ABC Journals
a new era in publishing
ABC Research House
Asian Journal of Applied Science and Engineering
Abbreviated key title: Asian j. appl. sci. eng.
Access this article online
http://journals.abc.us.org/index.php/ajase/issue/archive

Extended Retiming: Transforming Synchronous Data-Flow Graphs to Minimize Clock Period

Timothy W. O'Neil

Professor, Department of Computer Science, The University of Akron, Akron, Ohio, USA

ARTICLE INFO

Received: May 16, 2015
Accepted: Dec 15, 2015
Published: Dec 26, 2015

*Corresponding Contact
Email: toneil@uakron.edu

crossref doi Prefix 10.18034

ABSTRACT

Purpose: To develop an efficient transformation technique for synchronous data-flow graphs that produces an equivalent graph with minimum clock period.
Design/Methodology: We develop the technique via mathematical proof and test on numerous filters.
Findings: Our technique achieves the stated purpose.
Originality/Value: As discussed in the paper, the problem is significant and has been extensively studied. Any method prior to this one either doesn't achieve the goal or does so at great cost. The discussed technique optimizes the input graph efficiently while minimizing the resultant hardware costs.

Key Words: Data path optimization, hardware-software codesign, data flow architectures

Source of Support: University of Akron's Buchtel College of Arts and Sciences,

No Conflict of Interest: Declared

How to Cite: O'Neil TW. 2015. **Extended Retiming: Transforming Synchronous Data-Flow Graphs to Minimize Clock Period** *Asian Journal of Applied Science and Engineering,* 4, 201-218.

INTRODUCTION

Increasingly complicated embedded systems such as systems-on-chip (SoC) present an key challenge to today's designers. Engineers build such systems to exact specifications with consideration of restrictions on execution time, area, cost, etc. For high-performance embedded systems with limited on-chip resources, the hindrance in developing an efficient design is finding the minimum number and nature of functional units for executing an application with timing and code size requirements.

Many applications represented by embedded systems have their most computation-intensive and time-critical sections of code found in the repeating patterns of loops. We represent such instructions by *multi-rate* or *synchronous data-flow graphs* (SDFGs) (Parhi and Messerschmitt, 1991).Toenhance the operation of the system executing such a program, we could administer graph-transformation techniques to theSDFG in an attempt to increase the degree of parallelism thereby decreasing overall execution time. However, for some graphs, these techniques may not be able to produce a transformed SDFG with the best schedule length. In this paper, we will offer a new transformation methodthat does deliver optimal results. When compared with the traditional methods, our new technique quickly and easily produces a transformed graph without increasing the size of the SDFG.

As said earlier, each of the tasks comprising an embedded system has its own execution time and list of dependencies with other tasks. Typically, graph transformation techniques such as *retiming* (O'Neil and Sha, 2001) are employed to minimize the execution rates in a parallel or pipeline system. In the course of retiming, delays are redistributed among the edges so that the hardware usage is optimized while the application's function remains the same. Retiming has traditionally been applied to SDFGs to optimize the application's schedule of tasks (O'Neil and Sha, 2001), (O'Neil *et al*, 2011). It was later broadened to extend vectorization capabilities (Zivojnovic *et al*, 1994b) or minimize total delay count (Zivojnovic *et al*, 1994a).

The benefits of retiming single-rate data-flow graphs (DFGs) are widely reported in the literature, particularly when combined with other transformations (O'Neil and Sha, 2005), (Chao and Sha, 1997). However, our prior work (O'Neil, 2011), (O'Neil, 2014) indicates thatsuch combinations may not always be possible for SDFGs.Therefore, the primary point of this research becomes fundamentally changing the actual retiming methodology in order to achieve the same advantages for SDFGs. The contributions of this paper are as follows:

1. We propose a new graph transformation technique, *extended retiming*, which reduces the clock period of a retimed SDFG to the point where it matches the cycle period of the SDFG's optimal schedule. Also, it may be combined with DAG scheduling to form a flexible scheduling method.
2. Because of this, we can design an efficient algorithm for verifying the existence of an extended retiming $r$that will make $cl(G_r) \leq c$ for a given integer c.
3. We demonstrate an algorithm for deriving extended retimings.

Note that a preliminary version of this work for single-rate data-flow graphs has previously appeared (O'Neil and Sha, 2005), (O'Neil *et al*, 1999a), (O'Neil *et al*, 1999b). This initial work was incorporated into the IDOM framework (Zhuge *et al*, 2006) with other transformations to find minimum configurations of DSP processors satisfying timing and code size constraints. We are broadening it to the more general synchronous data-flow model.

In the next section, we will formalize fundamental concepts such as synchronous data-flow graphs, retiming and static scheduling. We present the theme of this paper in Section 3, along with the major theorems. Next, we consider detailed examples of this work. Finally, we summarize our work and provide further questions for exploration.

BACKGROUND

In this section, we review the relevant definitions and ideas regarding synchronous data-flow graphs to formalize these concepts.

Synchronous Data-Flow Graphs

Originally developed in (Lee and Messerschmitt, 1987), a*synchronousdata-flow graph* (*SDFG*) is a finite, directed, weighted graph $G = \langle V,E,d,t,p,k \rangle$ where

1. V is the vertex set of nodes that transform input data streams into output streams;
2. $E \subseteq V \times V$ is the edge set, representing channels that carry data streams;
3. $d : E \rightarrow \mathbb{N} \cup \{ 0 \}$ is a function with $d(e)$ the number of initial tokens (delays) on edge e;

4. $t : V \rightarrow \mathbb{N}$ is a function with $t(v)$ the execution time of node v;
5. $p : E \rightarrow \mathbb{N}$ is a function with $p(e)$ the number of data tokens produced at e's source node to be carried by e; and
6. $k : E \rightarrow \mathbb{N}$ is a function with $k(e)$ the number of data tokens consumed from e by e's sink node.

(In this definition $\mathbb{N}$is the set of natural numbers {1, 2, 3, ...}.) If $p(e) = k(e) = 1$ for all edges e, we say that G is a *homogeneous data-flow graph* (*HDFG*). Some authors refer to HDFGs as *single-rate data-flow graphs* or simply *data-flow graphs*.

To illustrate, consider the SDFG given in Fig. 1 below. We will assume that$t(v)$ = 2 for all nodes v in this simple example. The small numbers at either end of an edge denote tokens produced or consumed. In this example, the numbers at either end of the data channel connecting C and A, hereafter denoted (C,A), indicate that node C produces one token on this edge when it executes. Similarly, node A consumes two tokens from this edge each time it runs. The short bar-lines cutting (A,B) represent initial tokens to be consumed by B.

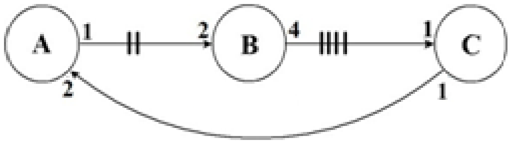


Fig. 1: Sample SDFG

It is sometimes useful to characterize an SDFG by its$|E| \times |V|$ *topology matrix*. Each row corresponds to one edge in the graph while each column corresponds to a node. A positive $(i,j)^{th}$ entry in the topology matrix indicates the number of tokens produced by the j^{th} node on the i^{th} edge. Similarly a negative entry here gives the number of tokens consumed by node j from edge i. All other entries are zero. As an example, the topology matrix of Fig. 1 is

$$M = \begin{bmatrix} 1 & -2 & 0 \\ 0 & 4 & -1 \\ -2 & 0 & 1 \end{bmatrix}$$

For purposes of this representation, nodes A, B and C are designated nodes 0, 1 and 2, respectively. We number the edges in the order (A,B), (B,C) and (C,A) in the matrix.

From (Lee and Messerschmitt, 1987), we know that a repeating sequential schedule for a SDFG Gcan be constructed if the rank G's topology matrix is one less than $|V(G)|$. If this condition holds there is a positive integer vector $\overline{q}$ in the null space of the topology matrix called a *repetition vector* for G. The repetition vector for G with the smallest norm is known as the *basic repetition vector* (*BRV*) for G (Lee and Messerschmitt, 1987). For example, the BRV for the SDFG in Fig. 1 is $\overline{q}$ = [2 1 4]^T. The elements of a BRV $\overline{q}$ indicate that q_j copies of node v_j must execute during every iteration of the static schedule. In our example, we must schedule two copies of A, one of B and four of C each time. A SDFG is *consistent* if it has a BRV.

The Iteration Bound of a SDFG

An *iteration* of an SDFG is simply a single execution of all copies of all nodes of the SDFG. The *cycle period* of the SDFG is then the average computation time of one iteration within a given schedule. In synchronous graphs containing loops, this figure is bounded from below by the graph's *iteration bound*(Renfors and Neuvo, 1981). We use the notation $B(G)$ to represent G's iteration bound, derived by computing the time-to-delay ratios for all cycles in the graph and retaining the maximum of these figures. While complicated to derive for SDFGs in general, as shown in (O'Neil *et al.*, 2011), we can quickly estimate this value via the inequality:

$$B(G) \geq \max_{loop\ \ell \in G} \frac{\sum_{v \in \ell} t(v)}{\sum_{e=(u,v)\in\ell} \lfloor d(e)/\max\{q_u, q_v\} \rfloor} \qquad (1)$$

where the synchronous graph G has a basic repetition vector of $[q_0,\ldots,q_{|V|-1}]$. Returning to Fig. 1, the depicted graph includes one loop with an adjusted delay count of two and node computation times summing to 6; therefore $B(G) \geq 3$ for Fig. 1.

Clock and Cycle Period

There are two measures of quality that we will use to assess static schedules. One is the *cycle period* based on a given schedule. For the other, define a *pathp* in an SDFG G to be a connected sequence of nodes and edges. We use the notation $T(p)$ to represent the sum of the individual node execution times along the path. We formally define the *clock periodcl*(G) of an SDFG G as "the maximum amount of propagation delay through which any signal must pass between clock ticks" (Leiserson and Saxe, 1991). Mathematically, this is the computation time of the longest path having insufficient delays to halt a signal between clock ticks; in other words

$$cl(G) = \max_{p \in G} T(p)$$

where $p \in G$ is a path with $d(e) < \max\{q_u,q_v\}$ for all $e = (u,v) \in p$. For HDFGs, this is simply the computation time of the longest zero-delay path. Our example graph in Fig. 1 has a clock period of 4.

We already see the discrepancy. For any distribution of the delays in Fig. 1 the clock period remains at least 4 while the smaller iteration bound for the same SDFG implies the existence of a more efficient static schedule. In general, the minimum cycle period is less than or equal to the minimum achievable clock period because of the flexibility built into static scheduling. (In cases where these figures match, we will consider both the schedule and hardware configuration to be optimal.) However, the advantages of using graph transformation techniques are clearly visualized and easily understood from working with graph models, motivating our desire to develop a new graph transformation method.

Static SDFG Scheduling

As shown in(O'Neil *et al.*, 2011), there are two models we consider when discussing static scheduling. Formally, a function $s : V \times (\mathbb{N} \cup \{0\}) \rightarrow \mathbb{N} \cup \{0\}$ is an *integral schedule* for a SDFG G where $s(v,i)$ is the starting time of node v in iteration i ($i \geq 0$). Likewise, a function $s : V \times (\mathbb{N} \cup \{0\}) \rightarrow \{r \in \mathbb{Q} \mid r \geq 0\}$ with the same stated properties is a *fractional schedule* for G. In either case, s is *legal* if, for each edge $e = (u,v)$ and iteration $i \geq 0$, $s(u,i) + t(u) \leq s(v, i + \lceil d(e) / \max\{q_u,q_v\} \rceil)$ (O'Neil *et al*, 2011). Such schedules are *repeating for cycle period* $^c/_f$ if $s(v,i+f) =$

$s(v,i) + c$ for all nodes v and iterations i (Chao, 1993). Repeating schedules may be characterized by their initial iterations, given the fact that the full legal schedule may be constructed by commencing a new copy of the partial timetable every c clock ticks. Finally, a schedule is *static* if a node's execution is designated to proceed on the same functional unit in every partial schedule instantiation (Chao, 1993).

As a final point, there are two design approaches to consider when devising a static schedule (Chao, 1993). Our work deals with those following a *non-pipelined implementation* wherein the next occurrence of a node cannot proceed until the prior copy has terminated execution, creating an inherent precedence relation between successive incidences of the same node. Alternately, schedules following a *pipelined implementation* are not subject to this limitation.

SDFG Scheduling Algorithm

Our method (O'Neil *et al*, 2011), (O'Neil, 2012) for devising an integral, repeating static schedule for a given SDFG G and target cycle period c begins with the construction of a *scheduling graph*$G^s = \langle V \cup \{v_0\}, E', \omega, \tau, p, k\rangle$. This model encapsulates all information we will ultimately need to create our schedule. First, each edge $e = (u,v)$ is reweighted according to the formula

$$w(e) = \left\lfloor \frac{d(e)}{\max\{q_u, q_v\}} \right\rfloor - \frac{t(u)}{c}$$

to preserve precedence relations among the nodes. Finally, we seek to uncover negative-weight cycles, indicating an infeasible clock period (O'Neil *et al.*, 2011). Start by inserting a dummy node v_0, then add directed zero-weight edges from this new node to every other node in G. As an illustration, the scheduling graph for Fig. 1 with cycle period 3 appears as Fig. 2 below.

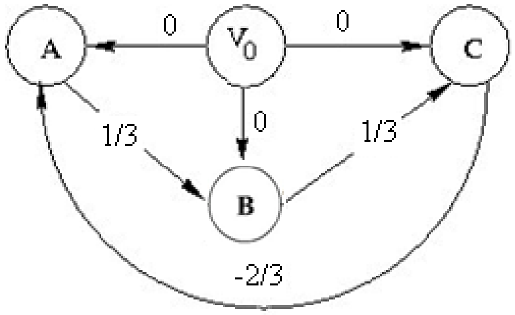


Fig. 2: The scheduling graph for Fig. 1 with cycle period 3

After this, the Bellman-Ford single-source shortest-path algorithm (Cormen *et al.*, 2009) is applied to the scheduling graph. If the algorithm finds a cycle having negative total weight, the proposed cycle period is infeasible, and the algorithm fails. Otherwise, we can obtain the shortest path weights from v_0 to every other node in the scheduling graph. This information is used to derive the starting times for all nodes v in the first f schedule iterations according to the formula

$$\boldsymbol{s(v,i) = \lceil c(i - sh(v)) \rceil}$$

for $i = 0, 1, \ldots, f - 1$. These then repeat every c steps to generate the entire schedule. As in

(O'Neil *et al.*, 2011), this method defines a legal, repeating, static schedule under either of the design styles described above.

Re-examining the scheduling graph forFig. 1, we see that $sh(A) = -⅔$, $sh(B) = -⅓$ and $sh(C) = 0$. This yields start times of 2, 1 and 0 for A, B, and C, respectively. We depict this schedule in Fig. 3 below.

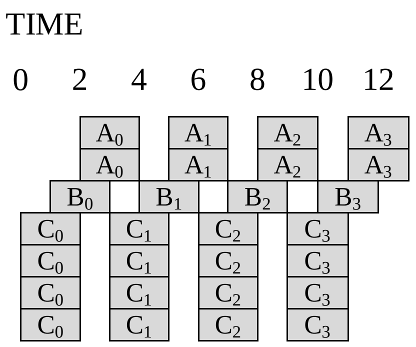


Fig. 3: The partial schedule for Fig. 1 with cycle period 3

EXTENDED RETIMING

Having demonstrated above that traditional retiming will not necessarily result in an optimal schedule, we are now prepared to devise a form of retiming that will.

The General Concept

An *extended retiming* of a SDFG $G = \langle V,E,d,t \rangle$ is a function $r{:}V \rightarrow \mathbb{Q}$such that $t(v) \cdot r(v) \in \mathbb{N} \cup \{ 0 \}$ for all $v \in V$. From this definition, $r(v)$ can be viewed as consisting of an integer part and a fractional part. The integer part $\lfloor r(v) \rfloor$ is the number of delays pushed *to* each *outgoing* edge of v while the fractional part conveys the position of a delay within a split node. Therefore, the value $\lceil r(v) \rceil$ is the number of delays drawn *from* each *incoming* edge of v, and if $\lceil r(v) \rceil - \lfloor r(v) \rfloor = 1$, a group of delays remains inside node $v$to divide it into two sub-nodes having computation times $t(v) \cdot (r(v) - \lfloor r(v) \rfloor)$and $t(v) \cdot (\lceil r(v) \rceil - r(v))$. For example, an extended retiming with $r(A) = 0$, $r(B) = ½$, and $r(C) = 0$ applied to the SDFG described in Fig. 1 above produces the retimed graph of Fig. 4below.

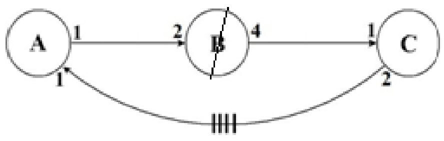


Fig. 4: Fig. 1 retimed.

We see that our new abstraction captures pipelining information that eludes traditional retiming, which will lead to more flexible scheduling options. As with standard retiming, we will denote the DFG retimed by r as $G_r = \langle V,E,d_r,t \rangle$. We utilize the notation$d(u \rightarrow v)$ to represent the total number of delays along an edge $e = (u,v)$, including delays contained within the end nodes $u$and v. As in the traditional case, we will refer to the number of delays on the edge,*not including* delays within end nodes, as $d(e)$. Using the example from Fig. 4, let $e_1 = (A,B)$, $e_2 = (B,C)$ and $e_3 = (C,A)$. Then $d(A \rightarrow B)$ and $d(B \rightarrow C)$ are each 1 due to a split end-node, even though $d(e_1)$ and $d(e_2)$ are each zero, while $d(C \rightarrow A) = d(e_3) = 4$ since there is no split end-node for this edge.

The delay count of the edge following retiming by r will be denoted $d_r(e)$ or $d_r(u\rightarrow v)$ as elsewhere. Based on the above discussion we can generate the following basic definitions.

Definition 1.*Let G be a SDFG and r an extended retiming. Let e =* (u,v) *be an edge in G. Then:*

1. $d_r(e) = d(e) + \max\{q_u,q_v\}\,(\lfloor r(u)\rfloor - \lceil r(v)\rceil)$.
2. $d_r(u\rightarrow v) = d_r(e) + q_u\,(\lceil r(u)\rceil - \lfloor r(u)\rfloor)$
   $+ q_v\,(\lceil r(v)\rceil - \lfloor r(v)\rfloor)$.

As with traditional retiming, an extended retiming is *legal* if $d_r(e) \geq 0$ for all edges $e\in E$ and *normalized* if $\min_v \lfloor r(v)\rfloor = 0$. To normalize an extended retiming, we must subtract $\min_v \lfloor r(v)\rfloor$ from all values $r(v)$.

Subpaths

Our previous definition of a path assumes that a path includes all pieces of its initial and final nodes. On the other hand, we will define a connected sequence of nodes and edges which includes *only some* of the pieces of its initial and final nodes to be a *subpath*. For example, consider the graph in Fig. 4. Any path that begins or ends with node B must include both pieces of the node while a subpath may begin or end at either piece and need not include both. Thus a path is a subpath, but a subpath is not necessarily a path.

It is clear that some pieces of the end-nodes are likely missing when we discuss a subpath. For example, suppose that u and v are the end-nodes of the path p. We may split each of these nodes by a delay into right and left subnodes. If w is a divided node, we will designate the left and right subnodes for w as w_L and w_R, respectively.

The total computation time of a subpath is the total computation time of the full path, minus the computation times of any excluded half-nodes. Computing the total delay count is not as straightforward for synchronous graphs due to differing production and consumption rates of the individual edges. More useful is the *lower delay bound* of the path, defined as

$$\delta(p) = \sum_{e=(u,v)\in p} \left\lfloor \frac{d(e)}{\max\{q_u, q_v\}} \right\rfloor.$$

We can now demonstrate the following:

Lemma 2.*Let G be a SDFG and c a potential clock period. cl(G) ≤ c if and only if, for every subpath p from u to v in G with no internal split nodes and T(p)> c, the lower delay bound δ(p)≥1.*

Proof. Suppose that $cl(G) \leq c$. By definition, $cl(G)$ is the maximum computation time among all paths with $d(e) < \max\{q_u,q_v\}$ for all edges $e=(u,v)$. Thus, for any subpath p, $\lfloor d(e) / \max\{q_u,q_v\}\rfloor = 0$ for all edges in the subpath and $\delta(p) = 0$ for all such subpaths. Hence, if p is a subpath with no internal split nodes and total computation time greater than c, there must exist at least one sufficiently large group of delays along some edge in the subpath. In other words, $T(p) > c$ implies $\delta(p)\geq 1$. On the other hand, assume that any subpath p with $\delta(p)= 0$ and without internal split nodes has computation time $T(p) \leq c$. Then $d(e) < \max\{q_u,q_v\}$ for all edges $e=(u,v)$ along p. Since $cl(G)$ is the maximum computation time of all such subpaths, we must have $cl(G) \leq c$ as well.

□

In the case of Fig. 4, this means that the clock period is indeed three due to the subpaths from A to B_L and from B_R to C.

Extended Retiming and SDFG Scheduling

Theorem 11 of (Leiserson and Saxe, 1991) establishes the equivalence of an optimal traditional retiming's existence and the absence of negative cycles in the scheduling graph for unit-time DFGs. We will now prove a similar relationship for general-time SDFGs.

Theorem 3.*Let* $G = \langle V,E,d,t\rangle$ *be a (general-time) SDFG without split nodes. Let* c *be a positive integer with* $t(v) \le c$ *for all* v*. Then there is a legal extended retiming* r *on* G *such that* $cl(G_r) \le c$ *if and only if the scheduling graph* G^s *contains no negative-weight cycle.*

Proof. Assume $cl(G_r) \le c$. By definition, the iteration bound $B(G_r)$ is less than or equal to $cl(G_r)$, and since retiming does not affect the iteration bound, $B(G) = B(G_r) \le c$. This implies that

$$\sum_{v\in\ell} t(v) \le c \cdot \sum_{e=(u,v)\in\ell} \left\lfloor \frac{d(e)}{\max\{q_u, q_v\}} \right\rfloor$$

for all cycles ℓ in G by equation (1) above, which implies that

$$\sum_{e=(u,v)\in\ell} \left(\left\lfloor \frac{d(e)}{\max\{q_u, q_v\}} \right\rfloor - \frac{t(v)}{c} \right) \ge 0.$$

Since a cycle in G is also one in G^s, we see that the sum of the edge weights is non-negative for all cycles ℓ in G^s. Therefore, G^s has no negative weight cycles.

On the other hand assume that G^s contains no negative-weight cycle. Let $X = \min_v sh(v)$ and $R(v) = c\cdot ((sh(v) - X) - \lfloor sh(v) - X \rfloor)$ for all nodes v. We now define

$$r(v) = \lfloor sh(v) - X \rfloor + \min\left\{1, \frac{R(v)}{t(v)}\right\}. \qquad (2)$$

We assert that r is a legal extended retiming. If so, based on our earlier discussion, any node split by retiming via r is divided into subnodes v_L and v_R having execution times $R(v)$ and $t(v) - R(v)$, respectively. We now want to show that r is a legal extended retiming and that $cl(G_r) \le c$.

- Choose any edge $e = (u,v)$ in G. Then

$$sh(v) \le sh(u) + \left\lfloor \frac{d(e)}{\max\{q_u, q_v\}} \right\rfloor - \frac{t(u)}{c}$$

and so

$$\begin{aligned} \lfloor sh(v) - X \rfloor &\le \lfloor sh(u) - X \rfloor \\ &\quad + \left\lfloor \left\lfloor \frac{d(e)}{\max\{q_u, q_v\}} \right\rfloor - \frac{t(u)}{c} \right\rfloor \\ &= \lfloor sh(u) - X \rfloor \\ &\quad + \left\lfloor \frac{d(e)}{\max\{q_u, q_v\}} \right\rfloor - 1 \end{aligned}$$

since $0 < t(u)/c \le 1$. By definition

$$\begin{aligned}\frac{d_r(e)}{\max\{q_u,q_v\}} &= \frac{d(e)}{\max\{q_u,q_v\}} + \lfloor r(u)\rfloor - \lceil r(v)\rceil \\ &\geq \left\lfloor\frac{d(e)}{\max\{q_u,q_v\}}\right\rfloor + \lfloor sh(u) - X\rfloor \\ &\quad -\lfloor sh(v) - X\rfloor - 1 \\ &\geq 0.\end{aligned}$$

Thus $d_r(e) \geq 0$ and r is a legal extended retiming by definition.

- Consider a subpath p in G_r with end nodes u and v, no internal split nodes and whose computation time is larger than c. We wish to show that the lower delay bound of p is greater than or equal to 1. There are now nine possibilities for p, five of which clearly have delay counts greater than 1 since p passes through a delay contained within a split end node:
    - u is split, and p is one of $u_L\to\ldots\to v$ (for v not split), $u_L\to\ldots\to v_R$ or $u_L\to\ldots\to v_L$ (for v split);
    - or$v$ is split, and p is either $u\to\ldots\to v_R$ (for u not split) or $u_R\to\ldots\to v_R$ (for u split).

  We now deal individually with the remaining four cases.

  Case 1. Suppose $p : u\to\ldots\to v$. In this case neither u nor v is split so $r(u) = sh(u) - X$ and $r(v) = sh(v) - X$ are both integers, making floor and ceiling functions unnecessary in the definition of the path's delay count. It is easy to see that

$$\begin{aligned} sh(v) &\leq sh(u) \\ &\quad + \sum_{e=(w,z)\in p}\left(\left\lfloor\frac{d(e)}{\max\{q_w,q_z\}}\right\rfloor - \frac{t(w)}{c}\right) \\ &\leq sh(u) - \frac{T(p)-t(v)}{c} \\ &\quad + \sum_{e=(w,z)\in p}\left\lfloor\frac{d(e)}{\max\{q_w,q_z\}}\right\rfloor.\end{aligned}$$

Thus

$$\begin{aligned}\delta_r(p) &= \sum_{e=(w,z)\in p}\left\lfloor\frac{d_r(e)}{\max\{q_w,q_z\}}\right\rfloor \\ &= \sum_{e=(w,z)\in p}\left\lfloor\frac{d(e)}{\max\{q_w,q_z\}} + \lfloor r(w)\rfloor - \lceil r(z)\rceil\right\rfloor \\ &\geq \sum_{e=(w,z)\in p}\lfloor\lfloor r(w)\rfloor - \lceil r(z)\rceil\rfloor \\ &\quad + \sum_{e=(w,z)\in p}\left\lfloor\frac{d(e)}{\max\{q_w,q_z\}}\right\rfloor.\end{aligned}$$

Now

$$\begin{aligned}&\sum_{e=(w,z)\in p}\lfloor\lfloor r(w)\rfloor - \lceil r(z)\rceil\rfloor \\ &\quad = \lfloor r(u)\rfloor + \sum_{\substack{w\in p\\ w\neq u,v}}(-\lceil r(w)\rceil + \lfloor r(w)\rfloor) - \lceil r(v)\rceil \\ &\quad = \lfloor r(u)\rfloor - \lceil r(v)\rceil\end{aligned}$$

since none of the internal nodes of p are split. Continuing from there

$$\begin{aligned}\delta_r(p) &\geq r(u) - r(v) + \sum_{e=(w,z)\in p} \left\lfloor \frac{d(e)}{\max\{q_w, q_z\}} \right\rfloor \\ &= sh(u) - sh(v) + \sum_{e=(w,z)\in p} \left\lfloor \frac{d(e)}{\max\{q_w, q_z\}} \right\rfloor \\ &\geq \frac{T(p) - t(v)}{c} > 0.\end{aligned}$$

Since the retimed delay count is an integer that is greater than or equal to a *strictly* positive fraction, we conclude that $\delta_r(p) \geq 1$.

Case 2. Suppose p is the path $u \rightarrow \ldots \rightarrow v_L$. Then p is part of some larger path $u \rightarrow \ldots \rightarrow v$ denoted p^*. Note that

1. $T(p) = T(p^*) - t(v_R) > c$ by assumption;
2. $\delta_r(p) = \delta_r(p^*)$ since both the path and the subpath share the same edges; and
3. Finally since v is split

$$\begin{aligned} t(v_L) &= t(v) \cdot (r(v) - \lfloor r(v) \rfloor) \\ &= t(v) \cdot \left(\lfloor sh(v) - X \rfloor + \frac{R(v)}{t(v)} - \lfloor sh(v) - X \rfloor \right) \\ &= c \cdot \big((sh(v) - X) - \lfloor sh(v) - X \rfloor \big) \end{aligned}$$

Now, since u is not split

$$\begin{aligned}\delta_r(p) &\geq \lfloor r(u) \rfloor - \lceil r(v) \rceil + \sum_{e=(w,z)\in p} \left\lfloor \frac{d(e)}{\max\{q_w, q_z\}} \right\rfloor \\ &= (sh(u) - X) - (\lfloor sh(v) - X \rfloor + 1) \\ &\quad + \sum_{e=(w,z)\in p} \left\lfloor \frac{d(e)}{\max\{q_w, q_z\}} \right\rfloor \\ &= (sh(u) - sh(v)) + \sum_{e=(w,z)\in p} \left\lfloor \frac{d(e)}{\max\{q_w, q_z\}} \right\rfloor \\ &\quad + (sh(v) - X) - (\lfloor sh(v) - X \rfloor + 1) \\ &\geq \frac{T(p^*) - t(v)}{c} + (sh(v) - X) - \lfloor sh(v) - X \rfloor - 1 \\ &= \frac{T(p^*) - t(v_R)}{c} - \frac{t(v_L)}{c} + \frac{t(v_L)}{c} - 1 \\ &> 0 \end{aligned}$$

as above and from the definition of $t(v_L)$. As before we conclude that $\delta_r(p) \geq 1$.

Case 3. Suppose p is the path $u_R \rightarrow \ldots \rightarrow v$. This time v is not split and again p is part of some larger path $u \rightarrow \ldots \rightarrow v$ called p^* with $T(p^*) - t(u_L) > c$, so as before

$$\begin{aligned}
\delta_r(p) &\geq \lfloor r(u)\rfloor - \lceil r(v)\rceil \\
&\quad + \sum_{e=(w,z)\in p} \left\lfloor \frac{d(e)}{\max\{q_w, q_z\}} \right\rfloor \\
&= \lfloor sh(u) - X\rfloor - (sh(v) - X) \\
&\quad + \sum_{e=(w,z)\in p} \left\lfloor \frac{d(e)}{\max\{q_w, q_z\}} \right\rfloor \\
&= (sh(u) - sh(v)) \\
&\quad + \sum_{e=(w,z)\in p} \left\lfloor \frac{d(e)}{\max\{q_w, q_z\}} \right\rfloor \\
&\quad + \lfloor sh(u) - X\rfloor - (sh(u) - X) \\
&= \frac{T(p^*) - t(v)}{c} - \frac{t(u_L)}{c} \\
&> 1 - \frac{t(v)}{c} \geq 0
\end{aligned}$$

since $0 < t(v) \leq c$ by assumption, so $\delta_r(p) \geq 1$.

Case 4. Finally suppose $p : u_R \Rightarrow v_L$. Both end nodes are split and $T(p^*) - t(u_L) - t(v_R) > c$. As above

$$\begin{aligned}
\delta_r(p) &\geq \lfloor r(u)\rfloor - \lceil r(v)\rceil + \sum_{e=(w,z)\in p} \left\lfloor \frac{d(e)}{\max\{q_w, q_z\}} \right\rfloor \\
&= \lfloor sh(u) - X\rfloor - \lfloor sh(v) - X\rfloor - 1 \\
&\quad + \sum_{e=(w,z)\in p} \left\lfloor \frac{d(e)}{\max\{q_w, q_z\}} \right\rfloor \\
&= (sh(u) - sh(v)) + \sum_{e=(w,z)\in p} \left\lfloor \frac{d(e)}{\max\{q_w, q_z\}} \right\rfloor \\
&\quad + (\lfloor sh(u) - X\rfloor - (sh(u) - X)) \\
&\quad - (\lfloor sh(v) - X\rfloor - (sh(v) - X)) - 1 \\
&\geq \frac{T(p^*) - t(v)}{c} + \frac{t(v_L)}{c} - \frac{t(u_L)}{c} - 1 \\
&= \frac{T(p^*) - t(u_L)}{c} + \frac{t(v_L) - t(v)}{c} - 1 \\
&= \frac{T(p^*) - t(u_L) - t(v_R)}{c} - 1 > 0
\end{aligned}$$

and again $\delta_r(p) \geq 1$.

In any case, the lower delay bound of the retimed subpath is larger than one, and by Lemma 2, $cl(G_r) \leq c$.

□

Note that extended retiming is only the first part of the overall scheduling process. Once a graph is retimed, DAG scheduling must be applied to the altered graph to compute start times for execution of the nodes. Let us now summarize what we have proven so far:

Theorem 4. *Let Gbe a SDFG andcan integer. The following statements are equivalent:*

1. *There is a legal extended retiming r on G such that $cl(G_r) \leq c$.*

2. *There exists a legal, integral, repeating, static schedule for G under the non-pipelined implementation with cycle period c.*
3. *The scheduling graph G^sassuming cycle period $c$contains no cycle having negative delay count and$t(v) \leq c$for all nodes $v$of G.*
4. *The iteration bound$B(G) \leq c$and$t(v) \leq c$for all nodes $v$of G.*

Proof. Theorem 3 establishes the equivalence of (1) and (3). The equivalence of (3) and (4)is proven in Lemma 3.1 of (O'Neil *et al.*, 2011). Finally, the equivalence of (4) and (2) is proven in Theorem 3.3 of (O'Neil *et al.*, 2011).

□

See that we have also developed a more efficient algorithm for checking the legality of a clock period. Traditionally, it has taken $O(\ |V|\ \ |E|\ )$ time to see if we have a legal retiming r such that $cl(G_r) \leq c$ for a given c. Now, we simply verify that the known values of $B(G)$ and the maximum $t(v)$ are all smaller than our chosen c and can know almost immediately if there's such an extended retiming. This theorem also shows that extended retiming can produce a clock period as small as the cycle period obtained by the scheduling algorithm of (O'Neil *et al*, 2011). Therefore, that scheduling algorithm is simply one specific case of this new extended-retime-then-DAG-schedule method, as stated above.

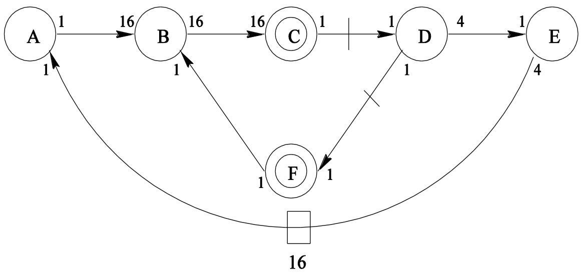
EXPERIMENTS WITH COMMON FILTERS

As an overview of everything discussed to this point, we now wish to demonstrate our method and its usefulness with some more interesting examples involving well known filters.

Detailed Multi-Rate Example: A Simplified Spectrum Analyzer

As an interesting quickly-reviewable example, let us apply our algorithms to a variation of the spectrum analyzer G from (Zivojnovic *et al.*, 1994a) which appears in Fig. 5(a), with node descriptions in Fig. 5(b). This graph has a BRV of q = [16 1 1 1 4 1]^T, so in the interest of space we will not display most of the intermediate figures at each step. Instead, we shall describe the pertinent information.

In Fig. 5(a) there are two loops. The path $B \rightarrow C \rightarrow D \rightarrow F \rightarrow B$ gives a time-to-delay ratio of 6/2 or 3 for smaller of the two. The other loop includes the path $A \rightarrow B \rightarrow C \rightarrow D \rightarrow E \rightarrow A$. The only edges with delays are (C,D), which requires no adjustment, and (E,A), whose adjusted delay count for our calculation is 1. Thus the time-to-delay ratio for this loop is also 3. In any case, $B(G)$ = 3 and three serves as a lower bound on our iteration period and will be our target when we schedule.



(a)

Node	**Description**	**Time**
A	Adaptive Low-Pass	1
B	FFT Zoom	1
C	Peak Detector	2
D	Interpolator	1
E	Decision	1
F	Zoom Control	2

(b)

Fig. 5: A simplified spectrum analyzer

When constructing the corresponding scheduling graph, we find that $w((D, F)) = w((E,A)) = ⅔$, $w((C,D)) = ⅓$, $w((F,B)) = − ⅔$ and $w((A,B)) = w((B,C)) = w((D,E)) = − ⅓$. The shortest path lengths are then $sh(A) = sh(F) = 0$, $sh(B) = sh(D) = − ⅔$ and $sh(C) = sh(E) = −1$.

The zero-delay path $F \rightarrow B \rightarrow C$ which prevents our CPU from operating at any speed slower than five clock ticks. We thus now apply retiming to attempt to reduce the length of this longest zero-delay path to match better the iteration bound. Substituting our shortest path lengths into equation (2) yields the function $r(A) = r(B) = r(D) = r(F) = 0$ and $r(C) = r(E) = 1$. Applying this retiming function produces the retimed analyzer of Fig. 6, which has longest zero-delay paths of length 3 and so is optimal.

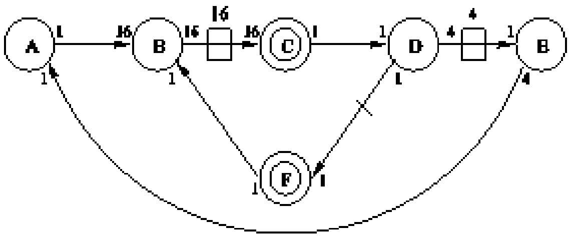


Fig. 6: The retimed analyzer

Detailed Homogeneous Examples

To further establish the benefit of extended retiming, we consider multiplevariations on familiar filters from (Kung *et al.*, 1985). As a detailed illustration, consider the 2-cascaded biquad filter pictured in Fig. 7 below. It is formed by mapping the output of one infinite impulse response (IIR) filter to the input of another. As above, square nodes represent adders while circles represent multipliers. As one can observe, there are two families of cycles in this DFG. The first ({*A*, *B*, *D*} and {*J*, *K*, *M*}) contain two adders and a multiplier with a delay count of 2. The other ({*A*, *C*, *D*} and {*J*, *L*, *M*}) includes the same hardware but with only one total delay. If *A* is the cost of addition and *M* that of multiplication, we can see that the iteration bound of this filter is 2*A*+*M*.

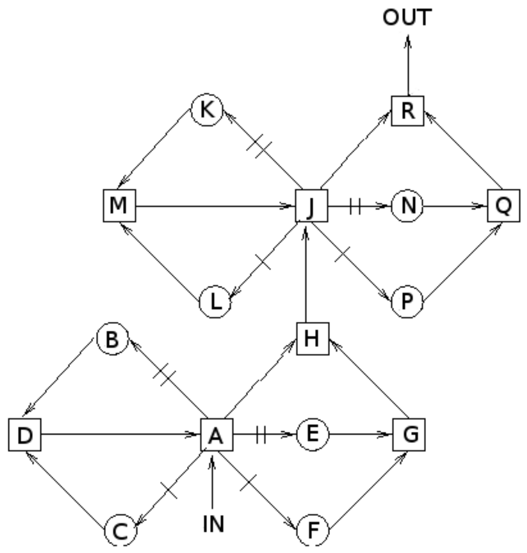


Fig. 7: The 2-cascaded biquad filter

To create more interesting variations on this example, we will multiply the register count of each edge by a factor of *S*.This is referred to in (Leiserson and Saxe, 1991) as applying a *slowdown* of *S* to our original circuit. The new iteration bound of the *S*-slow filter would then be 2*A*+*M* / *S*.

If we now let *A* = *M* = *S* = 2, we see that the iteration bound equals 3. It can be shown by the retiming algorithm of (Leiserson and Saxe, 1991) that traditional retiming reduces the clock period of this filter only to 4. However, our method produces a retiming function of *r*(*A*) = *r*(*L*) = *r*(*P*) = 2, *r*(*B*) = *r*(*C*) = *r*(*E*) = *r*(*F*) = *r*(*K*) = *r*(*N*) =3½, *r*(*D*) = *r*(*G*) = 3, *r*(*H*) = *r*(*M*) = *r*(*Q*) = 1½, *r*(*J*) = 1 and *r*(*R*) = 0. Applying this to the filter, we derive the DFG in Fig. 8 with clock period 3.

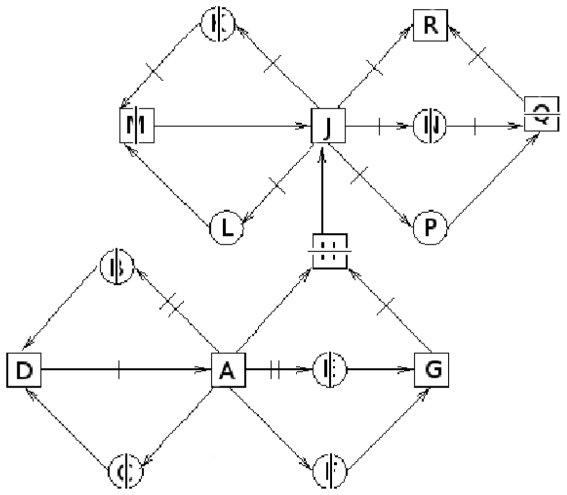


Fig. 8: The retimed 2-slow 2-cascaded biquad filter with *A* = *M* = 2.

To further establish the benefit of extended retiming, we repeat this experiment. The first considered filter, the all-pole lattice filter, is used in voice synthesis. The second, the fifth order elliptic filter, is also common in signal processing. Using our previous notation, the *S*-slow versions of these filters have iteration bounds of $4A + 2M / S$ and $10A + 3M / S$, respectively.

For our trials, one of these three filters was selected and addition allowed to range in cost from 1 to 29. Similarly, slowdowns between 1 and 20 were considered. The cost of multiplication ranged from the cost of addition up to 30. These values of *A*, *M* and *S* give us 3,959,950 different variations for each of the selected filters. We compute the iteration bounds for each trial and apply the ceiling function to derive an optimal clock period. From here, a trial was rejected if this optimal clock period was smaller than the cost of multiplication. Finally, the retiming algorithm of (Leiserson and Saxe, 1991) was applied to each acceptable filter variation to derive the best clock period achievable by traditional retiming.

We summarize the results of this experiment in Table 1 below. Here, a trial is classified as an "improvement" if traditional retiming could not achieve the target clock period while extended retiming could. Overall, extended retiming yielded a superior result (i.e. a smaller clock period) to traditional retiming in 62 percent of the circuits examined. The largest difference was found in the 7-slow elliptic filter with costs of 29 and 30 for addition and multiplication, respectively. Here extended retiming delivers a circuit with a minimal clock period of 55, while traditional retiming can achieve no better than a clock of 205.

Table 1: Summary of Experimental Results.

	Total Cases	Improvements	Percent
2-Cascaded Biquad Filter	810	224	28
All-Pole Lattice Filter	1822	678	37
Fifth Order Elliptic Filter	3905	3126	80

A representative sample of our experiments appears in
Table 2. In all cases cited, extended retiming gives us a better result than does traditional retiming.

Table 2: Minimal achievable clock periods for different circuits.

	Computation Time				Min. Clock Period	
Benchmark	Add	Mult	Slow-down	Iter. Bound	Ext.	Trad.
2-Cascaded Biquad	3	4	2	5	5	7
2-Cascaded Biquad	4	6	2	7	7	10
All-Pole Lattice	2	2	4	3	3	6
All-Pole Lattice	3	4	4	5	5	10
Fifth Order Elliptic	2	2	2	13	13	14
Fifth Order Elliptic	3	4	2	21	21	22

CONCLUSION

We have seen that our new method of extended retiming permits us to transform any data-flow graph to one whose clock period matches the cycle period of any of its legal schedules. We have demonstrated conditions under which an integer is a permissible choice for either the clock or cycle period of a retimed SDFG. To wit, it must be an upper bound on both the computation times of all nodes and the graph's iteration bound of the graph.

Throughout this paper, we have made simplifying assumptions, as do(Parhi and Messerschmitt, 1991), (Leiserson and Saxe, 1991), (Chao and Sha, 1997).This allowed us to build up a rigorous body of results before refining it to deal with real-world complications. Taking unconstrained resources for granted is the most notable of these. As seen in (O'Neil, 2012), it appears possible to apply our methods in a more restrictive environment, but must continue to develop this line of inquiry.

Furthermore, we have assumed the use of integral schedules under a non-pipelined implementation. The possibilities of *fractional* schedules, where operations may appear at any time (not necessarily at specific points), and *pipelined* implementations (Chao and Sha, 1995) remain. Combinations of these four parameters (integral or fractional DFG scheduling, pipelined or non-pipelined implementation) give us three additional models to explore as we discuss extended retiming.

The non-pipelining assumption is particularly crucial. The main result of this work depends on the binding of SDFG node execution times from above by the desired clock period. This restrictionreduces the usefulness of this methodin the general case. Indeed, by considering a 2-slow 2-cascaded biquad filter with A = 1 and M = 4, we can see the limitations. Of course, one can get around this restriction by using unfolding in addition to extended retiming, but removing this complication for a better stand-alone method remains a challenge.

Finally, we have also assumed the use of a very particular multi-rate data-flow model, the most straightforward one. Applications of this method to data-flow models which vary the execution times of nodes according to a probability distribution (Tongsima *et al.*, 2000) remain to be explored. Similarly, the consideration of multi-dimensional variations of these models (Passos, 1996) constitutes future work.

REFERENCES

Chao, L.F. (1993), "Scheduling and behavioral transformations for parallel systems", Ph.D. thesis, Princeton University.

Chao, L.-F. and Sha, E. H.-M. (1997), "Scheduling data-flow graphs via retiming and unfolding", *IEEE Transactions on Parallel and Distributed Systems*, Volume 8, pp. 1259 - 1267.

Chao, L.-F. and Sha, E. H.-M. (1995), "Static Scheduling for Synthesis of DSP Algorithms on Various Models", *Journal of VLSI Signal Processing*, Volume 10, pp. 207 - 223.

Cormen, T.H., Leiserson, C.E., Rivest, R.L. and Stein, C. (2009), *Introduction to algorithms*, The MIT Press, Cambridge MA.

Kung, S., Whitehouse, J. and Kailath, T. (1985),*VLSI and Modern Signal Processing*, Prentice Hall, Englewood Cliffs NJ.

Lee, E.A. and Messerschmitt, D.G. (1987), "Static scheduling of synchronous data-flow programs for digital signal processing", *IEEE Transactions on Computers* Volume 36, pp. 24 - 35.

Leiserson, C.E. and Saxe, J.B. (1991), "Retiming synchronous circuitry", *Algorithmica*, Volume 6, pp. 5 - 35.

O'Neil, T.W. (2014), "Rate-optimal unfolding of synchronous data-flow graphs", in *Proc.2014 ISCA International Conference on Computers and Their Applications*, pp. 261 - 266.

O'Neil, T.W. (2012), "Static scheduling of synchronous data-flow graphs under resource constraints", *Journal of Parallel and Distributed Computing and Networks*, DOI: *10/2316/Journal.211.2012.1.211-1057*.

O'Neil, T.W. (2011), "Unfolding synchronous data-flow graphs", in *Proc.2011 IASTED International Conference on Parallel and Distributed Computing and Sytems*, pp. 278 - 283.

O'Neil, T. W. and Sha, E. H.-M. (2005), "Combining extended retiming and unfolding for rate-optimal graph transformation", *Journal of VLSI Signal Processing*, Volume 39, pp. 273 - 293.

O'Neil, T.W. and Sha, E. H.-M. (2001), "Retiming synchronous data-flow graphs to minimize execution time", *IEEE Transactions on Signal Processing*, Volume 49, pp. 2397 - 2407.

O'Neil, T.W., Khasawneh, S.F., Richter, M.E. and Pullaguntla, R.K. (2011), "Transforming synchronous data-flow graphs to reduce execution time", *International Journal of Computers and Their Applications*, Volume 18, pp. 111 - 122.

O'Neil, T.W., Tongsima, S. and Sha, E. H.-M. (1999), "Extended retiming: optimal retiming via a graph-theoretical approach", in *Proc.1999 IEEE International Conference on Acoustics, Speech and Signal Processing*, Volume 4, pp. 2001 - 2004.

O'Neil, T. W., Tongsima, S. and Sha, E. H.-M. (1999), "Optimal scheduling of data-flow graphs using extended retiming", in *Proc. ISCA 12th International Conference on Parallel and Distributed Computing Systems*, pp. 292 - 297.

Parhi, K.K. and Messerschmitt, D.G. (1991), "Static rate-optimal scheduling of iterative data-flow programs via optimum unfolding", *IEEE Transactions on Computers*, Volume 40, pp. 178 - 195.

Passos, N. (1996), "Improving parallelism on multi-dimensional applications: the multi-dimensional framework", Ph.D. thesis, The University of Notre Dame.

Renfors, M. and Neuvo, Y. (1981), "The maximum sampling rate of digital filters under hardware speed", *Transactions on Circuits and Sampling*, Volume CAS-28, pp. 196 - 202.

Tongsima, S., Sha, E. H.-M., Chantrapornchai, C., Surma, D. and Passos, N. (2000), "Probabilistic loop scheduling for applications with uncertain execution time", *IEEE Transactions on Computers*, Volume 49, pp. 65 - 80

Zhuge, Q., Xue, C., Shao, Z., Qiu, M. and Sha, E. H.-M. (2006), "Design optimization and space minimization considering timing and code size via retiming and unfolding", *Microprocessors and Microsystems*, Volume 30, pp. 173 - 183.

Zivojnovic, V., Ritz, S. and Meyr, H. (1994), "Optimizing DSP programs under the multirate retiming transformation", in *Proc. 7th European Signal Processing Conference*, Volume 3, pp. 1597 - 1600.

Zivojnovic, V., Ritz, S. and Meyr, H. (1994), "Retiming of DSP programs for optimum vectorization", in *Proc. IEEE International Conference on Acoustics, Speech and Signal Processing*, Volume 1, pp. 465 - 468.

--0--