Short Papers

Retiming DAG's

P. Y. Calland, A. Mignotte, O. Peyran, Y. Robert, and F. Vivien

***Abstract*— This paper is devoted to a low-complexity algorithm for retiming circuits without cycles, i.e., those whose network graph is a direct acyclic graph (DAG). On one hand, DAG's have a great practical importance, as shown by the on-line arithmetic circuits used as a target application in this paper. On the other hand, retiming is a costly design optimization technique, in particular when applied to large circuits. Hence the need to design a specialized retiming algorithm to handle DAG's more efficiently than general-purpose retiming algorithms. Our algorithm dramatically improves on current solutions in the literature: we gain an order of magnitude in the worst case complexity, and we show convincing experimental results at the end of this paper.**

I. Introduction

Retiming is a technique used to optimize synchronous very-large-scale-integration (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 (longest path without registers) and/or the state (total number of registers) are the most frequently used. The survey paper of Leiserson and Saxe [9] 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 in latency. Section III is devoted to a review of known results concerning various instances of the retiming problem.

Retiming is a powerful design optimization technique. However, retiming algorithms have a large cost that limits their use: the efficient retiming of large circuits remains computationally very demanding, even though sophisticated optimization techniques have been introduced [13], [10], [16].

The main contribution of this paper is a new algorithm for retiming circuits without cycles, i.e., those whose network graph is a direct acyclic graph (DAG). DAG's are very commonly encountered in VLSI design, hence the importance of deriving efficient specialized algorithms for retiming such circuits. Section II emphasizes the practical importance and usefulness of scheduling DAG circuits by providing a target example: the design optimization of on-line arithmetic operators. We point out that several other application domains, such as real-time design, and more generally applications where data are circulated in a serial way, lead to acyclic circuits. Last, our result on DAG's may help further the efficiency of retiming general circuits.

Our new algorithm for retiming a DAG only requires two passes over the network graph, as opposed to $|V|$ passes in [9], where $|V|$ is the number of nodes in the network graph. The paper is organized as follows. Section II is devoted to our target application. As already stated, Section III is devoted to a review of existing literature, together with a formal statement of the retiming problems. Our algorithms are given in Section IV. Section V is devoted to experimental results: we compare the execution time of our new retiming algorithm against that of Leiserson and Saxe. We give some final remarks and conclusions in Section VI.

II. Motivating Example

In on-line arithmetic, operands circulate through operators in a digit-serial fashion, most significant digit first. On-line arithmetic was introduced by Ercegovac and Triveli [6], who proposed algorithms for on-line multiplication and division. The operands enter the design at time t, and the first digit of the result is 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. Each new digit of the result improves the final result precision. The computation can stop as soon as the required accuracy is reached.

This serialization is possible only if all the operands circulate the most significant digit first. Unfortunately, the usual serial algorithms for addition and multiplication work the least significant digit first. A specific number system is then used, namely, the signed digit redundant number system, or *borrow-save* system, whose redundancy makes it possible to perform addition and multiplication with no carry propagation. For instance, Fig. 1 shows a borrow-save adder for redundant arithmetic. This borrow-save adder is made out of plus plus minus (PPM) cells, as illustrated in Fig. 2.

Several algorithms and corresponding architectures have been proposed to perform basic and complex operators such as addition, multiplication, division, square root, and trigonometric functions [1], 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, Fig. 2 corresponds to one stage of the adder of Fig. 1. Registers are inserted to make sure that c^-_{n-1}, b^-_{n-1} of the previous stage is 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 Fig. 3(a): the critical path of the application is six PPM cells.

Nevertheless, the resulting clock period may not be satisfactory to the 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 (latches) 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 (a latch on every path) as shown in Fig. 3(b); the critical path is reduced to four PPM cells. Unfortunately, in that case, the critical path is not optimized: it can

Manuscript received October 25, 1995; revised May 18, 1998. This paper was recommended by Associate Editor R. Camposano.

The authors are with CNRS, Laboratoire LIP, Ecole Normale Supérieure de Lyon, Lyon Cedex 07, 69364 France.

Publisher Item Identifier S 0278-0070(98)09357-9.

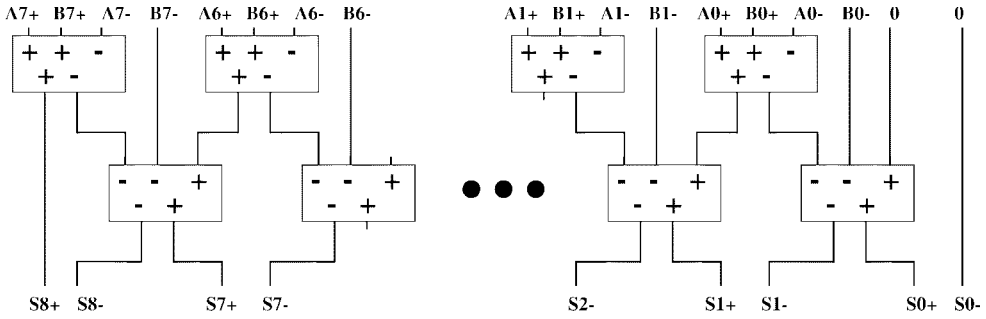

Fig. 1. A borrow-save adder made of PPM cells. The digits a_i, b_i, and s_i, whose values are $-1, 0$, or 1, are represented by the bits a_i^+, a_i^-, b_i^+, b_i^- such that $a_i = a_i^+ - a_i^-$, $b_i = b_i^+ - b_i^-, \cdots$.

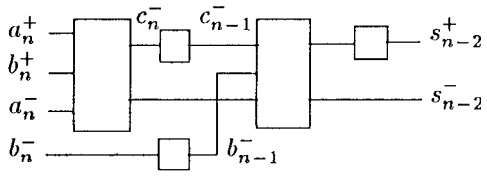

Fig. 2. A latency-two borrow-save serial adder, working the most significant digit first. Square cells are registers.

be reduced to three PPM cells by moving some registers, as shown in Fig. 3(c).

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 it 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.

III. Review of Retiming Techniques

A. A Graph-Theoretic Framework

Computational devices (VLSI circuits or VHDL programs) are represented by a finite, connected, vertex-weighted, edge-weighted directed multigraph $G = (V, E, d, w)$. 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 $w(e)$, which also corresponds to the number of "wait until clock" statements in VHDL programs. Edge delays are nonnegative integers.

We need a few definitions and notations, which we borrow from the reference paper of Leiserson and Saxe [9]. For an edge $e : u \to v$, 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 $v_{\text{from\_host}}$ and $v_{\text{to\_host}}$, with zero delay, to model the interface of the graph with the external world: $d(v_{\text{from\_host}}) = d(v_{\text{to\_host}}) = 0$. We let $V' = V \cup \{v_{\text{from\_host}}, v_{\text{to\_host}}\}$. Finally, we define a null-weighted edge from $v_{\text{from\_host}}$ to each input node and a null-weighted edge from each output nodes to $v_{\text{to\_host}}$. These edges are called, respectively, *input edges* and *output edges*. They form the set of *interface edges* that we denote by I. We let $E' = E \cup I$ (see Fig. 4 for an illustration).

For any simple path $P = v_0 \stackrel{e_0}{\to} v_1 \stackrel{e_1}{\to} \cdots \stackrel{e_{k-1}}{\to} v_k$, 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 $w(P) = 0$.

Leiserson and Saxe [9] deal with network graphs G that do 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 $\Phi(G) = \max\{d(P); w(P) = 0\}$. Intuitively, the clock period is the maximum amount of propagation delay through which any signal must ripple between clock ticks (as delays are integer values, clock period is an integer value). 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 $v \in V$: $r(v)$ registers are subtracted from (respectively, added to) the weight of each edge leaving v (whose tail is v), while $r(v)$ registers are added to (respectively, subtracted from) the weight of each edge entering v (whose head is v). Formally, a retiming function r is a mapping from V' to Z such that $r(v_{\text{from\_host}}) = r(v_{\text{to\_host}}) = 0$. It leads to a new edge-weighting function w_r defined for an edge $u \stackrel{e}{\to} v$ 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 $S_r(G)$ 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 1) is the basic feasibility problem and can be solved in $O(|V||E|)$ in the most general instance [9]. There are two variants to the optimization problem. The first variant is the *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 [9], [14], and the references therein for several bounds.

Problem 2) is the second variant, the *clock minimization problem*, and can be solved in $O(|V||E| \log |V|)$ in the most general formulation [9]. In fact, it turns out that a way to solve Problem 2) is to

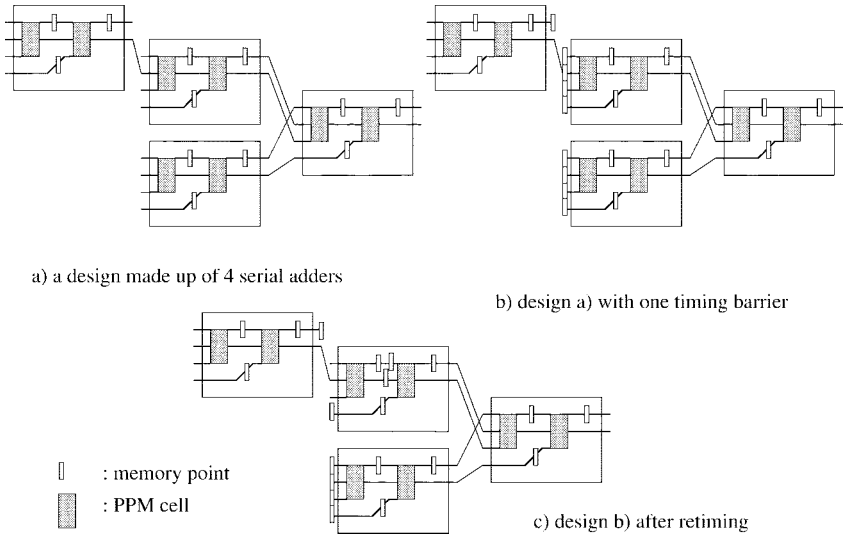

Fig. 3. Adding and moving registers to decrease the period.

repeatedly solve several instances of Problem 1) using a binary 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 [12]).

Our main result is that the generic algorithms of [9] can be specialized when applied to DAG's. Section IV presents a solution to Problem 1) that runs in $O(|E|)$ and one to Problem 2) that runs in $O(|E|(\log|V| + \log D))$.

B. Review of Related Problems

Retiming techniques have been applied to several related problems. Retiming was first introduced by Leiserson and Saxe in [8] in order to solve Problem 1) for systolic circuits (unit-delay circuits with at least one register between two functional units). Their solution, using the Bellman–Ford algorithm, requires $O(|E||V|)$ time.

Wehn *et al.* linked high-level synthesis and retiming in [2], 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 to these problems.

Some researchers proposed interesting results for simpler problems, like Papaefthymiou [12], who proposed an algorithm that gives the minimum clock period of a circuit with at most l levels of registers, where l is a given value. The algorithm runs in $O(|E|\log|V|)$, but the initial circuit has to be empty of registers. Therefore, it is only interesting when one needs to pipeline a combinational circuit.

In the same idea of optimizing a combinational circuit, Munzer and Hemme [11] proposed an algorithm in $O(|E|\cdot|V|^2)$ to solve the register minimization problem. From a register-empty circuit, they apply as-soon-as-possible and as-late-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, Chen *et al.* showed the relationship between retiming and loop folding [4]. They propose a solution to Problem 2) using an as-soon-as-possible pipeline algorithm. However, they did not formally explain the way registers are moved within the circuit, which is the most important step in terms of complexity.

In a later paper [9], Leiserson and Saxe improved their technique 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 cost function. The complexity is $O(|V|^3\log|V|)$.

Several recent developments are nicely surveyed by Shenoy [15]. In particular, retiming large circuits remains a practical problem because of the long running time of the (polynomial but) costly algorithms of Leiserson and Saxe. Efficient optimizations to reduce the number of constraints are described by Sapatnekar and Deokar [13] and by Maheshwari and Sapatnekar [10]. Szymanski [17] proposes several algorithms to generate (and process) relevant linear constraints only. See also Shenoy and Rudell [16] for heuristics (based on retiming selected subgraphs) that dramatically reduce the overall computation time. Last, several extensions of retiming problems to incorporate multiplexing and time folding are discussed by Fluiter *et al.* [5].

The high complexity of retiming algorithms motivates the need for low-complexity solutions specialized to some classes of network graphs. We present our low-cost algorithm to retime DAG's in the next section.

IV. 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 $d : V \rightarrow N$ can take any nonnegative value. However, we consider that $d(v_{\text{to\_host}}) = d(v_{\text{from\_host}}) = 0$ in all cases.

A. 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 $v_{\text{from\_host}}$, with respect to retiming rules: when processing a node, we can add (suppress) the same number of registers on each incoming (outgoing) edge, provided that the weight on each edge remains nonnegative. In

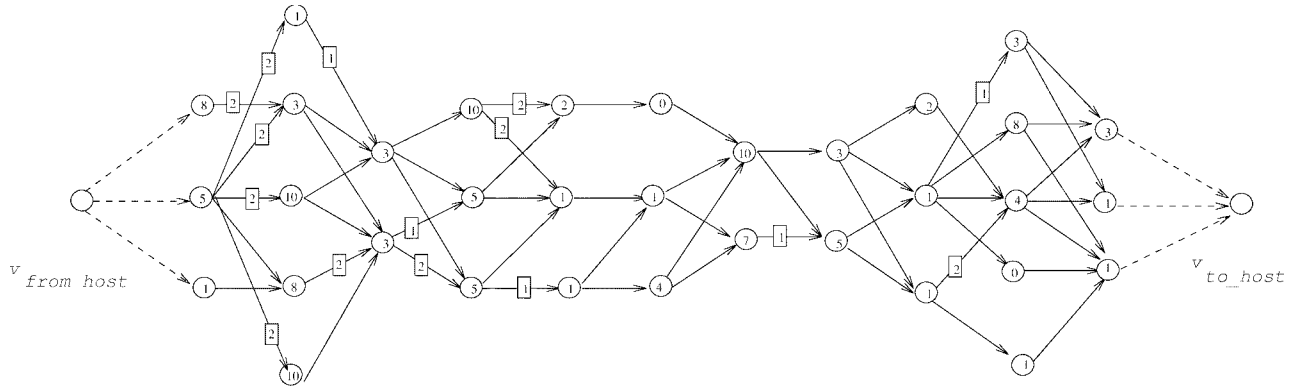

Fig. 4. Initial register distribution. Note that operators have different delays.

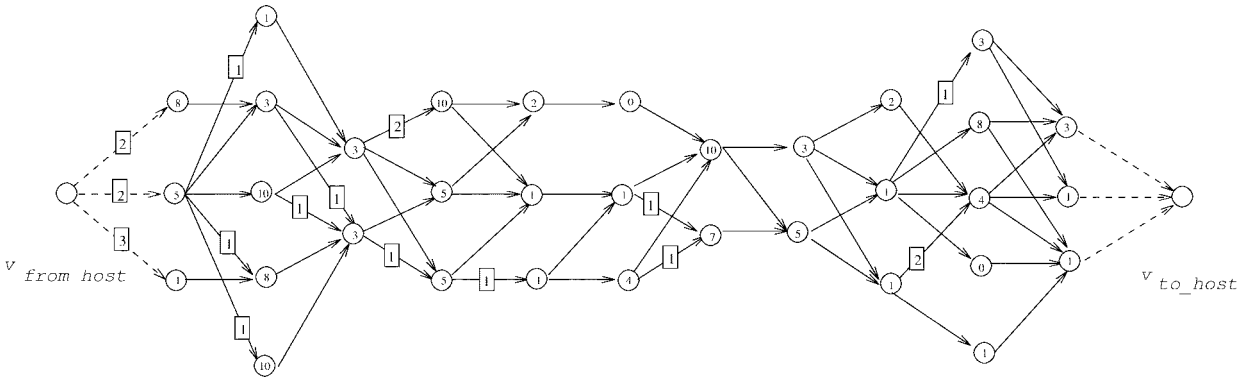

Fig. 5. Resulting distribution after the first pass of the algorithm.

the second pass, all registers are moved as far as possible from the node $v_{\text{from\_host}}$, 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 nonzero in- and out-degrees and that the set of all vertices of G is $V' = V \cup \{v_{\text{from\_host}}, v_{\text{to\_host}}\}$.

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
      $n = \min_{e \in E \mid t(e) = v} w(e)$
      $\forall e \in E \mid t(e) = v, w(e) \leftarrow w(e) - n$
      $\forall e \in E \mid h(e) = v, w(e) \leftarrow w(e) + n$
    enddo
Second pass
  $\Delta(v_{\text{from\_host}}) \leftarrow 0$
  for (each vertex $v \in V$ in topological order)
    do
      $n \leftarrow \min_{e \in E \mid h(e) = v} w(e)$
      $\backslash *$ Δ *computation* statement,
      taken over all edges whose head is v and whose weight is n $*\backslash$
      $\Delta(v) \leftarrow d(v) + \max\{\Delta(t(e)) : e \in E, h(e) = v$ and $w(e) = n\}$
      if $(\Delta(v) > T)$ **then**
        $\backslash *$ *Leave a register* case $*\backslash$
        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.

Theorem 1: 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.

Complexity: A detailed proof of the correctness of the two-pass algorithm (Theorem 1) is available in the companion research report [3]. 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 $O(|E|)$.

B. Period Minimization and Complexity

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

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 $t_{\min} \leftarrow t + 1$
        $t_{\max} \leftarrow \min(t_{\max}, \Phi(G_r))$
      endif
  Until $t_{\max} = t_{\min}$.

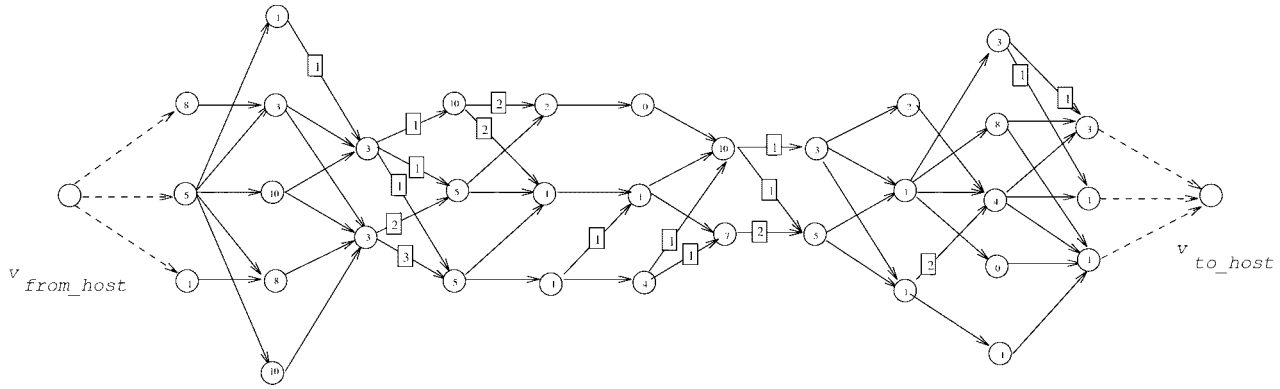

Fig. 6. Resulting distribution after the second pass of the algorithm.

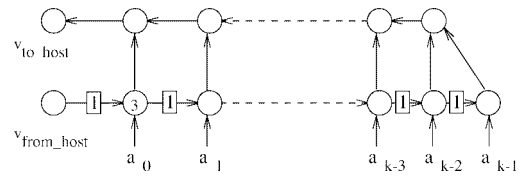

Fig. 7. Digital correlator. Adders have a delay equal to seven, while δ-operators have a delay equal to three.

TABLE I
COMPUTATION TIMES (IN SECONDS) FOR DIFFERENT APPLICATIONS

	# vertices	# edges	clock period	time_{LS}	time_{two}
SVD	44	56	26	0.19	0.07
3-Way	25	33	8	0.10	0.06
Correlator 10	19	28	14	0.09	0.06
Correlator 50	99	148	14	0.56	0.09
Correlator 100	199	298	14	2.05	0.11

This algorithm uses the "two-pass algorithm" as a kernel in a binary 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 $(G_r = (V, E, d, w_r))$.

We illustrate the execution of the algorithm on an example (see [3] for a detailed proof). The original graph to be retimed is given in Fig. 4. We assume that the optimal period $\Phi_{\text{opt}}(G) = 18$ is known, and we apply the two passes. The graph obtained after the first pass is shown in Fig. 5. Fig. 6 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 the general algorithm is $O(|E|(\log |V| + \log D))$, where $D = \max_{v \in V} d(v)$; indeed, $\Phi_{\text{opt}}(G)$ is upper bounded by $|V|D$.

Hence, we have proposed a solution to Problem 1) that runs in $O(|E|)$, and a solution to Problem 2) that runs in $O(|E|(\log |V| + \log D))$.

V. APPLICATIONS

In this section, we present some practical applications of our two-pass algorithm, and we compare its computation times with those of the original retiming algorithms introduced by Leiserson and Saxe [9].

A. The Digital Correlator

Consider the example of the digital correlator (see Fig. 7) that Leiserson and Saxe have chosen to illustrate their retiming technique [9]. The digital correlator is a DAG that takes a stream of bits $x_0, x_1, \ldots$ as input and compares it with a fixed-length pattern $a_0, a_1, \cdots, a_{k-1}$. It produces as output the value y_i

$$y_i = \sum_{j=0}^{k-1} \delta(x_{i-j}, a_j).$$

Let Cor_k denote the correlator of length k (i.e., with k δ-operators and $k-1$ adders). Leiserson and Saxe use Cor_4 as a target example in [9]. We use correlators of size $k = 10$, $k = 50$ and $k = 100$ in our experiments.

B. The Three-Way Circuit

Retiming is widely used to introduce pipelining in large operators. Fig. 8 shows the graph of an operator that performs the squaring of big numbers: a number A is decomposed into three parts, A_0, A_1, and A_2. The squaring of A uses only the squaring of the A_i, and simple operations (additions, shifts, and a final division by three) [18]. Our aim here is to insert one level of pipeline. Therefore, we insert one register on each input edge, and we run the retiming algorithms to determine where to place these registers in order to obtain the optimal clock period.

C. The SVD Circuit

Consider the circuit represented in Fig. 9, which is used to compute the singular value decomposition (SVD) of a rectangular matrix (see [7]). The SVD is a factorization technique used in many signal-processing transformations. It aims at factorizing a matrix A as $A = U\Sigma V^T$, where $U^T U = I$, $V^T V = I$, and Σ is a diagonal matrix with nonnegative elements. The principle is to perform a series of 2×2 SVD along the main diagonal of A, where each 2×2 SVD is realized by a two-sided rotation that diagonalizes the submatrix (algorithm FHSVD). Fig. 9 is the graph corresponding to the SVD scheme based on the on-line arithmetic implementation of the FHSVD algorithm.

D. Results

All timing results are summarized in Table I, where $time_{\text{two}}$ is the computation time needed to obtain the optimal clock period with our two-pass algorithm, and $time_{\text{LS}}$ is the corresponding time with the modified algorithm of Leiserson and Saxe. The correlator circuit was used to create larger graphs, to show the efficiency of our method when it comes to processing medium-size graphs.

Roughly speaking, the gain is 50% for small circuits, and becomes more and more significant as the size of the circuit increases.

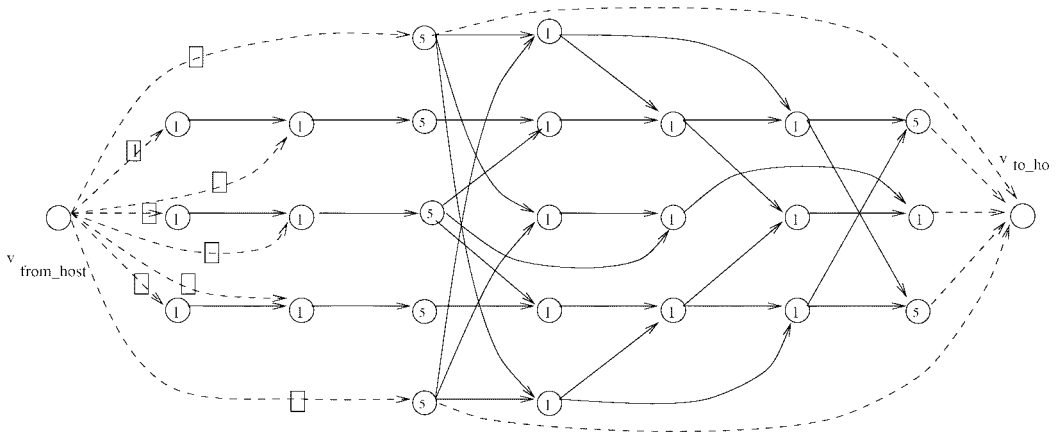

Fig. 8. Introducing one level of pipeline into the three-way method.

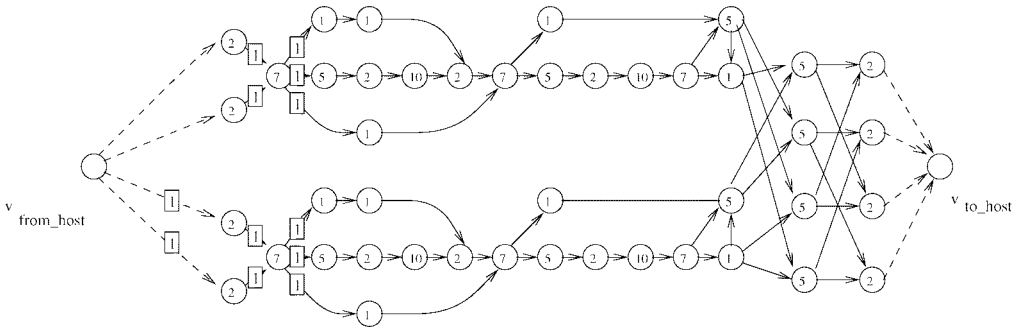

Fig. 9. On-line scheme for algorithm FHSVD.

VI. 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 DAG's.

We have succeeded in improving the known complexity of clock minimization problems for DAG's, and we have given experimental evidence of the superiority of our new retiming algorithms. 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 DAG's 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 Fig. 1 only involves identical devices of input degree three and output degree two. 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.

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, this is due to the redundant number representation used, which means twice as many 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

The authors would like to thank A. Darte for his careful reading of this paper. They are indebted to the reviewers for their comments and suggestions and for pointing out several bibliographical references.

References

[1] J. C. Bajard, S. Kla, and J. M. Muller, "BKM: A new hardware algorithm for complex elementary functions," *IEEE Trans. Comput.*, vol. 24, pp. 598–601, 1994.

[2] J. Biesenack, T. Langmaier, M. Münch, and N. Wehn, "Scheduling of behavioral VHDL by retiming techniques," in *Proc. Euro-DAC'94*, Sept. 1994, pp. 546–551.

[3] P. Y. Calland, A. Mignotte, O. Peyran, Y. Robert, and F. Vivien, "Retiming dags," LIP, Ecole Normale Supérieure de Lyon, Tech. Rep. 95-18, Sept. 1995. Available WWW: www.ens-lyon.fr/LIP/publis.us.html.

[4] W.-J. Chen, W.-K. Cheng, T.-F. Lee, A. C.-H. Wu, and Y.-L. Lin, "On the relationship between sequential logic retiming and loop folding," in *Proc. SASIMI'93*, Nara, Japan, Oct. 1993, pp. 384–393.

[5] B. de Fluiter, E. H. L. Aarts, J. H. M. Korst, W. F. J. Verhaeh, and A. van der Werf, "The complexity of generalized retiming problems," *IEEE Trans. Computer-Aided Design*, vol. 15, no. 11, pp. 1340–1353, 1996.

[6] M. D. Ercegovac and K. S. Trivedi, "On line algorithms for division and multiplication," *IEEE Trans. Comput.*, vol. C-7, pp. 681–687, 1977.

[7] M. D. Ercegovac and P. K.-G. Tu, "Application of on-line arithmetic algorithms to the SVD computation: Preliminary results," in *Proc. 10th Symp. Computer Arithmetic*, P. Kornerup and D. W. Matula, Eds., 1991, pp. 246–255.

[8] C. E. Leiserson and J. B. Saxe, "Optimizing synchronous systems," *J. VLSI Comput. Syst.*, vol. 1, pp. 41–67, 1983.

[9] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, pp. 5–35, 1991.

[10] N. Maheshwari and S. Sapatnekar, "Efficient retiming of large circuits," *IEEE Trans. VLSI Syst.*, vol. 6, no. 1, pp. 74–83, 1998.

[11] A. Münzner and G. Hemme, "Converting combinational circuits into pipelined data paths," in *Proc. ICCAD'91*, Nov. 1991, pp. 368–371.

[12] M. C. C. Papaefthymiou, "A timing analysis and optimization system for level-clocked circuitry," Ph.D. dissertation, Massachusetts Institute of Technology, Cambridge, Sept. 1993.

[13] S. Sapatnekar and R. B. Deokar, "Utilizing the retiming-skew equivalence in a practical algorithm for retiming large circuits," *IEEE Trans. Computer-Aided Design*, vol. 15, no. 10, pp. 1237–1248, 1996.

[14] J. B. Saxe, "Decomposable searching problems and circuit optimization by retiming: Two studies in general transformations of computational