Laboratoire de l'Informatique du Parallélisme

Ecole Normale Supérieure de Lyon
Unité de recherche associée au CNRS n°1398

Retiming DAGs

Pierre-Yves Calland
Anne Mignotte
Olivier Peyran
Yves Robert
Frédéric Vivien

September 1995

Research Report N$^{\mathrm{o}}$ 95-18

Ecole Normale Supérieure de Lyon
46 Allée d'Italie, 69364 Lyon Cedex 07, France
Téléphone : (+33) 72.72.80.00 Télécopieur : (+33) 72.72.80.80
Adresse électronique : lip@lip.ens−lyon.fr

Retiming DAGs

Pierre-Yves Calland
Anne Mignotte
Olivier Peyran
Yves Robert
Frédéric Vivien

September 1995

Abstract

The increasing complexity of digital circuitry makes *global* design optimization no longer possible: a designer will only consider the critical parts of his circuit. This paper discusses timing optimization problems when these critical parts can be represented by Direct Acyclic Graphs (DAGs). We deal with different (though related) clock period problems, under various external constraints. The main algorithm concerns the determination of the optimal clock period of a given circuit. We propose an efficient solution, based on the *retiming* technique, which improves current results in the literature. We give three different formulations depending on which combinational gates delay model is used: same delay for every gate, different delays or non-uniform delays.

Keywords: Direct Acyclic Graph, clock period, optimization, retiming, delay model, registers

Résumé

La complexité grandissante des circuits numériques rend désormais impossible toute optimisation *globale* d'un circuit: les concepteurs ne considèrent plus que les parties critiques de leur circuit. Cet article traite des problèmes d'optimisation temporelle lorsque ces parties critiques peuvent être représentées par des graphes acycliques orientés. Nous considérons différents problèmes concernant les contraintes sur la période d'horloge, qui interviennent lors de la conception d'un circuit, et qui découlent de la détermination de la période optimale du circuit. Nous proposons un algorithme de faible complexité, basé sur la technique de *resynchronisation*, pour résoudre ce problème essentiel, et nous donnons trois formulations différentes, dépendant du modèle de délai utilisé pour les portes combinatoires: même délai pour toutes les portes, délais différents ou délais non-uniformes.

Mots-clés: Graphe acyclique orienté, période d'horloge, optimisation, modèle de délai, registres

Retiming DAGs

P.Y. Calland, A. Mignotte, O. Peyran, Y. Robert and F. Vivien
LIP, CNRS URA 1398, Ecole Normale Supérieure de Lyon
69364 Lyon Cedex 07, France
e-mail: `Pierre-Yves.Calland, Anne.Mignotte, Olivier.Peyran, Yves.Robert, Frederic.Vivien@lip.ens-lyon.fr`

September 1995

Contents

1 Introduction **2**

2 Motivation **2**

3 Review of retiming techniques **4**
3.1 A graph-theoretic framework 4
3.2 Review of related problems 6
3.3 New results 6

4 Clock period minimization for a DAG with unit delay operators **7**
4.1 Circuit period 7
4.2 Two pass Algorithm 8
4.3 Proof and Complexity 10

5 Clock period minimization for a DAG with operators of any delay **12**
5.1 Some bounds on Φ_{opt} 12
5.2 From a register distribution w and a clock period T 13
5.3 Algorithm and complexity 15

6 Clock period minimization for a DAG with operators of non-uniform delay **17**
6.1 Some Definitions 17
6.2 Equivalence with the previous case 18
6.3 Algorithm and complexity 19

7 Conclusion **19**

1 Introduction

Retiming is a technique used to optimize synchronous VLSI circuits. The basic idea is to relocate registers along the paths in the circuit so as to reduce combinational rippling. The rule of the game is that the functional behavior of the circuit as a whole is preserved. There are different cost criteria to evaluate the efficiency of the retiming, but minimizing the clock period and/or the state (total number of registers) are the most frequently used. The survey paper of Leiserson and Saxe [8] gives several polynomial-time algorithms to compute the optimal retiming of a given circuit using the previous two cost criteria.

Pipelining VLSI circuits can be viewed as another instance of the retiming problem. Given a circuit with combinational elements, the problem is to determine the minimum number of registers that should be added, and to determine where to insert these registers, so as to achieve a fixed clock period (when operating in pipelined mode, hence the name *pipelining*). The rule of the game is still that the functional behavior of the circuit is preserved, but the price to pay here is an increase of the latency.

Section 3 is devoted to a review of known results concerning various instances of the retiming problem. We formulate all these instances via a unified graph-theoretic framework as introduced in [8].

The main contribution of the paper is a new algorithm for retiming circuits without cycles, i.e. whose network graph is a DAG (Direct Acyclic Graph). In a word, our algorithm for retiming a DAG only requires two passes over the network graph, as opposed to $|V|$ passes in [8], where $|V|$ is the number of nodes in the network graph (see below in Section 3 for formal definitions).

The rest of the paper is organized as follows: as already stated, Section 3 is devoted to a review of existing literature, together with a formal statement of our main results, whose proofs are given in Section 4 (for operators with the same delay), in Section 5 (for operators with arbitrary delays) and in Section 6 (for operators with non-uniform delays). Section 2 emphasizes upon the practical importance and usefulness of scheduling DAG circuits; a target application is the design optimization of on-line arithmetic operators. We give some final remarks and conclusions in Section 7.

2 Motivation

In on-line arithmetic, the operands circulate through the operators in a digit-serial fashion, most significant digit first. On-line arithmetic was introduced by Ercegovac and Triveli [4], who proposed algorithms for on-line multiplication and division. The operands enter the design at time t and the first digit of the result output at time $t + \delta$, where δ is the latency of the design. The main advantages of on-line arithmetics are the low resulting clock period even for complex applications and the simplified accuracy control. On-line arithmetic performs computation most significant digit first. Each new digit of the result is then a complementary precision. The computation can stop as soon as the required accuracy is reached.

This serialization is possible only if all the operands circulate most significant digit first. Unfortunately, the usual serial algorithms for addition and multiplication work least significant digit first. A specific number system is then used, namely the signed digit redundant number system, whose redundancy makes it possible to perform addition and multiplication with no carry propagation. For instance Figure 1 shows a borrow save adder for redundant arithmetic.

Several algorithms and corresponding architectures have been proposed to perform basic and complex operators such as addition, multiplication, division, square root and trigonometric func-

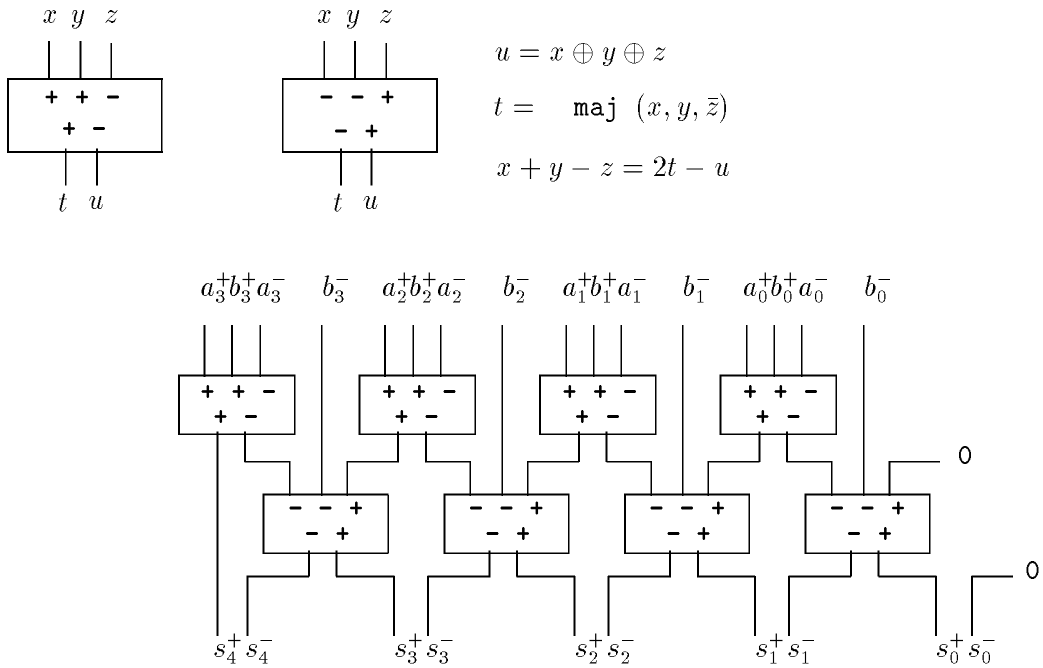

Figure 1: A "Borrow Save" adder, made of PPM (Plus Plus Minus) cells. The digits a_i, b_i and s_i,whose values are -1, 0 or 1, are represented by the bits a_i^+, a_i^-, b_i^+, etc. such that $a_i = a_i^+ - a_i^-$, $b_i = b_i^+ - b_i^-$, ...

tions [2], most significant digit first. These operators are already temporized so as to satisfy a specific behavior. They define a pipelined operator library for on-line arithmetic. For instance, Figure 2 shows a serial borrow save adder. It corresponds to one stage of the adder of Figure 1. Registers are inserted to make sure that c_{n-1}^-, b_{n-1}^- of the previous stage be used to compute s_{n-1}^+ and s_{n-2}^-.

Thus, a given application using on line arithmetic is pipelined by construction. This determines a given latency δ and a given clock period corresponding to the critical path of the design (i.e. the longest path in terms of cells with no register). Let us illustrate this on the application of Figure 3a. The critical path of the application is 6 cells.

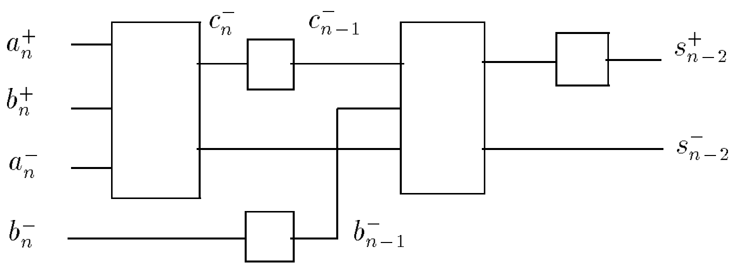

Figure 2: A latency-2 borrow-save serial adder, working most significant digit first. The square cells are registers.

Nevertheless the resulting clock period may not be satisfying for a designer. Therefore, one may have to transform the design by grouping digits to increase the clock period or at the opposite, by inserting timing barriers between operators to reduce the clock period. We will focus on the second transformation as the first one only deals with the pipelined operator library. Suppose that the designer inserts a timing barrier as shown in Figure 3b; the critical path is reduced to 4 cells. Unfortunately, in that case the critical path is not optimized : it can be reduced to 3 cells by moving some registers, as shown in Figure 3c.

The problem of moving registers to optimize the clock period while preserving the whole behavior is the well known *retiming problem*. Moreover, on line arithmetic has two main properties: the design is only made up of two basic cells (PPM and half adders) and there is no loop in the applications (thus, we could sensibly consider that all the cells of the design have the same delay); and the design can be described with a DAG. These two properties will be exploited to reduce the algorithm complexity of retiming as described in the next sections.

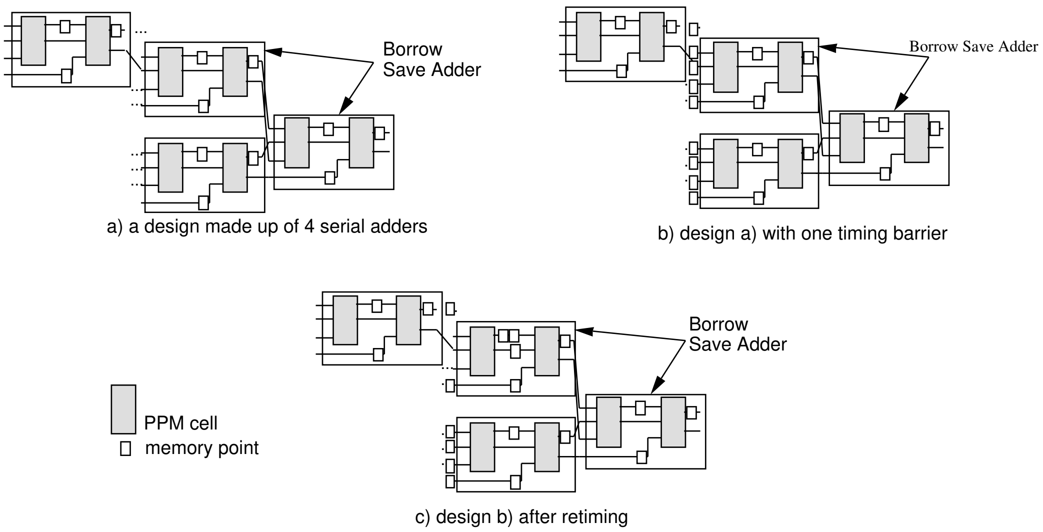

Figure 3: Adding and moving registers to decrease the period.

3 Review of retiming techniques

3.1 A graph-theoretic framework

Computational devices (VLSI circuits or VHDL programs) are represented by a finite, connected, vertex-weighted, edge-weighted directed multi-graph $G = (V, E, d, w)$. Vertices of the graph (or nodes) model the computational elements. Each vertex $v \in V$ is weighted with its nonnegative delay $d(v)$. Vertex delays d can be rational numbers, but since the graph is finite we can always change the time unit to have integer delays. The directed edges E of the graph model interconnections between functional elements. Each edge $e \in E$ is weighted with a "register" count $w(e)$ (the register count corresponds to the number of "wait until clock" statements in VHDL programs). Edge delays are nonnegative integers.

We need a few definitions and notations.

For an edge $e : u \longrightarrow v$, we write $u = t(e)$ the *tail* of e and $v = h(e)$ the *head* of e. We define as *input nodes* (respectively *output nodes*) the nodes whose in-degree (respectively out-degree) is zero.

We add two special vertices v_{from_host} and v_{to_host}, with zero propagation delay, to model the interface of the graph with the external world: $d(v_{from\_host}) = d(v_{to\_host}) = 0$. We let $V' = V \cup \{v_{from\_host}, v_{to\_host}\}$.

Finally, we define a null-weighted edge from v_{from_host} to each input node and a null-weighted edge from each output nodes to v_{to_host}. These edges are called respectively *input edges* and *output edges*. They form the set of *interface edges* that we denote by I. We let $E' = E \cup I$.

We use the same conventions as in [8]. For any simple path $P = v_0 \xrightarrow{e_0} v_1 \xrightarrow{e_1} \dots \xrightarrow{e_{k-1}} v_k$, we define the *path tail* $t(P) = v_0$ as the tail of its first edge, the *path head* $t(P) = v_k$ as the head of its last edge, the *path delay* $d(P) = \sum_{i=0}^{k} d(v_i)$ as the sum of the delay of the vertices of P, and the *path weight* $w(P) = \sum_{i=0}^{k-1} w(e_i)$ as the sum of the weights of the edges of P. We denote by $l(P)$ the length of P, i.e. the number of edges in P. If $k = 0$, we let $l(P) = 0$, $d(P) = d(v_0)$ and $w(P) = 0$.

We assume that G does not contain any zero-weight cycle: in any directed cycle C of G there is some edge with strictly positive weight, i.e. $w(C) > 0$. This condition ensures that the operation of G is well-defined (in the case of VLSI circuits, we say G is synchronous). The clock period of G is then well defined by the equation $\Phi(G) = \max\{d(P); w(P) = 0\}$. Intuitively, the clock period is the maximum amount of propagation delay through which any signal much ripple between clock ticks. The state of G is defined as the sum of the registers over all edges: $S(G) = \sum_{e \in E} w(e)$.

Retiming is an assignment of an integer lag $r(v)$ to each vertex $v \in V$: it amounts to suppress $r(v)$ "registers" to the weight of each edge leaving v (whose tail is v) and to add $r(v)$ registers to each edge entering v (whose head is v). Formally, a retiming function r is a mapping from V' to $\mathbb{Z}$ such that $r(v_{from\_host}) = r(v_{to\_host}) = 0$. It leads to a new edge-weighting function w_r defined for an edge $u \xrightarrow{e} v$ by $w_r(e) = w(e) + r(v) - r(u)$. A retiming is *legal* if the new edge weights w_r are all nonnegative. Obviously, a legal retiming does not change the global behavior of the computational graph G, but both the clock period $\Phi_r(G)$ and the total number of registers $S_r(G) = \sum_{e \in E} w_r(e)$ are altered.

Several problems can be formulated:

Problem 1 Given a graph $G = (V, E, d, w)$ and a maximum allowable clock period c, find a legal retiming r such that $\Phi_r(G) \leq c$.

Problem 2 Given a graph $G = (V, E, d, w)$, find a legal retiming r such that the clock period $\Phi_r(G)$ of the retimed circuit $G_r = (V, E, d, w_r)$ is as small as possible.

Problem 3 Given a graph $G = (V, E, d, w)$ and a maximum allowable clock period c, find the smallest nonnegative integer k such that when adding k registers to each input edge, there exists a legal retiming r such that $\Phi_r(G) \leq c$.

Problem 1 is the basic retiming problem, and can be solved in $O(|V||E|)$ in the most general instance (see [8]). We point out that *Problem 1* is sometimes formulated as a register minimization problem: given c, we ask to determine a legal retiming r such that $\Phi_r(G) \leq c$ and $S_r(G)$ is as small as possible. The complexity of this variant is dominated by the solution of a minimum cost-flow problem, see [8, 10] and the references therein for several bounds.

Problem 2 is a clock minimization problem, and can be solved in $O(|V||E| \log |V|)$ in the most general formulation (see [8]). In fact, it turns out that a way to solve *Problem 2* is to repeatedly solve several instances of *Problem 1* using a dichotomic search for c. A particular case for *Problem 2* supposes that the maximal delay of a node, $D = \max\{d(v), v \in V\}$, grows subpolynomially with respect to the number of functional elements in the circuit, i.e. that there exists a polynomial function P such that $D(G) \leq P(|V|)$. In this case, the algorithm complexity becomes $O(|V||E| \log D)$ (see [9]).

Problem 3 is a circuit scheduling problem that can be solved by solving several instances of *Problem 1*, using a dichotomic search for k (upperly bounded by $|V|$), hence a complexity $O(|V||E|\log^2|V|)$. When the graph G is a DAG given with no registers at all ($S(G)=0$), *Problem 3* is a formulation of the pipelining problem. In this particular case, *Problem 3* has a complexity reduced to $O(|E|\log D)$ (see [9]).

3.2 Review of related problems

Retiming techniques have been applied to several related problems. Retiming was first introduced by Leiserson and Saxe, in [7], in order to solve *Problem 1* for systolic circuits (unit-delay circuits with at least one register between two functional elements). Their solution, using the Bellman-Ford algorithm, requires $O(|E||V|)$ time.

Wehn *et al.* linked high level synthesis and retiming in [3], using the correspondence between "wait until" statements and registers. Using the conditions defined by Leiserson and Saxe, they stated different problems, like As-Soon-As-Possible scheduling or minimization of registers, as linear programming problems. They expressed these problems with various objective functions. However, they did not provide any solution that efficiently solves them.

Some researchers proposed interesting results for simpler problems, like C. Papaeftymiou in [9], who proposed an algorithm which gives the minimum clock period of a circuit with at most l levels of registers, where l is a given value (this problem is the dual of *Problem 3*). The algorithm runs in $O(|E|log|V|)$, but the initial circuit has to be empty of registers. Therefore it is only interesting when you need to pipeline a combinational circuit.

In the same idea of optimizing a combinational circuit, Munzer and Hemme proposed in [5] an algorithm in $O(|E|.|V|^2)$ to solve the register minimization problem. From a register-empty circuit, they apply an As-Soon-As-Possible and As-Last-As-Possible register locations, with the constraint of satisfying a given clock period. The two locations determine the parts of the circuit where registers are likely to be moved. Within these parts, they use a maximal flow algorithm to find the minimal number of registers.

Considering sequential circuits, Wei-Jeng Cheng et al. showed the relationship between retiming and loop folding ([1]). They propose a solution to the *Problem 2* using an As-Soon-As-Possible pipeline algorithm. However, they did not really explain the way they moved the registers within the circuit, which is the most important step in terms of complexity.

Finally, Leiserson and Saxe improved their technique ([8]) and proposed several algorithms to solve *Problem 2* by capturing it into a linear-programming problem, the best one running in $O(|E||V|log|V|)$. They also have a solution to the register minimization problem in the case of sequential circuits. They express this problem as a minimum cost flow problem, after having augmented the graph representing the circuit with virtual vertices and edges, in order to have a good function of cost. The complexity is $O(|V|^3 log|V|)$.

3.3 New results

All these problems can be particularized to the case where G is a DAG (with a nonzero state for *Problem 3*). Our main results show that the complexity of the three problems can be significantly decreased for DAGs, as stated in the following theorems.

Theorem 1 *Given a DAG $G=(V,E,d,w)$ and a maximum allowable clock period c, a legal retiming r such that $\Phi_r(G)\le c$ can be found in $O(|E|)$, if it exists.*

Theorem 2 *Given a DAG $G = (V, E, d, w)$, let $D = \max_{v \in V} d(v)$. A legal retiming r such that the clock period $\Phi_r(G)$ of the retimed circuit $G_r = (V, E, d, w_r)$ is as small as possible can be found in $O(|E|(\log|V| + \log D))$.*

Theorem 3 *Given a DAG $G = (V, E, d, w)$ and a maximum allowable clock period c, let $D = \max_{v \in V} d(v)$. Let k be the smallest nonnegative integer k such that when adding k registers to each input edge, there exists a legal retiming r such that $\Phi_r(G) \leq c$. Then k (and the corresponding retimed circuit) can be found in $O(|E|)$.*

We prove these theorems in the following sections.

4 Clock period minimization for a DAG with unit delay operators

In this section we present an algorithm which minimizes the period for a DAG whose operators have all the same delay. Without loss of generality we let $d(v) = 1$ for all $v \in V \setminus \{v_{from\_host}, v_{to\_host}\}$. In subsection 4.1 we propose a lower bound on the period. In Subsection 4.2 we describe an algorithm which achieves this lower bound. In subsection 4.3 we prove the correctness of our algorithm and we give its complexity.

4.1 Circuit period

Period value

Lemma 1 *Let $G = (V, E, d, w)$ be a DAG. The minimum clock period $\Phi_{opt}(G)$ of G satisfies to:*

$$\Phi_{opt}(G) \geq \max\{d(v) : v \in V\} \tag{1}$$

$$\Phi_{opt}(G) \geq \max\left\{\frac{d(P)}{w(P)+1} : P \text{ path of } G,\ t(P) = v_{from\_host},\ \text{and } h(P) = v_{to\_host}\right\} \tag{2}$$

Proof of lemma 1: The first bound is true by definition: a path of P of length 0 satisfies $w(P) = 0$. Let us prove the second bound.

Suppose that G has been retimed with respect to the period $\Phi_{opt}(G)$. Let P be a path of G, from v_{from_host} to v_{to_host}. $d(P)$ is equal to the number of vertices in P (except the two extra vertices v_{from_host} and v_{to_host}). Therefore, P contains $l(P) = d(P) + 1$ edges and $w(P)$ registers. Furthermore, between two consecutive registers of P there are at most $\Phi_{opt}(G)$ nodes, i.e. $\Phi_{opt}(G) - 1$ arcs without registers. For the same reason, between v_{from_host} (resp. v_{to_host}) and the first (resp. last) register, there are at most $\Phi_{opt}(G)$ arcs without registers, counting the input (resp. output) edge. Now, let us count the number of edges in P (i.e. the length $l(P)$ of P), counting separately edges with registers and edges between registers. This leads to:

$$l(P) \leq \Phi_{opt}(G) + w(P) + (w(P) - 1) * (\Phi_{opt}(G) - 1) + \Phi_{opt}(G)$$

$$\text{i.e. } l(P) \leq \Phi_{opt}(G) * (w(P) + 1) + 1$$

$$\text{i.e. } \Phi_{opt}(G) \geq \frac{l(P) - 1}{w(P) + 1}$$

This proves the lemma since $l(P)$ and $w(P)$ are unchanged by retiming. ■

In Subsection 4.2 we present an algorithm which achieves this lower bound, thereby proving the following theorem:

Theorem 4 *Let G be a DAG. The minimal period $\Phi_{opt}(G)$ of G is:*

$$\Phi_{opt}(G) = \max\left\{\max_{v\in V} d(v), \max\left\{\frac{d(P)}{w(P)+1} : P \text{ path of } G,\ t(P) = v_{from\_host},\ h(P) = v_{to\_host}\right\}\right\} \tag{3}$$

Period computation

Let us build a graph G', identical to G except that an edge $e_{round} : v_{to\_host} \rightarrow v_{from\_host}$ is added. This edge carries a single register: $w(e_{round}) = 1$.

Lemma 2 *The value of the minimum cycle mean of G' is the inverse of the clock period of the circuit G, if $\Phi_{opt}(G) > \max_{v\in V} d(v)$.*

Proof of lemma 2: The minimum cycle mean of a directed graph is always reached on an elementary cycle.

Let C be an elementary cycle of G'. C can be decomposed as e_{round},e_1, ..., e_k. We consider the path P of G equal to e_1, ..., e_k. By construction, $w(C) = w(P) + 1$ and $d(C) = d(P)$. The mean value on C is then equal to $\frac{w(P)+1}{d(P)}$.

Thus, the cycle mean of an elementary cycle of G' is minimal if it is the inverse of the period of a path of G from an input node to an output one. ■

The period of a graph G can thus be computed in $O(|V(G')| * |E(G')| \log |V(G')|)$, i.e. $O(|V(G)| * |E(G)| \log |V(G)|)$, by a dichotomic use of the Bellman-Ford algorithm, as the computation of $\max_{v\in V} d(v)$ can be done in $O(|V(G)|)$. Note that we can also use Karp's mean weight cycle algorithm (see [6]) on a graph G'' built from G: in G'', we collapse the input nodes, we collapse the output nodes, we link these new nodes with an edge with a single register and we cut out the two special nodes. The complexity of this second solution is $O(|V(G)| * |E(G)|)$.

We will see in Subsection 4.3 that this pre-computation of the period can be more efficiently replaced by a dichotomic search of the optimal period with the help of our "two pass algorithm".

4.2 Two pass Algorithm

The following algorithm operates through two passes over G to determine a legal retiming r such that $\Phi_r(G) \leq T$. In the first pass, all registers are moved as close as possible to the node v_{from_host}, with respect to retiming rules: when processing a node, we can add (suppress) the same number of registers on each incoming (outcoming) edge, provided that the weight on each edge remains nonnegative. In the second pass, all registers are moved as far as possible from the node v_{from_host}, with respect to the same rules, and also with respect to the target period T.

Algorithm

Remember that V is the set of vertices of G with non zero in and out-degrees and that the set of all vertices of G is $V' = V \cup \{v_{from\_host}, v_{to\_host}\}$.

Consistency test
    If $T < \max_{v\in V} d(v)$ **then ERROR endif**
First pass
    for (each vertex $v \in V$ in reverse topological order)

do
$n = \min_{e \in E | t(e)=v} w(e)$
$\forall e \in E \mid t(e) = v, w(e) \leftarrow w(e) - n$
$\forall e \in E \mid h(e) = v, w(e) \leftarrow w(e) + n$
enddo

Second pass
$\Delta(v_{from\_host}) \leftarrow 0$
for (each vertex $v \in V$ in topological order)
do
$n \leftarrow \min_{e \in E | h(e)=v} w(e)$
$\Delta(v) \leftarrow d(v) + \max\{\Delta(t(e)) : e \in E,\ h(e) = v \text{ and } w(e) = n\}$
* Δ *computation* statement *\
if $(\Delta(v) > T)$
* *Leave a register* case *\
then
if (n = 0) **then ERROR endif**
$n \leftarrow n - 1$
$\Delta(v) \leftarrow d(v)$
endif
$\forall e \in E \mid h(e) = v, w(e) \leftarrow w(e) - n$
$\forall e \in E \mid t(e) = v, w(e) \leftarrow w(e) + n$
enddo

Lemma 3 *Let $v \in V$ be a node of G. After the second pass of Algorithm 4.2 (if the* ***ERROR*** *case does not occur), $\Delta(v)$ is the largest delay of a path without register that ends at v:*

$$\Delta(v) = \max\{d(P) : P \text{ path of } G, w(P) = 0, h(P) = v\} \tag{4}$$

Proof of lemma 3: The proof is by induction on the length l of the longest path without register that ends at v.

If $l = 0$, there is at least one register on each of the incoming edges of v, i.e. the *leave a register* case occurred during the process of v. Thus $\Delta(v) = d(v)$.

Now, suppose that the property has been proved for any value of l between 0 and l'.

Let v be a node of G such that $l = l' + 1$. $l \geq 1$, then the *leave a register* case did not occurred and the value $\Delta(v)$ was set by the Δ *computation* statement. Thus:

$$\Delta(v) = d(v) + \max\{\Delta(t(e)) : e \in E,\ h(e) = v \text{ and } w(e) = 0\}$$

Then, because of the induction hypothesis,

$$\Delta(v) = d(v) + \max\{\max\{d(P) : P \text{ path of } G, w(P) = 0, h(P) = t(e)\} : e \in E,\ h(e) = v,\ w(e) = 0\}$$

which is equivalent to equation 4. ■

We illustrate the execution of the "two-pass algorithm" on an example. The original graph to be retimed is given in Figure 4, and as a period of 6. The graph obtained after the first pass is shown in Figure 5. Finally, the graph obtained after the second pass is shown in Figure 6.

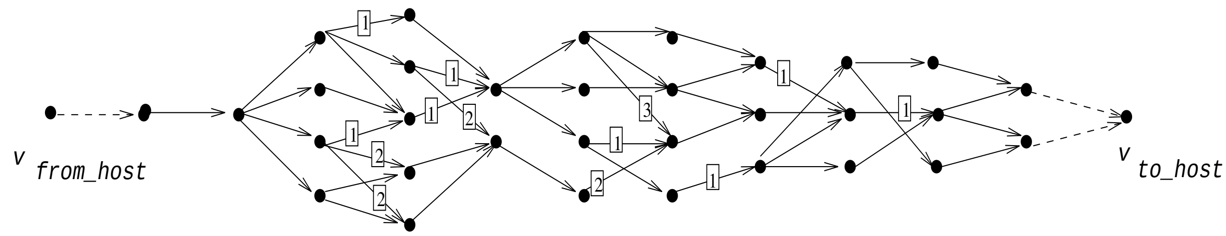

Figure 4: Initial register distribution with a period equal to 6.

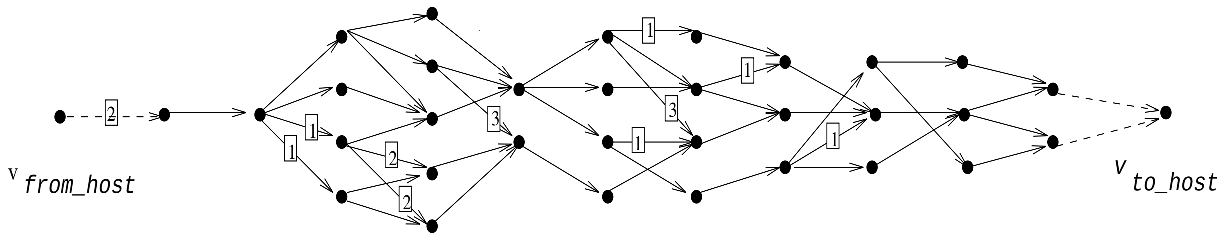

Figure 5: Resulting distribution after the first pass of the algorithm, when operators have same unit delay.

4.3 Proof and Complexity

Proof of the correctness of the algorithm

Theorem 5 *Let* $M = \max\left\{\max_{v \in V} d(v), \max\left\{\frac{d(P)}{w(P)+1} : P \text{ path of } G,\ t(P) = v_{from\_host},\ h(P) = v_{to\_host}\right\}\right\}$. *If the tested period* T *is greater than or equal to* M *then the "two-pass algorithm" on* G *succeeds to find a new register distribution with period less than or equal to* T.

Before establishing the proof of this theorem, we demonstrate two lemmas.

Lemma 4 *Let* v *be a node of* G, $v \neq v_{from\_host}$. *We suppose* $T \geq \max_{v \in V} d(v)$. *Then, after the first pass of algorithm 4.2, there is a path without any register from* v *to* v_{to_host}.

Proof of lemma 4: Each node $v \in V$ is processed by the first pass of algorithm 4.2. Registers are moved so that at least one of its outcoming edges has no register. To build recursively the path announced in the lemma, choose one of these edges as the first edge of the path and do the same operation on the edge head. ■

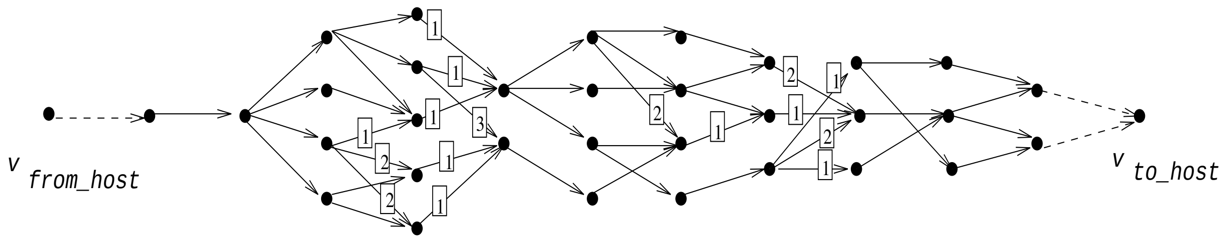

Figure 6: Resulting distribution after the second pass of the algorithm, when operators have same unit delay. $T = \Phi_{opt}(G) = 4$.

Lemma 5 *Let $v \in V$. We suppose that no* ***ERROR*** *occurred before or during the process of v by the second pass of algorithm 4.2. Then after this second pass, there is a path from an input node to v which begins by repeating several times the pattern $T-1$ edges without any register followed by an edge with a single register, plus $\Delta(v)-1$ edges with no register.*

Proof of lemma 5: We prove the property by induction, considering all $v \in V$ in the same order as processed by the second pass.

Obviously, this is true for any input node v: a void pattern plus $\Delta(v)-1 = 1-1 = 0$ edge.

Now, let $v \in V$ be a vertex of G which is not an input node and suppose that lemma 5 holds for all vertices processed before v. Consider the situation during the process of v and just after the Δ *computation* statement:

$n \leftarrow \min_{e \in E | h(e)=v} w(e)$
$\Delta(v) \leftarrow d(v) + \max\{\Delta(t(e)) : e \in E,\ h(e) = v \text{ and } w(e) = n\}$

Let e_1 be an edge such that, at this step of the algorithm, $w(e_1) = n$, $h(e_1) = v$, and $\Delta(v) = d(v) + \Delta(t(e_1))$. Let $v_1 = t(e_1)$. Since v_1 has already been processed, by induction hypothesis, there is a path P_1 from an input node to v_1 consisting of $T-1$ edges with no register followed by an edge with a single register (this pattern being repeated say r times - actually $r = w(P_1)$-), plus $\Delta(v_1)-1$ edges with no register.

We have now to consider two cases, depending whether the *leave a register* case occurs or not, i.e. whether $\Delta(v) > T$ or not.

Case $\Delta(v) \le T$: In this case, n and $\Delta(v)$ are not modified and n registers are moved to the outcoming edges of v. Therefore the weight of e_1 is modified to 0. Now, by adding e_1 to P_1, we get a path from an input node to v with the desired pattern repeated r times, followed by $\Delta(v_1)+1-1 = \Delta(v)-1$ edges with no register.

Case $\Delta(v) > T$ (*Leave a register* case): Since $\Delta(v) > T$, $\Delta(v_1) = T$. n is changed to $n-1$ and $\Delta(v)$ to $d(v) = 1$. One register is left on edge e_1. Therefore, by adding e_1 to P_1, we get a path from an input node to v with the desired pattern repeated r times, plus $\Delta(v_1)-1 = T-1$ edges without registers, plus one edge with a single register, i.e. $r+1$ times the desired pattern, followed by $\Delta(v)-1 = 0$ edge.

■

Proof of theorem 5: To prove theorem 5 we must prove first that the ERROR case cannot occur. Then we show that the register distribution satisfies the period T.

As T is greater than or equal to M, T is greater than $\max_{v \in V} d(v)$.

The first pass always ends correctly. Let us suppose that the second pass does not end correctly, i.e. that the ERROR case occurs during the treatment of a node v. This means that there exists a node v_1 such that $\Delta(v_1) = T$ and an edge e from v_1 to v which carries no register. v_1 was treated by the second pass, we conclude from lemma 5 that there exists a path P_1 from an input node to v_1 which is composed by $w(P_1)$ times the motif $T-1$ edges without any register followed by an edge with a single register, plus $T-1$ edges without any register.

The vertex v was treated by the first pass but not by the second pass, thus the result of lemma 4 is still valid (none of the successors of v have been touched since the first pass). Thus there exists a path P_2 from v to v_{to_host} which carries no register. $l(P_2) \ge 1$ since $v \ne v_{to\_host}$.

Now, we build a path P from v_{from_host} to v_{to_host} as the concatenation of an input edge, P_1, e, and P_2. $w(P) = w(P_1)$ and $l(P) = 1 + l(P_1) + 1 + l(P_2) = 1 + (w(P_1) * T + T - 1) + 1 + l(P_2)$. Now, replacing $w(P_1)$ by $w(P)$, and $l(P)$ by $d(P) + 1$ gives

$$T \leq \frac{d(P) - 1}{w(P) + 1}$$

which contradicts the fact that:

$$T \geq M \geq \frac{d(P)}{w(P) + 1} \text{ (see lemma 1).}$$

We have thus proved that the second pass ends without the **ERROR** case. We just have to check that the period of G is now smaller than T.

For all $v \in V$, either the *leave a register* case does not occur (and $\Delta(v) \leq T$) or $\Delta(v) = d(v) \leq T$. Furthermore, by lemma 3, $\Delta(v)$ is the largest delay of a path with no register that ends at v. Therefore, the period of G is one of the $\Delta(v)$ and is smaller or equal to T. ■

Corollary 1 *Theorem 4.*

Proof: Theorem 5 gives an upper bound on Φ_{opt} which is exactly the lower bound of lemma 1. ■

Complexity

The topological sort on G can be done in $O(|E| + |V|) = O(|E|)$ (recall that G is connected). The first and second passes of the algorithm process all nodes of G once. For each node, all its input and output edges are processed. The complexity of these passes is therefore $O(|E|)$. The complexity of the two-pass algorithm alone is then $O(|E|)$.

If we try to determine the minimal possible clock, we can use a dichotomic search. Indeed, as each node v has unit delay $d(v) = 1$, $\Phi_{opt}(G)$ is upperly bounded by $|V|$. Thus we can use the "two pass algorithm" for a dichotomic search of $\Phi_{opt}(G)$ in the range $[1, |V|]$: for each tested value we try to process G by the "two-pass algorithm", if the process succeeds then $\Phi_{opt}(G)$ is smaller than or equal to the tested value, else $\Phi_{opt}(G)$ is strictly greater. The complexity of this dichotomic search is therefore $O(|E| \log |V|)$ as $\Phi_{opt}(G)$ is upperly bounded by $|V|$. Thus, pre-computing $\Phi_{opt}(G)$ is less efficient than this strategy.

5 Clock period minimization for a DAG with operators of any delay

Let $G = (V, E, d, w)$ be a DAG. In this section the function $d : V \rightarrow \mathbb{N}$ is no longer constant. However, we still consider that $d(v_{to\_host}) = d(v_{from\_host}) = 0$ in all cases.

5.1 Some bounds on Φ_{opt}

Let $D = \max_{v \in V} d(v)$ be the largest operator delay and $\min_{v \in V} d(v)$ the smallest one. Let G' be a graph obtained by replacing d with $d' : v \in V \mapsto \min_{v \in V} d(v)$ and compute the minimum clock period $\Phi(G')$ (as in 4) for this graph. Similarly, let G'' be a graph obtained by replacing d with $d'' : v \in V \mapsto \max_{v \in V} d(v)$ and let $\Phi(G'')$ be its period. So we have:

$$\Phi(G') \leq \Phi_{opt}(G) \leq \Phi(G'')$$

Furthermore we have the obvious following bounds:

$$\max_{v \in V} d(v) \leq \Phi_{opt}(G) \leq \Phi(G)$$

Then:

$$\max(\Phi(G'), \max_{v \in V} d(v)) \leq \Phi_{opt}(G) \leq \min(\Phi(G''), \Phi(G))$$

5.2 From a register distribution w and a clock period T

Theorem 6 *Let $G = (V, E, d, w)$ be a DAG, and let T_{test} be an integer. If there exists a register distribution w_r on G whose clock period is less than or equal to T_{test} then the "two-pass algorithm" called with T set to T_{test} succeeds to find such a register distribution.*

Remark: If there exists a register distribution w_r on G whose clock period is less than or equal to T_{test}, then T_{test} is certainly greater than $\max_{v \in V} d(v)$. This property will implicitly be used in the rest of this section.

To establish the proof of this theorem, we need to prove first the two lemmas below.

Lemma 6 *Let $G = (V, E, d, w)$ be a line graph, $G : v_0 \xrightarrow{e_0} \ldots \xrightarrow{e_{i-1}} v_i \xrightarrow{e_i} \ldots \xrightarrow{e_{k-1}} v_k$. If $G = (V, E, d, w)$ admits at least one valid register distribution whose clock period is less than or equal to T_{test}, then the "two-pass algorithm" succeeds to find one.*

Proof: Let w' be any register distribution on G whose clock period is less than or equal to T_{test}. When applying the "two-pass algorithm" on (V, E, d, w') with the period T set to T_{test}, we obtain a new register distribution w''. As G is a line graph, a retiming does not change the total amount, n, of registers of G. Let $i_1 \leq \ldots \leq i_n$ (resp. $j_1 \leq \ldots \leq j_n$) be the indices of the edges of $G = (V, E, d, w')$ (resp. $G = (V, E, d, w'')$) which carry the n registers.

We prove by induction on l that $j_l \geq i_l$, for all $l \in \{1, \ldots, n\}$.

This is true for $l = 1$: indeed, i_1 (resp. j_1 is the position of the register which is the closest to v_0 in $G = (V, E, d, w')$ (resp. $G = (V, E, d, w'')$). By definition of the "two-pass algorithm", this register is as far as possible from v_0 in $G = (V, E, d, w'')$. Therefore, $j_1 \geq i_1$. Actually, j_1 is defined by:

$$\sum_{m=0}^{j_1} d(v_m) \leq T_{test} < \sum_{m=0}^{j_1} d(v_m) + d(v_{j_l+1})$$

Now, we suppose that $j_0 \geq i_0, \ldots, j_l \geq i_l$ and we prove that $j_{l+1} \geq i_{l+1}$. By definition of the "two-pass algorithm", j_{l+1} indicates at which vertex the *leave a register* case occurred for the $(l+1)$-th register. We thus have:

$$\sum_{m=1+j_l}^{j_{l+1}} d(v_m) \leq T_{test} < d(v_{1+j_{l+1}}) + \sum_{m=1+j_l}^{j_{l+1}} d(v_m)$$

To show that $j_{l+1} \geq i_{l+1}$, we just have to prove that $\sum_{m=1+j_l}^{i_{l+1}} d(v_m) \leq T_{test}$. But: $\sum_{m=1+j_l}^{i_{l+1}} d(v_m) \leq \sum_{m=1+i_l}^{i_{l+1}} d(v_m)$ (since $j_l \geq i_l$ by induction hypothesis) and $\sum_{m=1+i_l}^{i_{l+1}} d(v_m) \leq T_{test}$ as $G = (V, E, d, w')$ has a register of clock period smaller than or equal to T_{test} by hypothesis.

Now, it remains to show that after e_{j_n}, the **ERROR** case can not occur. Indeed, if the process of $G = (V, E, d, w)$ by the "two-pass algorithm" fails, we have $\sum_{m=j_n}^{k} d(v_m) > T_{test}$, but since $j_n \geq i_n$, we also have $\sum_{m=i_n}^{k} d(v_m) > T_{test}$ and the register distribution w' does not have a clock period smaller than or equal to T_{test}, which contradicts the hypothesis. Thus, the processing of $G = (V, E, d, w)$ by the "two-pass algorithm" succeeds. ■

Lemma 7 *Let $G = (V, E, d, w)$ be a DAG, and let T_{test} be an integer. Process $G = (V, E, d, w)$ by the "two-pass algorithm" with the period T set to T_{test}. Then, for any node v of V which has been successfully processed by the second pass of the algorithm, there exists a path P starting from v_{from_host}, ending at v and satisfying the following properties:*

- *The weight of every edge of P is equal to zero or one.*
- *if r_1 is a register of P, in P alone considered as a line graph, this register cannot be pushed further from the input nodes without violating the period (value) T_{test}.*

Proof of lemma 7: P is built from the head to the tail by induction: if v is the current tail of P and if v is not an input vertex, we choose the predecessor u of v and the edge e from u to v such that:

$$w(e) = \min_{f \in E}\{w(f) : \exists\ x \xrightarrow{f} v\} \quad \text{and} \quad \Delta(u) = \max_{x \in V}\{\Delta(x) : \exists\ x \xrightarrow{f} v, w(f) = w(e)\} \tag{5}$$

Then $w(e)$ is equal to 0 or 1, as the number of registers removed from the incoming edges of a vertex is equal to the minimal number of registers on these edges or is equal to this minimal number of register minus 1.

We prove now that on P considered as a line graph a register cannot be moved further from the input node.

Let r be a register of P, and let e be the edge that carries r. Let respectively u and v be the tail and head of e: $u \xrightarrow{e} v$. $w(e) = 1$ thus, by definition of e (see equation 5), every incoming edge of v carries a register. Thus, as by hypothesis the node v has been successfully processed by the second pass of the algorithm, there was during the process of v at least one predecessor u' of v linked to v by an edge e' such that $w(e')$ was minimal among the input edges of v and such that $\Delta(u') \geq T_{test} - d(v) + 1$ (conditions of occurrence of the *leave a register* case in algorithm 4.2). Then, by definition of u (see equation 5), $\Delta(u) \geq \Delta(u') \geq T_{test} - d(v) + 1$ and the register r cannot be pushed further from the input node of P without violating the period T_{test}. ■

Remark: The result of lemma 7 can be applied to any node of G which had successfully been processed by the second pass even if the whole processing of G failed.

Proof of theorem 6: If the "two-pass algorithm" ends correctly, i.e. if the ERROR case does not occur, then the register distribution found satisfies the period T_{test}: for each node v of G, $\Delta(v) \leq T = T_{test}$ and then lemma 3 allows us to conclude that the register distribution satisfies T_{test}.

We suppose now that the ERROR case occurred during the process of G. As there exists a register distribution valid for T_{test}, T_{test} is greater than or equal to $\max_{v \in V} d(v)$. Thus, the ERROR case occurred during the process of a node v: there is in G a node u and an edge e such that:

$$u \xrightarrow{e} v \text{ and } w(e) = 0 \text{ and } \Delta(u) + d(v) > T_{test}$$

We build for u the path P_u defined in lemma 7.

There is a path P' with no register from v to v_{to_host} as stated in lemma 4. Let P be the concatenation of P_u, e, and P'. By construction $w(P)$ is equal to $w(P_u)$. By definition of P_u (see lemma 7) no register of the line graph P_u can be moved further from v_{from_host}, and there is no register on $e + P'$. Thus, the "two-pass algorithm" is not able to find a register distribution valid for T_{test} on the line graph P. This means that such a register distribution does not exist (see lemma 6).

However, this contradicts the fact that there exists a retiming r on the whole graph G with clock period less than or equal to T_{test} (theorem hypothesis). Indeed, the restriction of r to the vertices of P (path from v_{from_host} to v_{to_host}) gives a distribution of the $w(P)$ registers of P considered as a line graph, with clock period less than or equal to the clock period of G. ■

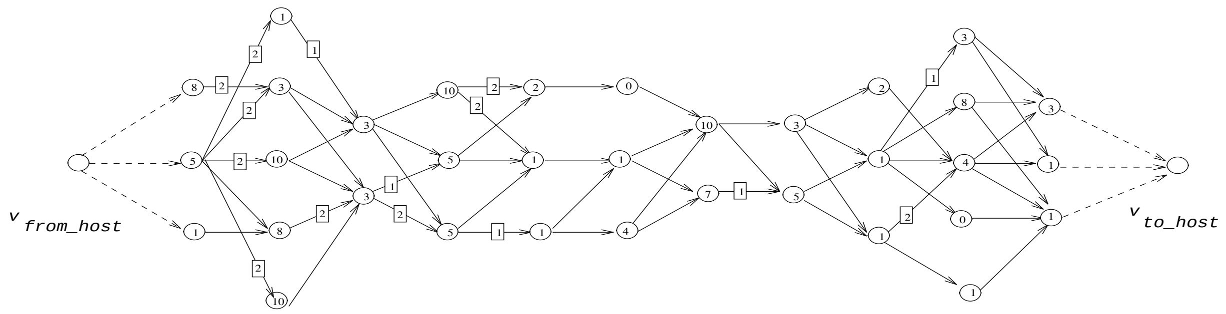

Figure 7: Initial register distribution.

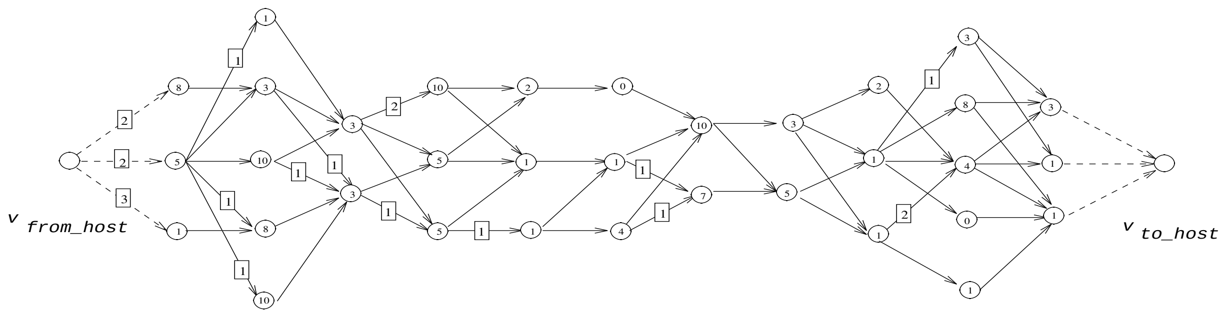

Figure 8: Resulting distribution after the first pass of the algorithm. Operators have different delays.

5.3 Algorithm and complexity

Algorithm

The algorithm below finds the minimum achievable period for a DAG and a valid retiming for this period.

This algorithm uses the "two-pass algorithm" as a kernel in a dichotomic search of the minimum clock period: if there exists a valid retiming for the tested period value c then the "two-pass algorithm" succeeds to find one, else it fails, and in both cases the set of possible period values is refined. G_r is the resulting graph ($G_r = (V, E, d, w_r)$).

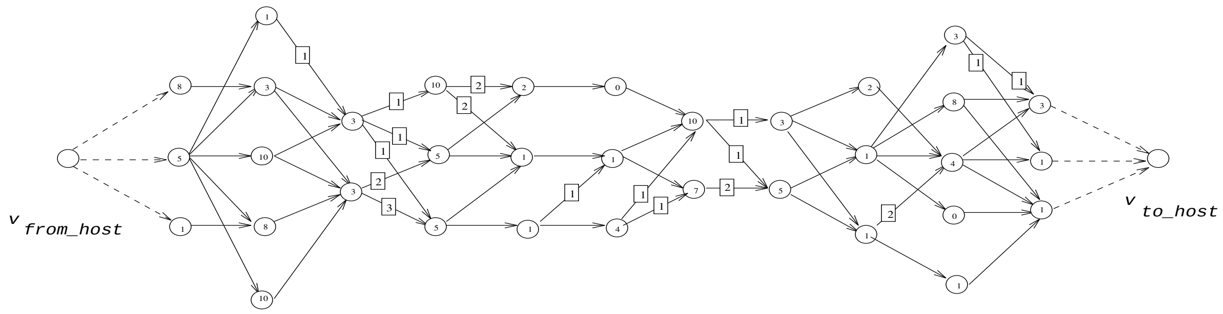

Figure 9: Resulting distribution after the second pass of the algorithm. Operators have different delays.

General algorithm

$t_{min} \leftarrow \max_{v \in V} d(v)$
$t_{max} \leftarrow \Phi(G)$
Repeat
  $t \leftarrow \lfloor \frac{t_{max} + t_{min}}{2} \rfloor$
  if the "two-pass algorithm" succeeds with $T = t$
    then $t_{max} \leftarrow \Phi(G_r)$
    else $t_{min} \leftarrow t + 1$
      $t_{max} \leftarrow \max(t_{max}, \Phi(G_r))$
    endif
Until $t_{max} = t_{min}$

We illustrate the execution of the algorithm on an example. The original graph to be retimed is given in Figure 7. We assume that the optimal period $\Phi_{opt}(G) = 18$ has been found and we apply the two passes. The graph obtained after the first pass is shown in Figure 8. Finally, Figure 9 represents the graph obtained after the second pass of our algorithm.

Complexity

The complexity of the "two-pass algorithm" is $O(|E|)$, thus the complexity of this algorithm is $O(|E|(\log|V| + \log D))$, where $D = \max_{v \in V} d(v)$): indeed, $\Phi_{opt}(G)$ is upperly bounded by $|V|D$.

The stated complexities of the "two-pass algorithm" and of the "general algorithm" directly prove Theorem 1 and Theorem 2. As for Theorem 3, we use a dichotomic search for k, using the "two-pass algorithm" once for each instantiation. The value of k is clearly upperly bounded by the diameter of G, hence by $|V|$ (note that we could slightly refine the given complexity using the diameter). This would lead to a complexity of $O(|E|\log|V|)$ for *Problem 3*. However, a much better solution is to add $|V|$ registers to each input edge, and run the "two-pass algorithm" once. At the end of the second pass, simply suppress the useless registers left on the output edges. Let k' be the minimum number of useless registers on the output edges. Then $k = |V| - k'$ is the desired value and the process we described gives the retimed circuit. Then, the complexity of the algorithm is reduced to $O(|E|)$.

6 Clock period minimization for a DAG with operators of non-uniform delay

During the conception of a circuit, the designer knows the different types of operators that he will use, independently of the technology on which the design will be implemented. Symbolic operator delays are available, which could be seen as worst case delays.

Once the design technology has been chosen, more accurate delays are available. An operator delay is no longer symbolic: in fact, there is a delay for each internal path (or internal edge) between an input and an output. Usually, the delay of an internal edge e_i is equal to

$$d(e_i) = \alpha_{e_i} + \beta_{e_i} * f_{e_i}$$

where α_{e_i} is the intrinsic delay, β_{e_i} is the extrinsic delay (depending on the output capacitance) and f_{e_i} is the output fanout.

This section aims at giving a new form of the two-pass algorithm taking into account this more accurate delay model.

6.1 Some Definitions

Let $G = (V, E, d, w)$ be a DAG. In this section we consider vertices in which the delays through individual elements are non-uniform. Let $I(v)$ and $O(v)$ be the inputs and outputs of v. For each pair $(i, o) \in I(v) \times O(v)$ of vertex v, we call $d_v(i, o)$ the delay from input i to output o of vertex v. We shall now consider the tail $t(e)$ of edge e as the pair (v, o), e coming out of node v from output o. In the same way, the head becomes the pair (i, v), where i is an input of node v. As in the previous sections, we add two new vertices v_{from_host} and v_{to_host} with a single output (resp. input). In the two-pass algorithm, we will need to compute a new value called Δ_{out}. The two-pass algorithm becomes:

Algorithm

Consistency test
  If $T < \max_{v \in V,\ (i,o) \in I(v) \times O(v)} d_v(i, o)$ **then ERROR endif**
First pass
  for (each vertex $v \in V$ in reverse topological sort order)
    do
    $n = \min_{e \in E | t(e) = v} w(e)$
    $\forall e \in E \mid t(e) = v, w(e) \leftarrow w(e) - n$
    $\forall e \in E \mid h(e) = v, w(e) \leftarrow w(e) + n$
    enddo

Second pass
  $\Delta_{out}(v_{from\_host}) \leftarrow 0$
  for (each vertex $v \in V$ in topological sort order)
    do
    $n \leftarrow \min_{e \in E | h(e) = v} w(e)$
    $\Delta_{out}(v, o) \leftarrow \max_{i \in I(v)}(d_v(i, o) + \Delta_{out}(t(e)) : e \in E, h(e) = (i, v), w(e) = n)$
    $\Delta(v) = \max_{s \in O(v)} \Delta_{out}(v, s)$
    if $(\Delta(v) > T)$

then
if (n = 0) **then ERROR endif**
$n \leftarrow n - 1$
$\forall o \in O(v), \Delta_{out}(v, o) = \max_{i \in I(v)} d_v(i, o)$
$\Delta(v) = \max_{o \in O(v)} \Delta_{out}(v, o)$
endif
$\forall e \in E \mid t(e) = v, w(e) \leftarrow w(e) - n$
$\forall e \in E \mid h(e) = v, w(e) \leftarrow w(e) + n$
enddo

We denote by o^v_m and i^v_m one of the output and input for which $\Delta(v)$ is reached ($\Delta(v) = d_v(i^v_m, o^v_m) + \Delta_{out}(t(e))$ where $e \in E, h(e) = (i^v_m, v), w(e) = n$).

6.2 Equivalence with the previous case

We will prove that theorem 6, Lemma 6 and Lemma 7 still apply to our new graph model and to our new two pass algorithm.

Definition of $nwp(v)$**:** $\Delta_{out}(v, o)$ can be considered as the arrival time of a signal s on the output o of vertex v. That is, the longest time it will take to the signal coming out from a register to *arrive* on the output o of vertex v. It means that there is a null weight path, that we will call $nwp(v)$, ending at v and of delay $\Delta_{out}(v, o)$.

Definition of $pr(v)$**:** If every incoming edge of v has at least one register, it means that the first computation of $\Delta(v)$, during the second pass, was greater than T (in such a case, $nwp(v)$ has only one vertex - v - since a register has been left):

$$\exists o^v_m \in O(v) : \Delta(v) = \Delta_{out}(v, o^v_m) > T$$

(we consider here the first computation of Δ_{out}). Besides

$$\Delta_{out}(v, o^v_m) = d_v(i^v_m, o^v_m) + \Delta_{out}(t(e)) \text{ with } e \in E, h(e) = (i^v_m, v), w(e) = n$$

We denote by $(pr(v), o')$ the tail of edge e and we call $pr(v)$ *the* "predecessor of v". We then can simplify the previous inequality:

$$\Delta(v) = d_v(i^v_m, o^v_m) + \Delta_{out}(pr(v), o') > T,$$

Remark that, after the second pass, there is exactly one register on the edge between v and its predecessor $pr(v)$. If registers had been pushed from the inputs to the outputs, it would have revealed a null weight path P such that $d(P) = \Delta(v) > T$. The last vertex of P would have been v, and the pen-ultimate one would have been $pr(v)$.

As Lemma 6 only deals with line-graphs, "internal edges" have no meaning as there is only one input and one output. Thus, Lemma 6 still can be applied to our new two-pass algorithm.

New proof of lemma 7: We shall now construct a path P as a list of null weight paths (P_i), the head of a "sub-path" being the predecessor of the tail of the next "sub-path":

$$P = (P_n = nwp(pr(t(P_{n-1}))), P_{n-1} = nwp(pr(t(P_{n-2}))), \ldots, P_1 = nwp(pr(t(P_0))), P_0 = nwp(v))$$

Every P_i is a null weight path, by definition of nwp. $P_0 = nwp(v)$ is the longest null weight path ending at v. Let us call its tail v^0_t. Let us now call v^1_h the predecessor of v^0_t. P_1 is then the longest null weight path ending at v_h^1. We construct the rest of P by induction, until we find v_{from_host} as a path tail.

There is exactly one register between the path tails and their predecessor, as, by definition of Δ_{out}, the edge between the two vertices was of minimal weight.

Assume now that r_1 is a register of P carried by the edge $v_1 \xrightarrow{e} v_2$. v_2 is a tail of one of the null weight path P_i. v_1 is its predecessor ($pr(v_2)$). By definition of the predecessor, we have

$$\Delta(v_2) = d_{v_2}(i_m^{v_2}, o_m^{v_2}) + \Delta_{out}(v_1, o') > T$$

As $\Delta(v_1) \geq \Delta_{out}(v_1, o')$, we can conclude that

$$\Delta(v_1) > T - d_{v_2}(i_m^{v_2}, o_m^{v_2})$$

which shows that if r_1 was pushed further from the tail of P, it would reveal a null weight path, ending at $v2$, of delay $\Delta(v_1) + d_{v_2}(i_m^{v_2}, o_m^{v_2}) > T$. ■

Finally, the proof of theorem 6 can be directly applied to our new graph model.

6.3 Algorithm and complexity

In conclusion, the two-pass algorithm can be generalized to our new delay model, conditionally to the described modification in the computation of Δ. The complexity remains the same.

7 Conclusion

We have dealt with various instances of the retiming problem. Our initial motivation was dictated by our target application: on-line computer arithmetic circuits are naturally pipelined circuits without cycles, hence the search for more efficient retiming algorithms applicable to DAGs.

We have succeeded in improving the known complexity of clock minimization problems. Further work could be aimed at improving the complexity of the register minimization problem. It is not clear that a more efficient solution can be found for DAGs than for arbitrary graphs with cycles. However, computer arithmetic circuits usually involve regular computational elements, and this characteristic may prove helpful. For instance, the "Borrow Save" adder of Figure 1 only involves identical devices of input degree 3 and output degree 2. Thus, in this case, our two-pass algorithm also minimizes the total register number, as registers are pushed, in the second pass, from inputs to outputs.

Finally, there remain many interesting open problems in the area. For instance, computational devices are usually selected from a cell library, and we can have the freedom to select, say, among several adders with different delays and input/output degrees. For example, if a "Borrow Save" adder has a very interesting delay, it is due to the redundant number representation used, which means twice more registers to memorize a number. This could be yet another parameter of the fundamental design optimization problem to be solved: match a clock period constraint while minimizing the total register number.

Acknowledgment

We would like to gratefully thank Alain Darte for his careful reading of this paper, and for his numerous comments and suggestions.

References

[1] Tsing-Fa Lee Allen, C.-H Wu Wei-Jeng Chen, Wei-Kai Cheng, and Youn-Long Lin. On the relationship between sequential logic retiming and loop folding. In *Proceedings of the SASIMI'93*, pages 384–393, Nara, Japan, October 1993.

[2] J.C. Bajard, S. Kla, and J.M. Muller. Bkm: A new hardware algorithm for complex elementary functions. *IEEE Transactions on Computers*, pages 598–601, 1994.

[3] J. Biesenack, T. Langmaier, M. Münch, and N. Wehn. Scheduling of behavioural VHDL by retiming techniques. In *Proceedings of the Euro-DAC'94*, pages 546–551, September 1994.

[4] M.D. Ercegovac and K.S.Trivedi. On line algorithms for division and multiplication. *IEEE Transactions on Computers*, 7:681–687, 1977.

[5] A. Münzer G. Hemme. Converting combinational circuits into pipelined data paths. In *Proceedings of the ICCAD'91*, pages 368–371, November 1991.

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

[7] C.E. Leiserson and J.B. Saxe. Optimizing synchronous systems. *Journal of VLSI and Computer Systems*, 1:41–67, 1983.

[8] C.E. Leiserson and J.B. Saxe. Retiming synchronous circuitry. *Algorithmica*, 6:5–35, 1991.

[9] M.C.C. Papaefthymiou. *A Timing Analysis and Optimization System for Level-Clocked Circuitry.* PhD thesis, Massachusetts Institute of Technology, September 1993.

[10] J.B. Saxe. *Decomposable Searching problems and Circuit Optimization by Retiming: Two Studies in General Transformations of Computational Structures.* PhD thesis, Carnegie Mellon University, August 1985.