# Low Area Pipelined Circuits by Multi-clock Cycle Paths and Clock Scheduling

Bakhtiar Affendi Rosdi, Atsushi Takahashi
Department of Communications and Integrated Systems, Tokyo Institute of Technology
2-12-1-S3-58 Ookayama, Meguro-ku, Tokyo 152-8552 Japan
Tel:+81-3-5734-2665 Fax:+81-3-5734-2902
E-mail:{fendi, atushi}@lab.ss.titech.ac.jp

**Abstract— 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.**

## I. 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 tradeoff 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. 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. The enhanced minimum clock-period algorithm uses the existence of a negative cycle to narrow the binary search interval efficiently. A negative cycle exists whenever the constraints are infeasible. 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. In the proposed algorithm, all intermediate registers of the pipelined circuit are initially removed. Then the minimum feasible clock period of the resulting circuit 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.

0-7803-9451-8/06/$20.00 

## II. 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 Constraint**

$$s(u) - s(v) \leq \beta_{u,v}T - d_{\max}(u,v) \tag{1}$$

**Hold Constraint**

$$s(v) - s(u) \leq d_{\min}(u,v) - \alpha_{u,v}T \tag{2}$$

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. The above constraints become:

**Setup Constraint**

$$s(u) - s(v) \leq \beta_{u,v}T_{\min} - d_{\max}(u,v) \tag{3}$$

**Hold Constraint**

$$s(v) - s(u) \leq d_{\min}(u,v) - \alpha_{u,v}T_{\min} - \alpha_{u,v}\delta \tag{4}$$

where $\delta$ is the clock-period range, i.e. $\delta = T_{\max} - T_{\min}$. Therefore if $\delta$ 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 (3) and (4) 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 $\beta_{u,v}T - d_{\max}(u,v)$($d_{\min}(u,v) - \alpha_{u,v}\delta - \alpha_{u,v}T$). The constraint graph $G$ corresponding to clock period $t$ is denoted by $G_t$.

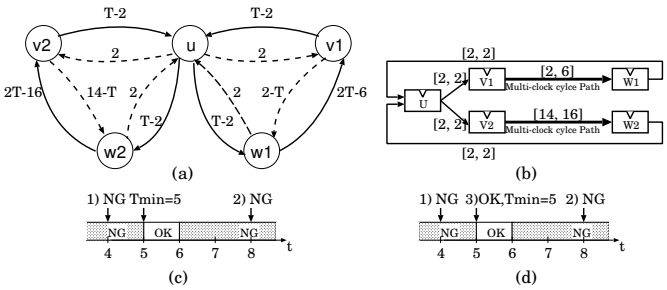


Fig. 1. (a) Constraint graph $G^1$. (b) Circuit graph $CG^1$ (containing multi-clock cycle paths). (c) Min clock-period computation by the algorithm shown in [4]. (d) Min clock-period computation by proposed algorithm.

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 $\infty$, according to whether $C$ is of **type P**, **type M**, or **type 0**, respectively.*

Example: For the constraint graph $G^1$ shown in Fig. 1 (a), 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 $C_2 = (u, v1, w1, u)$ 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.

## III. Minimum Feasible Clock Period

### A. 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.

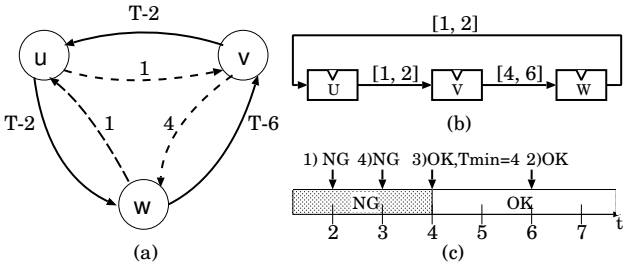


Fig. 2. (a) Constraint graph $G^2$. (b) Circuit graph $CG^2$ (containing just single-clock cycle paths). (c) Min clock-period computation by the algorithm shown in [4].

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 [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 there exists a negative cycle of **type 0**, 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.

Using the algorithm shown in [4] let us determine the minimum feasible clock period of the circuit shown in Fig. 2 (b). Note that, throughout this paper, the precision used is 1. 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:

**Check** $L(L = 2)$**:** $G_2^2$ is infeasible, cycle $C_1 = (u, w, v, u)$ with $w(C_1) = 3T - 10$ is negative, so next step is check $U$.

**Check** $U(U = 6)$**:** $G_6^2$ is feasible, so next step is check $M = (U + L/2)$.

**Check** $M(M = 4 = (6 + 2/2))$**:** $G_4^2$ is feasible, so 4 becomes new upper bound $U$.

**Check** $M(M = 3 = (4 + 2/2))$**:** $G_3^2$ is infeasible, cycle $C_1 = (u, w, v, u)$ with $w(C_1) = 3T - 10$ is negative. Since $U - L = 1$, output 4 as the minimum feasible clock period $T$.

The flow when we apply the algorithm is shown in Fig. 2 (c).

### *B. A circuit that contains multi-clock cycle paths*

For a circuit that contains just single-clock cycle paths, if the circuit is feasible then the circuit is feasible at the initial upper bound $U$, otherwise the circuit is infeasible. 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 $t$ ($t < U$or $t > U$).

**Input:** circuit graph $CG$, target clock-period range $\delta$.

**Output:** minimum feasible clock period $T$.

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

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

When the initial upper bound $U$ is infeasible, the algorithm [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 (b). Initial lower bound $L = 4$ and initial upper bound $U = 8$. So, the algorithm does binary search between 4 and 8 as follows:

**Check** $L(L = 4)$**:** $G_4^1$ is infeasible, cycle $C_1 = (u, w2, v2, u)$ with $w(C_1) = 4T - 20$ is negative, so next step is check $U$.

**Check** $U(U = 8)$**:** $G_8^1$ is infeasible, cycle $C_2 = (u, v1, w1, u)$ with $w(C_2) = -T + 6$ is negative and the algorithm is stopped.

The flow when we apply the algorithm is shown in Fig. 1 (c). As you can see from the above example, the algorithm [4] 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 algorithm does binary search between lower and upper bounds

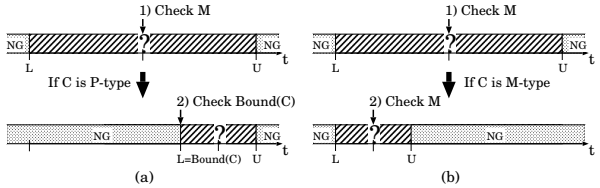


Fig. 4. (a) If $C$ is of **type P** then $L = Bound(C)$ and check $L$. (b) If $C$ is of **type M** then $U = Bound(C)$.

same as in the algorithm shown in [4]. We extend the algorithm [4] by introducing checking the type of cycle when there exists a negative cycle in the constraint graph. If the circuit is infeasible at given clock period, there exists a negative cycle. The lower and upper bounds are adjusted based on the type of negative cycle and the *Bound* value.

The new algorithm to determine the minimum feasible clock period of a circuit that contains multi-clock cycle 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, there exists a negative cycle $C$. 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 an 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, there exists a negative cycle $C$. 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 $C$ 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 $U < L$, 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 $G_M(M = (U + L)/2)$ containing any negative cycle or not. If there are no negative cycles in $G_M$, then $M$ is our new upper bound $U$ and continue do binary search. Otherwise, if there exists a negative cycle in $G_M$ then the algorithm will check the type of it. In case $C$ is of **type 0**, the circuit is infeasible and the algorithm is stopped. From Theorem 2, in case $C$ is of **type P** then $Bound(C)$ is our new lower bound $L$ 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** then $Bound(C)$ 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 (b). Our target clock-period range $\delta$ 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:

**Check** $L(L = 4)$**:** $G_4^1$ is infeasible, cycle $C_1 = (u, w2, v2, u)$ with $w(C_1) = 4T - 20$ is negative and of **type P**. So next step is check $U$.

**Check** $U(U = 8)$**:** $G_8^1$ is infeasible, cycle $C_2 = (u, v1, w1, u)$ with $w(C_2) = -T + 6$ is negative and of **type M** and $Bound(C_2) = 6$, therefore 6 is our new $U$.

**Check** $M(M = 5 = (6 + 4/2))$**:** $G_5^1$ is feasible. Since $U - L = 1$, output 5 as the minimum feasible clock period $T$.

The flow when we apply the algorithm is shown in Fig. 1 (d). The algorithm can determine the minimum feasible clock period of the circuit which is 5.

## IV. 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. When the intermediate registers are removed, the minimum delay is reduced compared with the original circuit. Therefore, the hold constraints which correspond to the D-edges in the cycles become tight. If the circuit is infeasible at any clock period, there exists a negative cycle in the constraint graph. As we mentioned earlier, the hold constraints are the reason why the timing constraints become tight. Therefore to make the circuit feasible at target clock-period range, an intermediate register which corresponds to a D-edge contained in the negative cycle is inserted to the multi-clock cycle path.

Our target is to get a circuit, where the minimum feasible clock period is lower than or equal to the original circuit. If the minimum feasible clock period of the obtained circuit is higher than the original circuit, the minimum feasible clock period needs to be reduced. Our proposed algorithm find a critical cycle in the constraint graph that decide the minimum feasible clock period of the obtained circuit. Then, in order to reduce the minimum feasible clock period, an intermediate register which corresponds to a D-edge contained in the critical cycle is inserted to the multi-clock cycle path. The details of our proposed algorithm is omitted here due to the space constraint.

To explain the behavior of the algorithm, we implemented the algorithm to the pipelined circuit shown in Fig. 5. 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. Our target clock-period range $\delta$ 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 draw in different vertex to make it easy

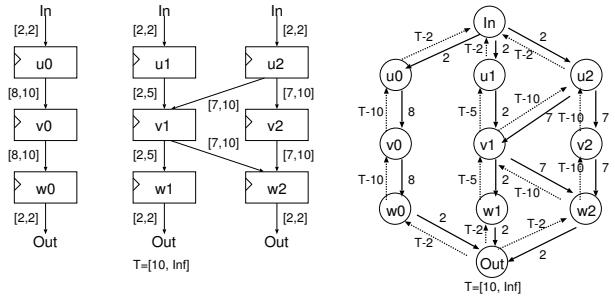


Fig. 5. Pipelined circuit with intermediate registers and the corresponding constraint graph $G^{in}$. $T_{\min}(G^{in}) = 10$.

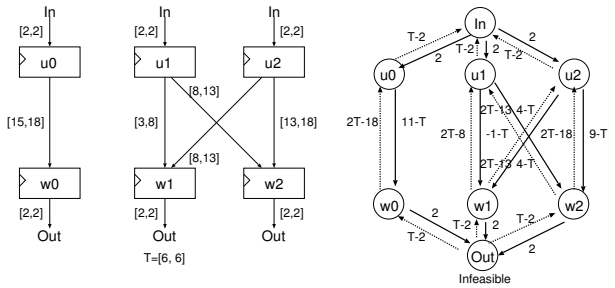


Fig. 6. Pipelined circuit after removing all intermediate registers and the corresponding constraint graph $G^0$ (D-edge weight of the multi-clock cycle paths in $G^0$ is reduced 4 compared with the corresponds minimum delay in the circuit).

to understand. For the original circuit with zero clock-skew, the minimum feasible clock period $T_{\min}(G^{in})$ is 10. The circuit after removing the intermediate registers $v_0$, $v_1$, $v_2$ is shown in Fig. 6. Note that the D-edge weight of the multi-clock cycle paths in the constraint graph is reduced 4 compared with the corresponds minimum delay in the circuit because $\delta$ is 4. When the minimum clock period of constraint graph $G^0$ is computed, there exists a negative cycle $C^0 = (in, u_1, w_1, out)$ which is of **type M** in the constraint graph $G^0_9$. The minimum clock period algorithm concludes that it is infeasible. This means that the circuit works at clock period 6 but $\delta$ is not secured. Since path $(u_1, w_1)$ is a D-edge which is multi-clock cycle path, intermediate register $v_1$ is inserted. The circuit after inserting the intermediate register $v_1$ is shown in Fig. 7. $T_{\min}(G^1) = 11 > T_{\min}(G^{in})$, therefore the minimum feasible clock period of the obtained circuit need to be reduced. The critical cycle $C^1 = (u2, w2, v1, u2)$ is found in the constraint graph $G^1_{11}$. Since path $(u_2, w_2)$ is a D-edge which is multi-clock cycle path, intermediate register $v_2$ is inserted. The circuit after inserting the intermediate register $v_2$ is shown in Fig. 8. $T_{\min}(G^2) = 7 < T_{\min}(G^{in})$, so the algorithm stop and output the circuit and clock timings as shown in Fig. 8.

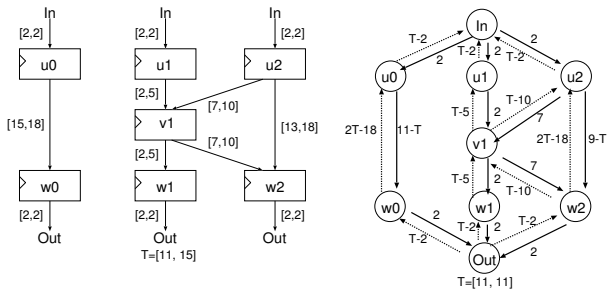


Fig. 7. Pipelined circuit after inserting the intermediate register $v_1$ and the corresponding constraint graph $G^1$ (D-edge weight of the multi-clock cycle paths in $G^1$ is reduced 4 compared with the corresponds minimum delay in the circuit). $T_{\min}(G^1) = 11) > (T_{\min}(G^{in}) = 10$.

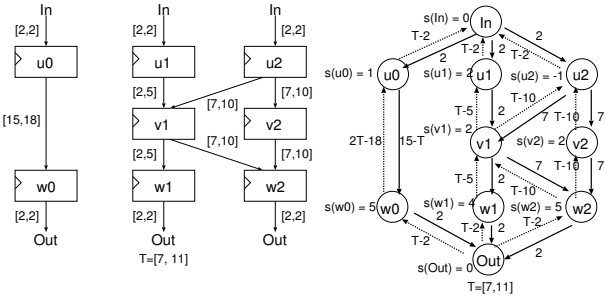


Fig. 8. Pipelined circuit after inserting the intermediate registers $v_1$ and $v_2$, and the corresponding constraint graph $G^2$. $T_{\min}(G^2) = 7$.

TABLE I
STATISTICS OF ADDER AND MULTIPLIER

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

## V. EXPERIMENTS

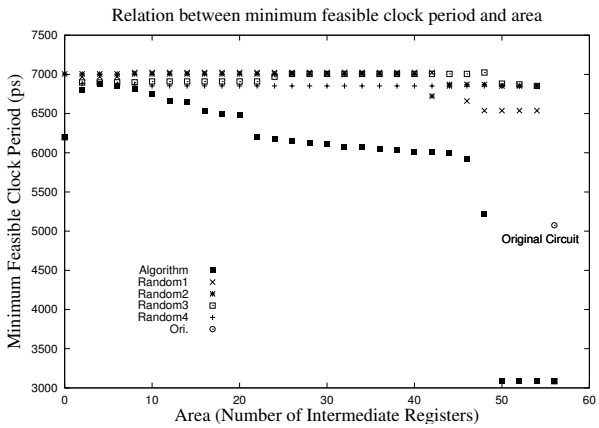


Fig. 9. Relation between $T_{\min}(ps)$ and # Int. FF of a 16 bit multiplier. $\delta = 500ps$. Delay variation = 0%.

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

TABLE II
EXPERIMENTAL RESULTS

| | | Delay variation = 0% | | | | Delay variation = 20% | | | | |
|---|---|---|---|---|---|---|---|---|---|---|
| Circuit | $\delta$ | $T_{\min}$ | | Int. FF | | $T_{\min}$ | | | Int. FF | |
| | $(ps)$ | $(ps)$ | $(\%)^a$ | # | (%) | $(ps)$ | $(\%)^a$ | $(\%)^b$ | # | (%) |
| 4bit | Ori. | 2840 | (100) | 10 | (100) | 3093 | (110) | (100) | 10 | (100) |
| add | 0 | 1422 | (50) | 0 | (0) | 2047 | (72) | (66) | 0 | (0) |
| | 100 | 1522 | (54) | 0 | (0) | 2147 | (76) | (69) | 0 | (0) |
| | 200 | 1622 | (57) | 0 | (0) | 2247 | (79) | (73) | 0 | (0) |
| | 300 | 1722 | (61) | 0 | (0) | 2347 | (83) | (76) | 0 | (0) |
| | 400 | 1822 | (64) | 0 | (0) | 2447 | (86) | (79) | 0 | (0) |
| | 500 | 1922 | (68) | 0 | (0) | 2547 | (90) | (82) | 0 | (0) |
| 8bit | Ori. | 4474 | (100) | 18 | (100) | 4890 | (110) | (100) | 18 | (100) |
| add | 0 | 1702 | (38) | 0 | (0) | 2629 | (59) | (54) | 0 | (0) |
| | 100 | 1802 | (40) | 0 | (0) | 2729 | (61) | (56) | 0 | (0) |
| | 200 | 1902 | (43) | 0 | (0) | 2829 | (63) | (58) | 0 | (0) |
| | 300 | 2002 | (45) | 0 | (0) | 2929 | (66) | (60) | 0 | (0) |
| | 400 | 2102 | (47) | 0 | (0) | 3029 | (68) | (62) | 0 | (0) |
| | 500 | 2202 | (49) | 0 | (0) | 3129 | (70) | (64) | 0 | (0) |
| 16bit | Ori. | 7634 | (100) | 34 | (100) | 8366 | (110) | (100) | 34 | (100) |
| add | 0 | 3024 | (40) | 0 | (0) | 4533 | (59) | (54) | 0 | (0) |
| | 100 | 3124 | (41) | 0 | (0) | 4633 | (61) | (55) | 0 | (0) |
| | 200 | 3224 | (42) | 0 | (0) | 4733 | (62) | (57) | 0 | (0) |
| | 300 | 3324 | (44) | 0 | (0) | 4833 | (63) | (58) | 0 | (0) |
| | 400 | 3424 | (45) | 0 | (0) | 4933 | (65) | (59) | 0 | (0) |
| | 500 | 3524 | (46) | 0 | (0) | 5033 | (66) | (60) | 0 | (0) |
| 16bit | Ori. | 5075 | (100) | 56 | (100) | 5551 | (110) | (100) | 56 | (100) |
| mul | 0 | 4722 | (93) | 48 | (86) | 3590 | (71) | (65) | 49 | (88) |
| | 100 | 4822 | (95) | 48 | (86) | 3590 | (71) | (65) | 49 | (88) |
| | 200 | 4922 | (97) | 48 | (86) | 3590 | (71) | (65) | 49 | (88) |
| | 300 | 5022 | (99) | 48 | (86) | 3590 | (71) | (65) | 49 | (88) |
| | 400 | 3086 | (61) | 49 | (88) | 3670 | (72) | (66) | 49 | (88) |
| | 500 | 3086 | (61) | 49 | (88) | 3770 | (74) | (68) | 49 | (88) |

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

- $n$-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 I. The ROHM 0.35 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 II shows the results when the algorithm shown in section IV was implemented. Ori. is the original circuit containing the intermediate registers and with the clock timing of all registers fixed at 0 (zero clock-skew). "$\delta$" and "$T_{\min}(ps)$" are the target clock-period range and 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%, i.e. the delay variation for each gate and register was set at $\pm 10\%$.

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.

The relation between the minimum feasible clock period and the number of intermediate registers of the 16 bit multiplier is shown in Fig. 9. 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.

## VI. 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 work 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," in *Proc. IEEE International ASIC/SOC Conference*, 2003, pp. 161–164.

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

[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)*, vol. 1, 1994, pp. 407–410.

[4] A. Takahashi, "Practical fast clock schedule design algorithms," in *Proc. 18th Karuizawa Workshop*, 2005, pp. 515–520.

[5] 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)*, 1997, pp. 37–42.

[6] 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)*, 2004, pp. 333–338.

[7] S.Malik, K.J.Singh, R.K.Brayton, and A.Sangiovanni-Vincentelli, "Performance optimization of pipelined circuits," in *Proc. IEEE/ACM International Conference on Computer Aided Design (ICCAD)*, 1990, pp. 410–413.

[8] C. Wallace, "A suggestion for fast multiplier," *IEEE Trans. on Electronic Computers*, vol. 13, no. 2, pp. 14–17, 1964.