論文 / 著書情報
Article / Book Information

Title	Multi-clock Cycle Paths and Clock Scheduling for Reducing the Area of Pipelined Circuits
Authors	Bakhtiar Affendi Rosdi, Atsushi Takahashi
Citation	IEICE Trans. Fundamentals, Vol. E89-A, No. 12, pp. 3435-3442
Pub. date	2006, 12
URL	http://search.ieice.org/
Copyright	(c) 2006 Institute of Electronics, Information and Communication Engineers

PAPER Special Section on VLSI Design and CAD Algorithms

Multi-Clock Cycle Paths and Clock Scheduling for Reducing the Area of Pipelined Circuits

Bakhtiar Affendi ROSDI[†a)] ***and*** **Atsushi TAKAHASHI**[†]**,** ***Members***

SUMMARY A new algorithm is proposed to reduce the number of intermediate registers of a pipelined circuit using a combination of multi-clock cycle paths and clock scheduling. The algorithm analyzes the pipelined circuit and determines the intermediate registers that can be removed. An efficient subsidiary algorithm is presented that computes the minimum feasible clock period of a circuit containing multi-clock cycle paths. Experiments with a pipelined adder and multiplier verify that the proposed algorithm can reduce the number of intermediate registers without degrading performance, even when delay variations exist.

key words: *pipelined circuits, multi-clock cycle paths, clock scheduling*

1. Introduction

The sustained progress of VLSI technology has led to increasing wire delays, shrinking clock period and growing chip size. Circuit pipelining is one technique that has been used in order to shrink the clock period. Pipelining is a method in which a circuit is divided into a small number of stages and intermediate registers are inserted between stages to store the intermediate data. With this method, extra circuit area is required to situate the additional intermediate registers and the size of the clock tree is also increased.

Recently, to overcome this problem, several studies have been carried out on wave pipelining [1], which is a method of speeding up the circuit without the insertion of intermediate registers. However, wave pipelining requires tighter timing constraints. In wave pipelining, there may exist a number of 'waves' of data in a circuit at any given time. Therefore, to avoid data collisions, delay balancing is required, which increases the circuit area.

This paper presents a new algorithm to reduce the number of intermediate registers of a pipelined circuit by using a combination of multi-clock cycle paths and clock scheduling. A multi-clock cycle path is a path from register to register where data transmission takes more than one clock period. Note that in wave pipelining, all paths are multi-clock cycle paths. Introducing a multi-clock cycle path into a pipelined circuit allows some intermediate registers to be removed. However, as mentioned above, certain timing constraints must be satisfied. Therefore, there is a trade-off between area reduction from register removal and area increase from delay balancing. Clock scheduling is a technique in which the clock skew of a register is intentionally introduced to improve circuit performance by relaxing the timing constraints. Using clock scheduling, more intermediate registers can be removed, without the need for delay balancing.

The minimum feasible clock period in terms of clock scheduling is obtained by linear programming [2], by graph-theoretic approaches with binary search [3], [4], or by graph-theoretic approaches without binary search [5]. Graph-theoretic approaches are based on construction of a constraint graph that represents various constraints and which can handle a circuit of practical size. The constraints are feasible if and only if the constraint graph contains no negative cycle. In graph-theoretic approaches with binary search [4], the Bellman-Ford shortest path algorithm is used to decide whether the graph contains a negative cycle and a simple negative cycle detection method is employed to increase speed. The algorithm proposed in [4] is for a circuit that contains single-clock cycle paths only. However, when the algorithm [4] is applied to a circuit containing multi-clock cycle paths, there are some cases for which the minimum feasible clock period cannot be determined. The clock period for such a circuit is bounded above, unlike the situation for a circuit containing only single-clock cycle paths. This range of feasible clock periods has to be taken into account in clock-schedule design.

In this paper, we initially discuss the constraints on a circuit containing multi-clock cycle paths. These constraints take into account the range of feasible clock periods required to make the circuit tolerant of clock jitter and delay variation. Using the constraints, we enhance the algorithm in [4] to find the minimum feasible clock period of a circuit that contains multi-clock cycle paths. A negative cycle exists in the constraint graph whenever the constraints are infeasible and it is found in the algorithm. The enhanced minimum clock-period algorithm uses the information from the found negative cycle to narrow the binary search interval efficiently. Then, we propose an algorithm to reduce the number of intermediate registers of a pipelined circuit by introducing multi-clock cycle paths with clock scheduling. An intermediate register is a register that stores one of the data between the stages of the data flow. Here we consider a pipelined circuit such as adder or multiplier, whose data flow is in one way without any feedback. However, we believe that our algorithm can be enhanced so that it can be used to a pipelined circuit with feedback. In the proposed

Manuscript received March 13, 2006.
Manuscript revised June 13, 2006.
Final manuscript received August 1, 2006.

[†]The authors are with the Department of Communications and Integrated Systems, Tokyo Institute of Technology, Tokyo, 152-8550 Japan.
a) E-mail: fendi@lab.ss.titech.ac.jp

DOI: 10.1093/ietfec/e89–a.12.3435

algorithm, all intermediate registers of the pipelined circuit are initially removed. Then the minimum feasible clock period of the resulting circuit under the condition that the target clock-period range is secured is computed by the proposed minimum clock-period algorithm. If this value is too high, i.e. greater than the target minimum clock period, then intermediate registers are repeatedly inserted into the multi-clock cycle paths until the minimum feasible clock period has been sufficiently reduced.

Experiments with a pipelined adder and multiplier verify that, given a particular target clock-period range, the proposed algorithm can reduce the number of intermediate registers, even when delay variations are present.

2. Preliminaries

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

We make the basic assumption that a circuit works correctly if the following two types of constraint are satisfied for each register pair with signal propagation [2], [6]:

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

where T is the clock period, $d_{\max}(u,v)$ ($d_{\min}(u,v)$) is the maximum (minimum) propagation delay from register u to register v along a combinatorial circuit, and $\beta_{u,v}$ and $\alpha_{u,v}$ are given constants ($\beta_{u,v} > \alpha_{u,v} \geq 0$). Note that for a pair of registers with a single-clock cycle path, $\beta_{u,v}$ and $\alpha_{u,v}$ are given by 1 and 0, respectively. This formulation is sufficiently general to deal with multi-clock cycle paths, a mixture of positive-edge and negative-edge registers, latch based circuitry, and multi-clocks that have different periods.

If $\alpha_{u,v}$ is 0 for every pair, the feasible clock period has no upper bound, i.e. if the clock period T is feasible then any T'(where $T' \geq T$) is feasible. However, the feasible clock period is bounded above if $\alpha_{u,v}$ is not 0 for some pair (u,v).

From the above constraints, when the clock schedule and the signal propagation delay are known, the minimum and maximum feasible clock period, $T_{\min}$ and $T_{\max}$, can be determined from the setup and hold constraints, respectively.

If the clock timing is not fixed, then $T_{\min}$ and $T_{\max}$ depend on each other. $T_{\min}$ has to be minimized under the constraint that the circuit works correctly throughout a certain clock-period range, in order for the circuit to tolerate clock jitter and delay variation. The above constraints become:

(1) Setup Const. :
$s(u) - s(v) \leq \beta_{u,v}T_{\min} - d_{\max}(u,v)$
(2) Hold Const. :
$s(v) - s(u) \leq d_{\min}(u,v) - \alpha_{u,v}\delta - \alpha_{u,v}T_{\min}$

where δ is the clock-period range, i.e. $\delta = T_{\max} - T_{\min}$. Therefore if δ is given, then, by using the above constraints, clock timings can be determined so that the circuit works correctly for a clock period between $T_{\min}$ and $T_{\min}+\delta$. In the following, our target is to minimize $T_{\min}$ under the constraint that the circuit is feasible throughout the given clock-period range, i.e. that constraints (1) and (2) hold.

These constraints are represented by the constraint graph $G(V,E)$ of the circuit, which 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 constraint; an edge (u,v) corresponding to the setup (hold) constraint is called a Z-edge (D-edge), and the weight $w(u,v)$ of (u,v) is $\beta_{v,u}T - d_{\max}(v,u)(d_{\min}(u,v) - \alpha_{u,v}\delta - \alpha_{u,v}T)$. The constraint graph G corresponding to clock period t and clock-period range δ is denoted by $G_\delta(t)$. We may denote $G_\delta(t)$ as $G(t)$ if no confusions occur.

In the following we assume that δ is given as constant. In a constraint graph G, for any cycle C, the cycle weight $w(C)$ is the sum of edge weights over the cycle. The cycle weight can be expressed as $kT - w$, where T is the clock period and k and w are constants.

Definition 1: In the constraint graph G, a cycle C for which $w(C) = kT - w$ is said to be of **type 0**, **type P**, or **type M**, as $k = 0$, $k > 0$, or $k < 0$, respectively.

Theorem 1: Let C be a negative cycle in the constraint graph $G(t)$. If C is of **type 0**, then for any t', there exists a negative cycle in the constraint graph $G(t')$.

Theorem 2: Let C be a cycle in the constraint graph G such that $w(C) = kT - w$. If C is of **type P**, then for any $t < w/k$, there exists a negative cycle in the constraint graph $G(t)$, whereas, if C is of **type M**, then for any $t > w/k$, there exists a negative cycle in the constraint graph $G(t)$.

Definition 2: Let C be a cycle in the constraint graph G such that $w(C) = kT - w$. Then $Bound(C) = w/k, w/k$, or ∞, according to whether C is of **type P**, **type M**, or **type 0**, respectively.

Example: The constraint graph G_0^1 of the circuit shown in Fig. 1(a) where clock-period range $\delta = 0$ is shown in Fig. 1(b). In G_0^1, the cycle $C^1 = (u, w2, v2, u)$ with $w(C^1) = 4T - 20$ is of **type P** and $Bound(C^1) = 5$. The cycle

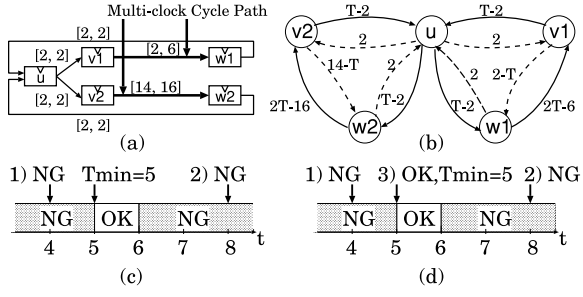

Fig. 1 (a) Circuit containing multi-clock cycle paths. (b) Constraint graph G^1. (c) Min. clock-period computation by the algorithm shown in [4]. (d) Min. clock-period computation by the proposed algorithm.

$C^2 = (u, v1, w1, u)$ with $w(C^2) = -T + 6$ is of **type M** and $Bound(C^2) = 6$.

Note that, in a constraint graph of a circuit that contains just single-clock cycle paths, only **type P** and **type 0** cycles can exist, whereas in a constraint graph of a circuit that contains multi-clock cycle paths, all three cycle types can be present.

3. Minimum Feasible Clock Period

3.1 A Circuit That Contains Just Single-Clock Cycle Paths

The minimum feasible clock period of a circuit that contains just single-clock cycle paths can be determined by graph-theoretic approach with binary search [4]. 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 delay from a register to another register gives also a lower bound of feasible clock period. They adopt the larger of these two lower bounds as an initial lower bound L of the binary search. They adopt the maximum signal propagation delay between registers as an initial upper bound U since it gives a feasible clock period even in zero clock-skew framework.

In the algorithm proposed in [4], the initial lower bound L and upper bound U are initially checked. If L is feasible the algorithm is stopped and output L as the minimum feasible clock period. While, if L is infeasible, U is checked to confirm there is no negative cycle of **type 0**. If a negative cycle of **type 0** is found, the circuit is infeasible and the algorithm is stopped. If U is feasible, the algorithm does binary search by adjusting the lower and upper bounds to determine the minimum feasible clock period. Throughout this paper, the precision ε used in the binary search is 1.

Using the algorithm, let us determine the minimum feasible clock period of the circuit shown in Fig. 2(a). The constraint graph is independent of the clock-period range δ since $\alpha_{u,v} = 0$ for every pair, and there is no upper bound of the clock period. In this example, initial lower bound $L = 2$ and initial upper bound $U = 6$. So, the algorithm does binary search between 2 and 6 as follows:

1) Check 2(L)**:** Since negative cycle $C^1 = (u, w, v, u)$ with $w(C^1) = 3T - 10$ is found, $G^2(2)$ is infeasible. So the next step is check U.

2) Check 6(U)**:** $G^2(6)$ is feasible, so the next step is check $H = (U + L)/2 = 4$.

3) Check 4(H)**:** $G^2(4)$ is feasible, so 4 becomes new upper bound U and $H = 3$.

4) Check 3(H)**:** Since negative cycle $C^1 = (u, w, v, u)$ with $w(C^1) = 3T - 10$ is found, $G^2(3)$ is infeasible. Since $U - L = 1$, output 4 as the minimum feasible clock period T.

The check sequence of the algorithm is shown in Fig. 2(c).

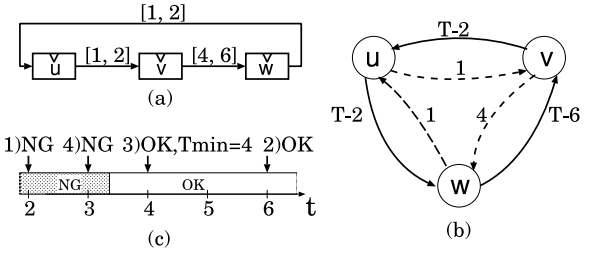

Fig. 2 (a) Circuit containing just single-clock cycle paths. (b) Constraint graph G^2. (c) Min. clock-period computation by the algorithm shown in [4].

3.2 A Circuit That Contains Multi-Clock Cycle Paths

For a circuit that contains just single-clock cycle paths, if the circuit is infeasible at the initial upper bound U, then the circuit is infeasible at any clock period. However, for a circuit that contains multi-clock cycle paths, even if the circuit is infeasible at initial upper bound U, there are some possibilities that the circuit is feasible at clock period t (where $t < U$ or $t > U$).

When the initial upper bound U is infeasible, the algorithm proposed in [4] concludes that the circuit is infeasible at any clock period and stop. For example, let us determine the minimum feasible clock period of the circuit shown in Fig. 1(a). Here, our target clock-period range δ is 0. Initial lower bound $L = 4$ and initial upper bound $U = 8$. So, the algorithm does binary search between 4 and 8 as follows:

1) Check 4(L)**:** Since negative cycle $C^1 = (u, w2, v2, u)$ with $w(C^1) = 4T - 20$ is found, $G^1(4)$ is infeasible. So the next step is check U.

2) Check 8(U)**:** Since negative cycle $C^2 = (u, v1, w1, u)$ with $w(C^2) = -T + 6$ is found, $G^1(8)$ is infeasible and the algorithm is stopped.

The check sequence of the algorithm is shown in Fig. 1(c). As you can see from the above example, the algorithm is stopped after checking the initial upper bound U and concludes that the circuit is infeasible at any clock period.

However as we mentioned earlier, the conclusion is correct for a circuit that contains just single-clock cycle paths, while for the circuit that contains multi-clock cycle paths the conclusion might be incorrect. Therefore, the above approach might miss the minimum feasible clock period. In fact, in this case, the algorithm cannot determine the minimum feasible clock period which is 5.

We enhance the algorithm that has been introduced in [4] to determine the minimum feasible clock period of a circuit that contains multi-clock cycle paths. The enhanced algorithm does binary search between lower and upper bounds same as in the algorithm in [4]. We extend the algorithm in [4] by introducing checking the type of cycle when a negative cycle is found in the constraint graph $G_\delta(t)$. If the circuit is infeasible at given clock period t, a negative cycle is found in $G_\delta(t)$. The lower and upper bounds are adjusted based on the type of the found negative cycle and the *Bound* value.

The new algorithm to determine the minimum feasible clock period of a circuit that contains multi-clock cycle

Input: constraint graph $G_\delta(t)$
Output: minimum feasible clock period T.

1. $L_{\text{self}} := \max_{(u,u)\in E_{hold}} d_{\max}(u,u)$.
   $L_{\text{diff}} := \max_{(u,v)\in E_{hold}} (d_{\max}(u,v) - d_{\min}(u,v))$.
   $L := \max\{L_{\text{self}}, L_{\text{diff}}\}$.
   $U := \max_{(u,v)\in E_{hold}} \{(d_{\max}(u,v) + s(u) - s(v))/\beta_{u,v}\}$.
2. Check whether $G_\delta(L)$ is feasible.
   If there is no negative cycle in $G_\delta(L)$ return L,
   else if there exists a negative cycle C of type 0 or type M return ∞.
3. Check whether $G_\delta(U)$ is feasible.
   If there exists a negative cycle C:
   Case C is of type 0 return ∞.
   Case C is of type P then repeat the following:
   $L := Bound(C)$.
   If there is no negative cycle in $G_\delta(L)$ return L,
   else if there exists a negative cycle C'.
   If C' is of type 0 or type M return ∞,
   else $C \leftarrow C'$.
   Case C is of type M then $U := Bound(C)$ and
   if $U < L$ return ∞.
4. While $(U - L > \varepsilon)$ do:
   $H := (U + L)/2$.
   Check whether $G_\delta(H)$ is feasible.
   If there is no negative cycle in $G_\delta(H)$ then $U := H$,
   else let C be a negative cycle:
   Case C is of type 0 return ∞.
   Case C is of type P then:
   $L := Bound(C)$.
   If there is no negative cycle in $G_\delta(L)$ return L,
   else if there exists a negative cycle C' of type 0
   or type M return ∞.
   Case C is of type M then $U := Bound(C)$.
   endwhile.
5. $T := U$. return T.

Fig. 3 Minimum feasible clock period algorithm of the circuit that contains multi-clock cycle paths.

paths is shown in Fig. 3.

For the initial value of lower bound L and upper bound U of the binary search, we adopt the same approach as in the algorithm shown in [4]. Initial lower bound L will be checked whether it is feasible or not, if L is feasible, then output L as the minimum feasible clock period. Otherwise, a negative cycle C is found. If C is of **type 0** or **type M**, the circuit is infeasible and the algorithm is stopped. While, if C is of **type P** then the initial upper bound U will be checked whether it is feasible or not.

If the initial upper bound U is feasible, then the algorithm does binary search to determine the minimum feasible clock period. Otherwise, a negative cycle C is found. In case C is of **type 0**, the circuit is infeasible and the algorithm is stopped. In case C is of **type P**, $Bound(C)$ is our new lower bound L and L will be checked whether it is feasible or not. If our new lower bound L is feasible then output L as the minimum feasible clock period. Otherwise, repeat until L is feasible or a found negative cycle is of **type 0** or **type M**, where the circuit is infeasible and the algorithm is stopped. In case C is of **type M**, $Bound(C)$ is our new upper bound U and the algorithm will check whether $U < L$ or not. If

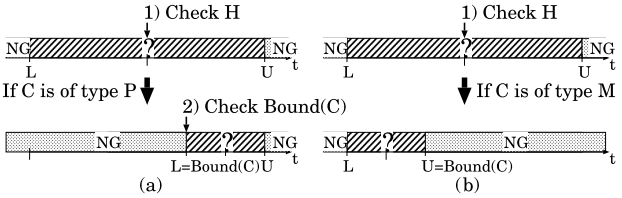

Fig. 4 (a) If the found negative cycle C is of **type P** then $L = Bound(C)$ and check L. (b) If the found negative cycle C is of **type M** then $U = Bound(C)$.

$U < L$, then the circuit is infeasible and the algorithm is stopped. Otherwise, our algorithm does binary search by adjusting the lower and upper bounds to determine the minimum feasible clock period.

In binary search, the algorithm will check whether the constraint graph $G_\delta(H)$ (where $H = (U + L)/2$) contains any negative cycle or not. If there are no negative cycles in $G_\delta(H)$, then H is our new upper bound U and continue do binary search. Otherwise, negative cycle C is found in $G_\delta(H)$ and the algorithm will check the type of it. In case C is of **type 0**, the circuit is infeasible and the algorithm is stopped. In case C is of **type P**, $Bound(C)$ is our new lower bound L from Theorem 2, and L will be checked whether it is feasible or not (Refer Fig. 4(a)). If our new lower bound L is feasible then output L as the minimum feasible clock period, otherwise, continue do binary search. In case C is of **type M**, $Bound(C)$ is our new upper bound U (Refer Fig. 4(b)), and continue do binary search.

Using the proposed algorithm, let us find the minimum feasible clock period of the circuit shown in Fig. 1(a). Here, our target clock-period range δ is 0. Initial lower bound $L = 4$ and initial upper bound $U = 8$. So, the algorithm does binary search between 4 and 8 as follows:

1) Check $4(L)$**:** Since negative cycle $C^1 = (u, w2, v2, u)$ with $w(C^1) = 4T - 20$ is found, $G^1(4)$ is infeasible. C^1 is of **type P**, so the next step is check U.
2) Check $8(U)$**:** Since negative cycle $C^2 = (u, v1, w1, u)$ with $w(C^2) = -T + 6$ is found, $G^1(8)$ is infeasible. C^2 is of **type M** and $Bound(C^2) = 6$, therefore 6 is our new U and $H = 5$.
3) Check $5(H)$**:** $G^1(5)$ is feasible. Since $U - L = 1$, output 5 as the minimum feasible clock period T.

The check sequence of the algorithm is shown in Fig. 1(d). The algorithm can determine the minimum feasible clock period of the circuit which is 5.

In Step 4 of the proposed algorithm, when H is checked and the type of the found negative cycle is of **type P**, the lower bound L is updated and checked. However, in Step 2 and Step 4, when L is checked and the type of the found negative cycle is of **type P**, the lower bound L has not been updated. The reason is, if we just update the lower bound L value without checking whether it is feasible or not, the checking of this value is postponed until binary search range reaches the precision ε. Unluckily, if that updated value is the minimum feasible clock period, it increases the num-

ber of checking times. While, if the updated lower bound L value is checked and it is infeasible due to **type P** negative cycle and this is repeated, then the number of checking times increases, especially, when the amount of the increasing of the lower bound L is small. Therefore, in our proposed algorithm, when the lower bound L is checked and it is infeasible, we do not update the lower bound L value based on the found negative cycle.

4. Reduction on the Number of Intermediate Registers

In this paper we consider a problem on how to reduce the number of intermediate registers of a pipelined circuit, subject to the minimum feasible clock period is lower than or equal to the original circuit and works at target clock-period range. In the proposed algorithm, all intermediate registers of the pipelined circuit are initially removed. Then the minimum feasible clock period is computed using the algorithm shown in Fig. 3.

A D-edge (Z-edge) which corresponds to the single-clock cycle path is called single D-edge (single Z-edge). While D-edge (Z-edge) which corresponds to the multi-clock cycle path is called multi D-edge (multi Z-edge). When an intermediate register is removed, the weight of Z-edge and D-edge of the constraint graph are changed, as well as its topology. As for example, when an intermediate register v is removed, single-clock cycle paths (u, v) and (v, w) whose total minimum (maximum) delay is σ (σ') become two-clock cycle path (u, w) whose minimum (maximum) delay is $\sigma - \omega$ ($\sigma' - \omega'$), where ω ($\omega > 0$) and ω' are the minimum and maximum delay of the intermediate register v, respectively. In the constraint graph of the circuit, the single D-edges (u, v) and (v, w) whose total weight is σ are removed and a multi D-edge (u, w) whose weight is $\sigma - \omega - T$ is inserted. Similarly, the single Z-edges (w, v) and (v, u) whose total weight is $2T - \sigma'$ are removed and a multi Z-edge (w, u) whose weight is $2T - \sigma' - \omega'$ is inserted. The weight of the multi D-edge is reduced compared with the corresponding original single D-edges. While the weight of the multi Z-edge is increased compared with the corresponding original single Z-edges.

When an intermediate register is removed, the obtained circuit may become infeasible or the minimum feasible clock period of the obtained circuit may be increased. If the circuit is infeasible at any clock period, a negative cycle is found in the constraint graph and it contains a multi D-edge. When the intermediate register which corresponds to a multi D-edge is inserted, the number of multi D-edges in the found negative cycle is reduced, and the negative cycle is eliminated by repeated intermediate register insertion. If the minimum feasible clock period of the circuit is increased, a critical cycle is found in the constraint graph and it contains a multi D-edge. When the intermediate register which corresponds to a multi D-edge is inserted, the number of multi D-edges in the found critical cycle is reduced, and the critical cycle is eliminated by repeated intermediate register insertion.

Inputs : Constraint graph G^{in} of a pipelined circuit with intermediate registers, the minimum clock period (zero clock-skew) T_{comp} of the original circuit and clock-period range δ.
Outputs : Constraint graph G_0^{out} of the circuit after removing the intermediate registers, clock timing $s(u)$ and minimum clock period $T_{\min}(G_0^{out})$.
Step 0 : Remove all of the intermediate registers. Let G_δ be the constraint graph of the obtained circuit.
Step 1 : Compute the minimum clock period of the constraint graph G_δ by the algorithm shown in Fig. 3. If the output of the algorithm is ∞ then a negative cycle C is found, and insert the intermediate register to the multi-clock cycle path which corresponds to a multi D-edge contained in C, and update G_δ. Repeat this step until the output of the algorithm is not ∞.
Step 2 : Let G_δ^1 be the constraint graph of the obtained circuit.
Step 3 : Compute the minimum clock period of the constraint graph G_δ^1 by the algorithm shown in Fig. 3. If $T_{\min}(G_\delta^1) > T_{comp}$ then a critical cycle C^1 is found, and insert the intermediate register to the multi-clock cycle path which corresponds to a multi D-edge contained in C^1, and update G_δ^1. Repeat this step until $T_{\min}(G_\delta^1) \le T_{comp}$.
Step 4 : Let G_0^{out} be the constraint graph of the obtained circuit and output G_0^{out}, $T_{\min}(G_0^{out})$ and the clock timing for all registers.

Fig. 5 Algorithm to reduce the number of intermediate registers.

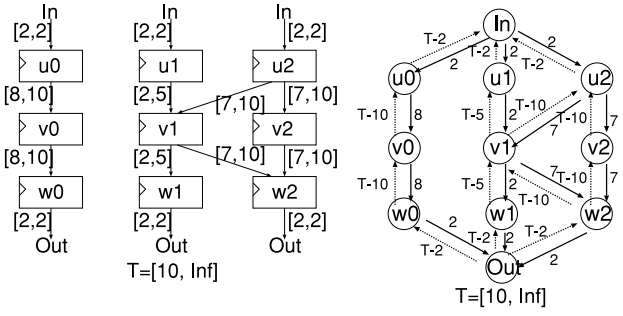

Fig. 6 Pipelined circuit with intermediate registers and the corresponding constraint graph G^{in}. $T_{comp} = 10$.

Our algorithm is heuristic. A different circuit might be obtained depending on the found negative and critical cycle. Furthermore, if there are more than one multi D-edge in negative or critical cycle, our algorithm chooses one of them randomly. Thus, a different circuit might be obtained depending on the chosen multi D-edge.

The details of our proposed algorithm is shown in Fig. 5. Note that, in Step 1 and Step 3 of our proposed algorithm, if there is no more multi D-edge, the algorithm is stopped and output the input constraint graph.

4.1 Example

To explain the behavior of the algorithm, we apply the algorithm to the pipelined circuit shown in Fig. 6. In this example, the timing of each I/O pin is fixed at 0, while the timing of each register is scheduled. We also assume that Setup and Hold Time for registers are 0 and the minimum and maximum delay of the intermediate registers are 1 and 2, respectively. Here, our target clock-period range δ is 4. Note that in the constraint graph, vertices *In* and *Out* are the same vertex because we fix the timings of I/O pins to 0. In

the figure, vertices *In* and *Out* are drawn in different vertex to make it easy to understand. For the original circuit with zero clock-skew, the minimum feasible clock period T_{comp} is 10. The circuit after removing the intermediate registers $v0$, $v1$, $v2$ is shown in Fig. 7. The minimum clock period of the constraint graph G_4^0 is computed by the minimum clock period algorithm. In the binary search, when $T = 9$, negative cycle $C^0 = (in, u1, w1, out)$ is found in the constraint graph G_4^0, which is of **type M**. The minimum clock period algorithm concludes that it is infeasible. Actually the circuit works at clock period 6 but δ is not secured. Since edge $(u1, w1)$ is a multi D-edge, intermediate register $v1$ is inserted between the multi-clock cycle path $(u1, w1)$. The circuit after inserting the intermediate register $v1$ is shown in Fig. 8. G_4^1 is the constraint graph of the obtained circuit. The minimum feasible clock period of the obtained circuit is 11, and critical cycle $C^1 = (u2, w2, v1, u2)$ is found in

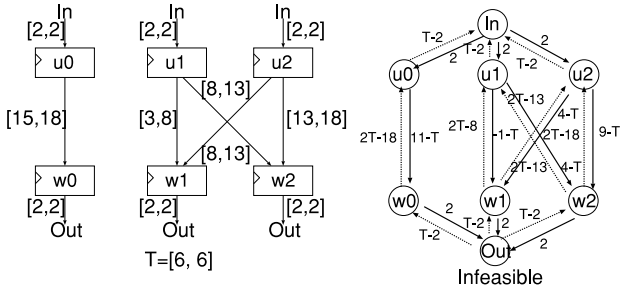

Fig. 7 Pipelined circuit after removing all intermediate registers and the corresponding constraint graph G_4^0.

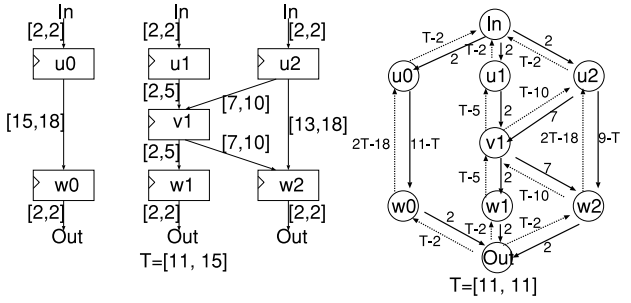

Fig. 8 Pipelined circuit after inserting the intermediate register $v1$ and the corresponding constraint graph G_4^1. $T_{\min}(G_4^1) = 11 > T_{comp} = 10$.

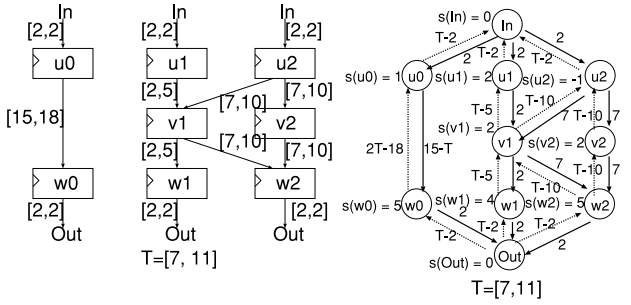

Fig. 9 Pipelined circuit after inserting the intermediate registers $v1$ and $v2$, and the corresponding constraint graph G_0^2. $T_{\min}(G_0^2) = 7$.

the constraint graph $G_4^1(11)$ by the minimum clock period algorithm during the binary search. Since $11 > T_{comp}$ and edge $(u2, w2)$ is a multi D-edge, intermediate register $v2$ is inserted between the multi-clock cycle path $(u2, w2)$. The circuit after inserting the intermediate register $v2$ is shown in Fig. 9. G_4^2 is the constraint graph of the obtained circuit. The minimum feasible clock period of the obtained circuit is 7, which is less than T_{comp}, so the algorithm stop and output the circuit and clock timings as shown in Fig. 9. Note that in Fig. 9, the clock timings are computed based on the constraint graph G_0^2. The output circuit works correctly between 7 to 11, while the original circuit works correctly between 10 to ∞.

5. Experiments

The proposed algorithms were written in C++ and implemented on a Pentium 4 (CPU 3 GHz, memory 513764 kb). Since there are no benchmark examples of pipelined circuits, two simple examples, briefly described below, were constructed for our experiments.

- n-bit ($n = 4, 8, 16$) add: A 2-stage adder that added four n-bit numbers (A, B, C and D) [7]. The first stage computed the partial sum A + B and C + D and the second stage computed the final sum. Each adder was of ripple-carry type.
- 16-bit mul: A 2-stage multiplier that multiplied two 16-bit numbers. The first stage used a carry-save adder with Wallace tree structure [8] and the second stage used a carry-look-ahead adder.

The statistics of the circuits are shown in Table 1. The ROHM 0.35 μm process library was used for these experiments. The timing of each I/O pin was scheduled as well as the timing for each register.

Table 2 shows the results when the algorithm shown in Sect. 4 was applied. Ori. is the original circuit containing the intermediate registers and the clock timing of all registers are fixed at 0 (zero clock-skew). "δ [ps]" and "$T_{\min}$ [ps]" are the target clock-period range and the output minimum feasible clock period, respectively. "Int. FF (#)" is the number of intermediate registers, and "Int. FF (%)" is the percentage of the number of intermediate registers present compared with the total in the original circuit. Delay variation was 20% (50%), i.e. the delay variation for each gate and register was

Table 1 Statistics of adder and multiplier.

		Circuit delay [ps]			
circuit	# FF	1st stage		2nd stage	
		min	max	min	max
4 bitadd	32	588	2454	588	2840
8 bitadd	60	598	4079	598	4474
16 bitadd	116	598	7239	598	7634
16 bitmul	120	757	5075	373	4050

Table 2 Experimental results.

		Delay variation = 0%					Delay variation = 20%						Delay variation = 50%					
Circuit	δ	$T_{\min}$		Int. FF		Time	$T_{\min}$			Int. FF		Time	$T_{\min}$			Int. FF		Time
	[ps]	[ps]	(%)[a]	#	(%)	[s]	[ps]	(%)[a]	(%)[b]	#	(%)	[s]	[ps]	(%)[a]	(%)[c]	#	(%)	[s]
4 bitadd	Ori.	2840	(100)	10	(100)	-	3093	(110)	(100)	10	(100)	-	3472	(123)	(100)	10	(100)	-
	0	1422	(50)	0	(0)	0.01	1929	(68)	(62)	0	(0)	0.01	2691	(95)	(78)	0	(0)	0.01
	200	1622	(57)	0	(0)	0.01	2129	(75)	(69)	0	(0)	0.01	1986	(70)	(57)	10	(100)	0.02
	400	1822	(64)	0	(0)	0.01	2329	(82)	(75)	0	(0)	0.01	1986	(70)	(57)	10	(100)	0.02
	600	2022	(71)	0	(0)	0.01	2529	(89)	(82)	0	(0)	0.01	1986	(70)	(57)	10	(100)	0.02
	800	2222	(78)	0	(0)	0.01	2729	(96)	(88)	0	(0)	0.01	1986	(70)	(57)	10	(100)	0.02
	1000	2422	(85)	0	(0)	0.01	1295	(46)	(42)	10	(100)	0.02	1986	(70)	(57)	10	(100)	0.02
8 bitadd	Ori.	4474	(100)	18	(100)	-	4890	(110)	(100)	18	(100)	-	5515	(123)	(100)	18	(100)	-
	0	1702	(38)	0	(0)	0.02	2509	(56)	(51)	0	(0)	0.02	3720	(83)	(68)	0	(0)	0.02
	200	1902	(43)	0	(0)	0.02	2709	(61)	(55)	0	(0)	0.02	3920	(88)	(71)	0	(0)	0.02
	400	2102	(47)	0	(0)	0.02	2909	(65)	(59)	0	(0)	0.02	4120	(92)	(75)	0	(0)	0.02
	600	2302	(51)	0	(0)	0.02	3109	(69)	(64)	0	(0)	0.02	4320	(97)	(78)	0	(0)	0.02
	800	2502	(56)	0	(0)	0.02	3309	(74)	(68)	0	(0)	0.02	3007	(67)	(55)	18	(100)	0.06
	1000	2702	(60)	0	(0)	0.02	3509	(78)	(72)	0	(0)	0.02	3007	(67)	(55)	18	(100)	0.06
16 bitadd	Ori.	7634	(100)	34	(100)	-	8366	(110)	(100)	34	(100)	-	9465	(124)	(100)	34	(100)	-
	0	2199	(29)	0	(0)	0.06	3588	(47)	(43)	0	(0)	0.06	5673	(74)	(60)	0	(0)	0.06
	200	2399	(31)	0	(0)	0.06	3788	(50)	(45)	0	(0)	0.06	5873	(77)	(62)	0	(0)	0.06
	400	2599	(34)	0	(0)	0.06	3988	(52)	(48)	0	(0)	0.06	6073	(80)	(64)	0	(0)	0.06
	600	2799	(37)	0	(0)	0.06	4188	(55)	(50)	0	(0)	0.06	6273	(82)	(66)	0	(0)	0.06
	800	2999	(39)	0	(0)	0.06	4388	(57)	(52)	0	(0)	0.06	6473	(85)	(68)	0	(0)	0.06
	1000	3199	(42)	0	(0)	0.06	4588	(60)	(55)	0	(0)	0.06	6673	(87)	(71)	0	(0)	0.06
16 bitmul	Ori.	5075	(100)	56	(100)	-	5551	(110)	(100)	56	(100)	-	6266	(124)	(100)	56	(100)	-
	0	4870	(96)	11	(20)	0.72	3590	(71)	(65)	49	(88)	2.70	4466	(88)	(71)	49	(88)	2.70
	200	5070	(100)	11	(20)	0.73	3590	(71)	(65)	49	(88)	2.69	4466	(88)	(71)	49	(88)	2.72
	400	5069	(100)	35	(63)	1.97	3590	(71)	(65)	49	(88)	2.73	4470	(88)	(71)	49	(88)	2.72
	600	5071	(100)	48	(86)	2.53	3750	(74)	(68)	49	(88)	2.70	4670	(92)	(75)	49	(88)	2.75
	800	3336	(66)	49	(88)	2.61	3950	(78)	(71)	49	(88)	2.71	4870	(96)	(78)	49	(88)	2.73
	1000	3536	(70)	49	(88)	2.59	4150	(82)	(75)	49	(88)	2.73	5070	(100)	(81)	49	(88)	2.75

[a]Compared with original circuit with zero clock-skew and delay variation = 0%.
[b]Compared with original circuit with zero clock-skew and delay variation = 20%.
[c]Compared with original circuit with zero clock-skew and delay variation = 50%.

set at ±10% (±25%). In the circuit 16 bitmul of 20% delay variation, the maximum delay variation between registers is 782 [ps]. The target clock-period range δ is set from 0 [ps] to 1000 [ps]. "Time[s]" is the computation time of the proposed algorithm.

The results show that by a combination of multi-clock cycles and clock scheduling, the number of intermediate registers and the minimum feasible clock period can be reduced, even in the presence of delay variations in gates and registers.

Note that, our algorithm can determine efficiently which register must be recover back, in order to reduce the minimum feasible clock period of the circuit. As you can see from the circuit 16 bitmul (delay variation = 0%, δ = 600 [ps], 800 [ps]), by recovering back only one register, the minimum feasible clock period can be reduced from 5071 [ps] to 3336 [ps].

The relation between the minimum feasible clock period and the number of intermediate registers of the 16 bit multiplier is shown in Fig. 10. In the graph, the label "Algorithm" indicates results using the proposed algorithm for insertion of the intermediate registers, while "Random1-4" labels indicate results when the intermediate registers are inserted randomly. The graph shows that the proposed algorithm can construct an equivalent circuit using fewer registers and with a smaller minimum feasible clock period.

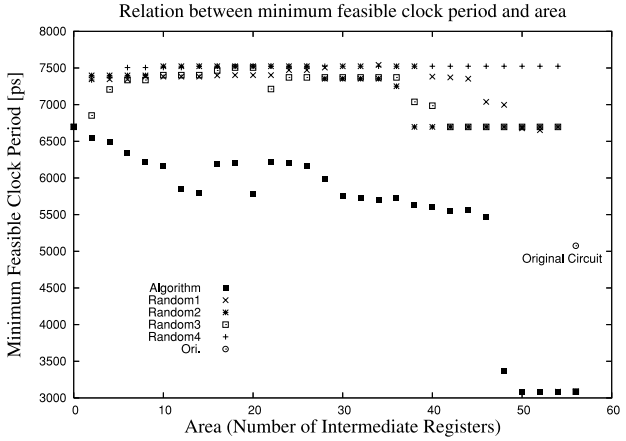

Fig. 10 Relation between $T_{\min}$ [ps] and # Int. FF of a 16 bit multiplier. $\delta = 1000$ [ps]. Delay variation = 0%.

6. Conclusion

It has been shown that the number of intermediate registers of a pipelined circuit can be reduced by implementing a multi-clock cycle path technique together with clock scheduling. The proposed algorithm inserts intermediate registers without considering delay balancing in order to make the circuit works correctly throughout the target clock-period range. We believe that by using delay balancing together with intermediate register insertion, circuit area can be further reduced. This is a topic for future investigation.

Acknowledgements

This work is supported by VLSI Design and Education Center (VDEC), the University of Tokyo in collaboration with Synopsys, Inc., Cadence Design Systems, Inc., Rohm Corporation and Toppan Printing Corporation.

References

[1] W.J. Kim and Y. Kim, "Clocking for correct functionality on wave pipelined circuits," Proc. IEEE International ASIC/SOC Conference, pp.161–164, 2003.
[2] J.P. Fishburn, "Clock skew optimization," IEEE Trans. Comput., vol.39, no.7, pp.945–951, 1990.
[3] R.B. Deokar and S.S. Sapatnekar, "A graph-theoretic approach to clock skew optimization," Proc. International Symposium on Circuits and Systems (ISCAS), pp.407–410, 1994.
[4] A. Takahashi, "Practical fast clock schedule design algorithms," Proc. 18th Karuizawa Workshop, pp.515–520, 2005.
[5] A. Takahashi and Y. Kajitani, "Performance and reliability driven clock scheduling of sequential logic circuits," Proc. Asia and South Pacific Design Automation Conference (ASP-DAC), pp.37–42, 1997.
[6] B.A. Rosdi and A. Takahashi, "Reduction on the usage of intermediate registers for pipelined circuits," Proc. Workshop on Synthesis and System Integration of Mixed Technologies (SASIMI 2004), pp.333–338, 2004.
[7] S. Malik, K.J. Singh, R.K. Brayton, and A. Sangiovanni-Vincentelli, "Performance optimization of pipelined circuits," Proc. IEEE/ACM International Conference on Computer Aided Design (ICCAD), pp.410–413, 1990.
[8] C. Wallace, "A suggestion for fast multiplier," IEEE Trans. Electronic Computers, vol.13, no.2, pp.14–17, 1964.

Bakhtiar Affendi Rosdi received his B.E., and M.E. degrees in electrical and electronic engineering from Tokyo Institute of Technology, Tokyo, Japan, in 1999 and 2004, respectively. He is currently a D.E. student of Department of Communications and Integrated Systems in Tokyo Institute of Technology. His research interests are in VLSI design automation and combinational algorithms.

Atsushi Takahashi received his B.E., M.E., and D.E. degrees in electrical and electronic engineering from Tokyo Institute of Technology, Tokyo, Japan, in 1989, 1991, and 1996, respectively. He had been with the Tokyo Institute of Technology as a research associate from 1991 to 1997 and has been an associate professor since 1997. He visited University of California, Los Angeles, U.S.A., as a visiting scholar from 2001 to 2002. He is currently with Department of Communications and Integrated Systems, Graduate School of Science and Engineering, Tokyo Institute of Technology. His research interests are in VLSI layout design and combinational algorithms. He is a member of IEEE and IPSJ.