined as the sum of differences between the clock-timing $s(v)$ and the target clock-timing $o(v)$ of register v. The algorithm finds a clock-schedule with the minimum cost efficiently.

The computation time depends on an initial clock-schedule, though the final cost is same. We found a

```
Procedure TryDec( $G_T(V, E_T)$, $l$, $o$, $s$ )
Input: constraint graph $G_T$, lower bound $l(v)$, target timing
       $o(v)$, and current clock-timing $s(v)$ for $v \in V$.
Output: modified clock-timing $s(v)$ for $v \in V$.
 1. $V_1 := \{v \in V \mid s(v) > o(v)\}$.
 2. $V_2 := \{v \in V \mid s(v) \leq o(v)\}$.
 3. $F_2 := \{v \in V \mid s(v) = l(v)\}$.
 4. $E^c := \{$ critical edges in $G_T$ for $s\}$.
 5. if $V_1 \neq \emptyset$ then
 6.   $X :=$ FindCRD($G(V_1, V_2, E^c), F_2$).
 7.   For each $X_i \in X$
 8.     $\delta_1 :=$ median$_{v \in X_i}(s(v) - o(v))$.
 9.     $\delta_2 := \min_{(u,v) \in E_T, u \in X_i, v \notin X_i} \Delta_{s,T}(u, v)$.
10.     $\delta := \min(\delta_1, \delta_2)$.
11.     for all $v \in X_i$ let $s(v) := s(v) - \delta$.
12.   endfor
13. endif
14. return $s$.
```

Figure 4: Reduce Cost by Cost Reducible Vertex Sets.

```
Procedure FindCRD( $G(V_1, V_2, E)$, $F_2$ )
Input: critical graph $G$, vertex set $F_2 \subseteq V_2$.
Output: set $X$ of CRDs $X_i \subseteq V_1 \cup V_2$ such that $|X_i \cap V_1| >$
        $|X_i \cap V_2|$, $X_i \cap F_2 = \emptyset$, and there is no outgoing edge
        from $X_i$ to $(V_1 \cup V_2) \setminus X_i$ in $G$.
1. $F := \{u \in V_1 \cup V_2 \mid$ directed path from $u$ to $v \in F_2$ in $G\}$.
2. $E^t := \{(u, v) \mid$ directed path from $u \in V_1$ to $v \in V_2$ in $G\}$.
3. $V_1^b := V_1 \setminus F$, $V_2^b := V_2 \setminus F$.
4. $E^b := \{(u, v) \in E^t \mid u \in V_1^b, v \in V_2^b\}$.
5. Find maximum matching in bipartite graph $G^b(V_1^b, V_2^b, E^b)$.
6. Let $V^c$ be the set of vertices unreachable from any un-
   matched vertex in $V_1^b$ in $G^b$ using unmatched edges from
   a vertex in $V_1^b$ and matched edges from a vertex in $V_2^b$.
7. Let $X$ be the set of vertex sets of connected components of
   the graph obtained from $G$ by deleting vertices in $V^c \cup F$.
8. return $X$.
```

Figure 5: Find Cost Reducible Vertex Sets by Decrease.

better initial solution in step 1 of Procedure **MinCost** in Fig. 3 by using a variance of Bellman-Ford shortest path algorithm, but a detailed discussion is omitted here. The initial solution is modified repeatedly by procedures **TryDec** and **TryInc** until an optimum solution is obtained. In Procedure **TryDec**, the clock-timing of each register in the vertex set X_i obtained by Procedure **FindCRD** is decreased by δ. Procedure **TryDec** and Procedure **FindCRD** are shown in Figs. 4 and 5, respectively. Procedure **TryInc** is defined similarly but omitted. Although the strategy of this algorithm is same as that in [18], the constraints on upper bound $u(v)$ and lower bound $l(v)$ of clock-timing of register v are taken into account.

For a given clock-schedule, a vertex set X_i is said to be *cost reducible by decrease* (CRD) if the cost of the clock-schedule is reduced without violating constraints by decreasing the clock-timing of each register in X_i by the same amount. That is, $|\{v \in X_i \mid s(v) > o(v)\}| > |\{v \in X_i \mid s(v) \leq o(v)\}|$ X_i contains no vertex v such that $s(v) = l(v)$, and X_i has no outgoing critical edge to $V \setminus X_i$. For each CRPD, the amount of change is determined in order to reduce the cost maximally without violating the constraints.

```
Procedure Clock-Schedule( $C$, $T$, $l$, $u$, $r$, $o$ )
Input: circuit graph $C(V_c, E_c)$, target clock-period $T$, and
       lower bound $l(v)$, upper bound $u(v)$, minimum range
       $r(v)$ and target timing $o(v)$ for $v \in V$.
Output: clock-period $T$, clock-timing $(s_{\min}(v), s_{\max}(v))$ for
        $v \in V$.
1. Construct constraint graph $G(V, E)$ from $C$, $l$, $u$, $r$.
2. if constraints are infeasible
       Output Statistics and return "Infeasible".
3. if target clock-period $T$ is not given
       let $T :=$ MinClock( $G$ ).
4. $s :=$ MinCost( $G_T$, $o$ ).
5. $s :=$ MaxRange( $G_T$, $s$ ).
6. Output Schedule and Statistics.
```

Figure 6: Clock-Scheduling Engine.

Lemma 1 *Procedure* **FindCRD** *finds cost reducible vertex sets by decrease, if exist.*

The proof is omitted here for space. Notice that cost reducible vertex sets by increase can be found by similar procedure.

Theorem 1 *Procedure* **MinCost** *obtains a minimum cost schedule s, i.e.* $\sum_{v \in V} |s(v) - o(v)|$ *is minimum.*

The proof is omitted here for space. Although the time complexity of Procedure **MinCost** is $O(n^2 m)$ where n and m are the number of registers and the number of register pairs with signal path, respectively, the experiments show that it is practically fast.

5 Clock-Scheduling System

The procedures described in previous sections and others are combined into one system as shown in Fig. 6. Here, other procedures in the system are briefly described.

By defining margin$_s(v)$ and margin$_h(v)$, the minimum range of clock-timing for each register is secured. However, since the effect of clock-scheduling is not considered in the current circuit synthesis, the clock-timing of most registers in the circuit can have very large range without violating the circuit functionality. The larger the range of clock-timing is, the easier the realization is. This means that the possibility of reduction of other metrics such as clock wire length, power consumption in clock-tree increase. Therefore, we secure the range of clock-timing as large as possible by a greedy heuristic **MaxRange** (details are omitted).

This system is used as a subroutine of synthesis tools. Therefore, it provides information of a circuit to improve the circuit performance. A critical part of the circuit in term of the minimum clock-period obtained by clock-scheduling form a cycle in the constraint graph. More precisely, the union of these cycles are the critical part of the circuit. The strongly connected component of the graph obtained from the constraint graph by deleting non-critical edges in the clock-schedule that achieves the

Table 1: Statistics of Circuits (0.25 & 0.18 [μm]).

circuit	#reg	#path	max-d [ps]	min-cp [ps] (%)
block1	1654	11697	11569	8323 (71.9)
block2	6439	113101	11911	11654 (97.8)
block3	7973	104136	11808	9553 (80.9)
block4	7052	126559	8354	6256 (74.9)
block5	12460	947082	12621	9665 (76.6)

#reg : number of registers (including I/O pins).
#path : number of register pairs with signal paths.
max-d : maximum of maximum delay over clock-cycle.
min-cp : minimum feasible clock-period.

minimum clock-period is obtained. Among the strongly connected components obtained, the components consist of more than one vertex, and the components consist of one vertex with critical self-loop are detected as the critical parts of the circuit.

6 Experiments

The algorithms proposed are written in C++ and implemented on Sun Ultra 10 (384M memory, 778M swap). For comparisons, LP solver lp_solve_3.2 [9] and CPLEX [5] are executed on the same machine.

The data used in experiments shown in Table 1 are extracted from circuits designed in an industry in 0.25[μm] (block1, 2, and 3) and in 0.18[μm] (block4 and 5) technologies. Among circuits, block1, 2, and 3 contain multi-clock-cycle signal paths. Feasible clock-period of block2 and 3 have upper bounds, since they contain a register pair with non-zero $\alpha_{u,v}$. In experiments, unless otherwise specified, the timing of each I/O pin is scheduled as well as registers, the target clock-timing and its minimum range are both set to 0, for simplicity. Of course, the resultant minimum feasible clock-period and clock-schedule may differ if the constraints differ, which will be demonstrated in Table 6.

In Table 2, the computation times to decide whether the target clock-period is feasible or not by using Bellman-Ford shortest path algorithm with and without negative cycle detection are shown. The computation times of two algorithms are almost same if there is no negative cycle. This shows that the overhead of negative cycle detection is small enough. When the circuit size is large and the negative cycle exists, the expansion of the computation time of algorithm without negative cycle detection is recognized. This reflects the fact that the time complexity without negative cycle detection is $O(nm)$. While, the computation time of the algorithm with negative cycle detection remains small when the circuit size is large. It is confirmed that the number of repetitions of the update procedure, k, is almost independent of the circuit size.

In Table 3, the comparisons on computation times between our proposed minimum clock-period computation algorithm and LPs are shown. LP1 is lp_solve_3.2 and LP2 is an industrial tool CPLEX. In our proposed algorithm, the BF-ncd is executed one time except for block5 since the minimum clock-period of each circuit is equal to its lower bound explained. The computation time of our proposed algorithm is small even if LP can not handled the system of inequalities.

Table 2: Negative Cycle Detection Algorithms.

circuit	t-cp	Ans.	BF	BF-ncd (Proposed.)		
	[ps]		time [s]	time [s]	(%)	k
block1	4000	No	26.66	0.0023	(0.000)	3
	8322	No	0.8762	0.0148	(0.017)	6
	8323	Yes	0.1255	0.1259	(1.003)	6
	11569	Yes	0.1248	0.1250	(1.002)	1
block5	8763	No	11669.6	0.253	(0.000)	3
	9664	No	1096.11	0.246	(0.000)	4
	9665	Yes	7.256	7.313	(1.007)	9
	10692	Yes	7.130	7.151	(1.002)	5

BF : Bellman-Ford shortest path algorithm.
BF-ncd : Bellman-Ford with negative cycle detection.
t-cp : target clock-period.
time : computation time.
k : number of repetitions of update procedure.

Table 5: Computation Time of Algorithms.

name	read[s]	cp[s]	cost[s]	range[s]	write[s]	total[s]
block1	0.89	0.14	0.70	0.10	0.19	2.02
block2	9.59	1.80	0.72	0.84	1.55	14.50
block3	10.00	3.13	15.72	1.14	1.88	31.87
block4	10.84	2.17	5.89	1.41	1.82	22.13
block5	71.30	38.73	61.42	4.79	12.52	198.76

read : time for reading data.
cp : time for minimum feasible clock-period.
cost : time for minimum cost schedule.
range : time for setting range of clock-timing.
write : time for outputting result.
total : time for total computation.

In Table 4, the comparisons on computation time between our proposed minimum cost schedule algorithm and LPs are shown. Our proposed algorithm obtains optimal solutions constantly faster than others. Also, the efficiency in terms of space complexity against LPs is proved empirically.

In Table 5, computation times of overall system are shown. It is confirmed that the computation time is almost linear to the circuit size. In Table 6, clock-schedule results under various constrains are shown. In this experiment, the following parameters are given to the system as constraints or used to control the system: the target clock period, the minimum range of clock-timing, the clock-timing of I/O is fixed or not, and the sum of ranges of clock-timings is maximized or not.

Table 6: Schedule Results (by proposed algorithm).

	Constraints				Result			
name	t-cp	min-r	pin	range	min	max	cost	time
	[ps]	[ps]			[ps]	[ps]	[ns]	[s]
block5	opt	0	No	No	9665	∞	175.7	193.97
	opt	0	Yes	Yes	9665	∞	177.2	293.79
	opt	100	No	Yes	9815	∞	188.0	221.59
	opt	100	Yes	No	9815	∞	189.4	311.85

t-cp : target clock-period. (opt : optimum is set)
min-r : minimum range of clock-timing.
pin : Is clock-timing of I/O fixed to 0?
range : Is range of clock-timing maximized?
min (max) : minimum (maximum) feasible clock-period.
cost : sum of differences of obtained and target clock-timings.
time : total computation time.

Table 3: Minimum Clock-Period Algorithms.

circuit	min-cp [ps]	L_{self} [ps]	L_{diff} [ps]	Proposed Algorithm time[s]	(read[s])	γ	time/γ [s]	LP1(lp_solve_3.2) time[s]	(read[s])	LP2(CPLEX) time[s]	(read[s])
block1	8323	6036	8323	0.14	(0.89)	1	0.14	60.22	(4.38)	6.23	(0.38)
block2	11654	11654	10647	1.80	(9.59)	1	1.80	18.11	(1609.90)	27.90	(4.31)
block3	9553	–	9553	3.13	(10.00)	1	3.13	OT	(1324.58)	1651.90	(3.95)
block4	6256	4358	6256	2.17	(10.84)	1	2.17	830.36	(1888.90)	52.93	(4.72)
block5	9665	8763	8758	38.73	(70.35)	14	2.77	–	(OT)	OM	(43.05)

LP1 : lp_solve_3.2.
LP2 : CPLEX 7.0.0.
min-cp : minimum feasible clock-period.
γ : number of repetitions of BF-ncd.
L_{self} : maximum delay from a register to the same register.
L_{diff} : maximum difference between max and min delays in signal paths.
time : time for computation.
read : time for reading data.
OT : time > 5000[s].
OM : out of memory.

Table 4: Minimum Cost Schedule Algorithms.

circuit	t-cp[ps]	cost[ns]	Proposed Algorithm time[s]	#rep	time/#rep [s]	LP1(lp_solve_3.2) time[s]	(read[s])	LP2(CPLEX) time[s]	(read[s])
block1	8323	111.1	0.70	33	0.02	11.87	(15.83)	3.15	(0.44)
block2	11654	6.9	0.72	6	0.12	34.63	(998.80)	32.68	(4.32)
block3	9553	656.1	15.72	77	0.20	1166.90	(1169.71)	59.19	(4.07)
block4	6256	82.0	5.89	35	0.17	327.13	(1198.57)	50.18	(4.86)
block5	9665	175.7	61.42	75	0.82	–	(OT)	OM	(39.17)

LP1 : lp_solve_3.2.
LP2 : CPLEX7.0.0.
t-cp : target clock-period.
cost : sum of differences of obtained and target clock-timings.
#rep : the number of repetitions of FindCRDs.
time : time for computation.
read : time for reading data.
OT : time > 5000[s].
OM : out of memory.

7 Conclusions

This paper introduces a practical fast clock-scheduling engine. The results to the circuit with about 10000 registers and 100000 data paths are obtained within a few minutes. It handles various design constraints such as clock-timing of I/O pin and multi-clock-cycle signal path.

For the future work, the tolerance for the clock-jitter under the existence of multi-clock-cycle signal path is urgent. The range of feasible clock-period should be taken into account in clock-schedule design. The improvement in budgeting to the range of clock-timing is also important which will improve the various circuit metric such as area, power, and clock-circuitry.

References

[1] C. Albrecht, B. Korte, J. Schietke, and J. Vygen. Cycle time and slack optimization for VLS-chips. In *Proc. International Conference on Computer-Aided-Design (ICCAD)*, pages 233–238, 1999.

[2] B. Cherkassky and A. Goldberg. Negative-cycle detection algorithms. tr 96-029, NEC Research Institute, Inc., March 1996.

[3] R. B. Deokar and S. S. Sapatnekar. A graph-theoretic approach to clock skew optimization. In *Proc. International Symposium on Circuits and Systems (ISCAS)*, volume 1, pages 407–410, 1994.

[4] J. P. Fishburn. Clock skew optimization. *IEEE Trans. on Computers*, 39(7):945–951, 1990.

[5] ILOG. CPLEX7.0.0 User's Manual.

[6] R. M. Karp. A characterization of the minimum cycle mean in a digraph. *Discrete Mathematics*, 23:309–311, 1978.

[7] I. Kourtev and E. Friedman. Clock skew scheduling for improved reliability via quadratic programming. In *Proc. International Conference on Computer-Aided-Design (ICCAD)*, pages 239–243, 1999.

[8] E. L. Lawler. *Combinatorial Optimization, Networks and Matroids.* Holt, Rinehart and Winston, New York, 1976.

[9] lp_solve_3.2. ftp://ftp.es.ele.tue.nl/pub/lp_solve.

[10] R. Mader, E. Friedman, A. Litman, and I. Kourtev. Large scale clock skew scheduling techniques for improved reliability of digital synchronous VLSI circuits. In *Proc. International Symposium on Circuits and Systems (ISCAS)*, volume 1, pages 357–360, 2002.

[11] J. L. Neves and E. G. Friedman. Optimal clock skew scheduling tolerant to process variations. In *Proc. 33rd Design Automation Conference (DAC)*, pages 623–628, 1996.

[12] B. A. Rosdi and A. Takahashi. Reduction on the usage of intermediate registers for pipelined circuits. In *Proc. the Workshop on Synthesis and System Integration of Mixed Technologies (SASIMI 2004)*, pages 333–338, 2004.

[13] K. A. Sakallah, T. N. Mudge, and O. A. Olukotun. checkTc and minTc: Timing verification and optimal clocking of synchronous digital circuits. In *Proc. International Conference on Computer-Aided-Design (ICCAD)*, pages 552–555, 1990.

[14] N. Shenoy, R. Brayton, and A. Sangiovanni-Vincentelli. Graph algorithms for clock schedule optimization. In *Proc. International Conference on Computer-Aided-Design (ICCAD)*, pages 132–136, 1992.

[15] T. G. Szymanski. Computing optimal clock schedules. In *Proc. 29th Design Automation Conference (DAC)*, pages 399–404, 1992.

[16] A. Takahashi and Y. Kajitani. Performance and reliability driven clock scheduling of sequential logic circuits. In *Proc. Asia and South Pacific Design Automation Conference (ASP-DAC)*, pages 37–42, 1997.

[17] J. G. Xi and W. W. M. Dai. Jitter-tolerant clock routing in two-phase synchronous systems. In *Proc. International Conference on Computer-Aided-Design (ICCAD)*, pages 316–320, 1996.

[18] T. Yoda and A. Takahashi. Clock schedule design for minimum realization cost. *IEICE Transactions on Fundamentals*, E83-A(12):2552–2557, 2000.

[19] N. Young, R. Tarjan, and J. Orlin. Farster parametric shortest path and minimum balance algorithms. *Networks*, 21:205–221, 1991.