Optimizing Resource Utilization Using Transformations

Miodrag Potkonjak, *Member, IEEE,* and Jan Rabaey, *Senior Member, IEEE*

***Abstract*—The goal of the high level synthesis process for real time applications is to minimize the implementation cost, while still satisfying all timing constraints. In this paper, we present how a combination of four conceptually simple, yet powerful, transformations: namely retiming, associativity, commutativity and inverse element law, can help to further this goal.**

Since the minimization problem associated with those transformations is NP complete, a new fast iterative improvement probabilistic algorithm has been developed. The effectiveness of the proposed algorithm and the associated transformations is demonstrated in multiple ways: using standard benchmark examples, with the aid of statistical analysis and through a comparison with estimated minimal bounds.

I. Introduction

1.1. Motivation

THE GOAL OF THE HIGH LEVEL synthesis process for application specific integrated circuits is to translate the specification of the algorithm (defined in terms of its behavioral semantics as well as its performance requirements) into architectural primitives (an interconnection of execution units, memory and control units) in such a way that the resulting silicon implementation minimizes a certain function. Most often this function is either the area and/or the throughput.

The essential high level synthesis tasks are module and clock selection, scheduling, assignment and allocation. However, even when the highest quality algorithms are used for those tasks, the quality of a final result is often constrained by the computational structure of the specified algorithm.

This is illustrated using the example of Fig. 1(a). This figure shows a direct form II biquadratic section often used in the realization of IIR filters. Assume that at most four clock cycles are available for the execution of this flow graph and that both multiplication and addition take a single clock cycle. Inspection of the computational graph reveals that its critical path equals four clock cycles as well. A potential schedule for this filter, requiring a minimal amount of hardware (for the sake of simplicity we will address only execution units in this example), is shown in Table I. Regardless of the scheduling technique used, we need at least two multipliers, since the graph contains four multiplications and no multiplications can be scheduled in the last control cycle. Also, since no addition can be executed in the first control cycle, two adders are needed. This can be verified with the aid of Fig. 2, which plots the maximum parallelism available in the graph over time (in terms of the number of additions and multiplications which can be simultaneously executed in a given control step).

TABLE I
POSSIBLE BIQUADRATIC FILTER SCHEDULE BEFORE AND AFTER RETIMING

	BEFORE		AFTER	
CYCLES	Multipliers	Adders	Multipliers	Adders
1	1, 3	-	2	8
2	2, 4	5	3	6
3	-	6, 8	1	7
4	-	7	4	5

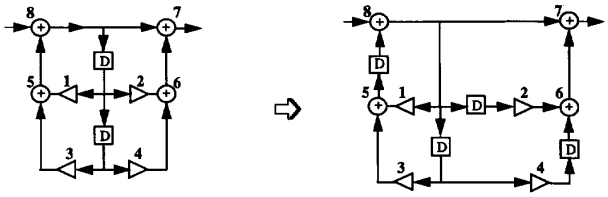

Fig. 1. Biquadratic filter: (a) initial structure and (b) transformed structure.

The resource utilization is obviously not equally balanced over time. If we define the resource utilization as the ratio of the number of cycles a resource is exploited over the total number of available cycles, then the resource utilization for adders and multipliers in this example is maximally 50%. This is an indication of a relatively low quality solution, in this case caused by the inherent structure of the computational graph.

Transformations are the best way to overcome these resource utilization bounds. Three transformations that are particularly effective in achieving this goal are retiming, associativity, and commutativity as presented in Fig. 3.

1.2. Retiming, Associativity, Commutativity and Inverse Element Law

- *Retiming* (Fig. 3(a)) uses the distributivity property of the delay operator over most other operators. In other words, when D is defined as the delay operator, the statement $D(a) * D(b)$ is equivalent to $D(a * b)$ and vice versa (with * an arbitrary operator).

Manuscript received December 20, 1991; revised September 6, 1992. A preliminary version of this paper was presented at the IEEE International Conference on Computer-Aided Design ICCAD-91. This paper was recommended by Associate Editor A. Parker.

M. Potkonjak was with the Department of EECS, University of California-Berkeley. He is now with C&C Research Laboratories, NEC USA, Princeton, NJ 08540.

J. Rabaey is with the Department of EECS, University of California-Berkeley, Berkeley, CA 94720.

IEEE Log Number 9214062.

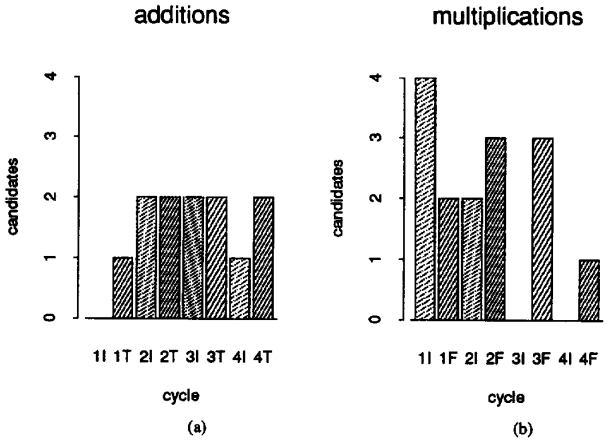

Fig. 2. Available parallelism (maximum number of operations which can be scheduled in a given control cycle simultaneously before (a) and after (b) retiming for resource utilization.

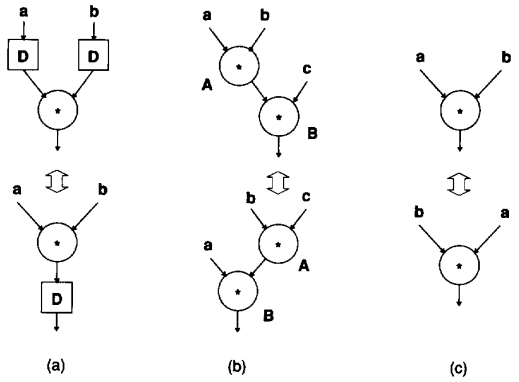

Fig. 3. Basic (a) retiming, (b) associativity and (c) commutativity moves.

Consider now the flow graph, shown in Fig. 1(b), obtained by moving the delays (retiming) in the original flow graph of Fig. 1(a). It is easy to check that the resulting graph has the same input/output relationship as the original graph. The available parallelism is plotted in Fig. 2(b). It is obvious that the resource utilization is far more balanced and a solution with one multiplier and one adder can now be achieved as shown in Table I. The resource utilization for the execution units now equals 100%.

- ☐ *Associativity* (Fig. 3(b)) is a basic property of many algebraic structures, starting from group to vector spaces and matrix algebra [81]. Associativity postulates that, in the set over an algebraic structure defined using an operation *, for every a, b, and c, which are elements of the set, it holds that a * (b * c) = (a * b) * c.
- ☐ *Commutativity* (Fig. 3(c)) is a well known property of many algebraic structures [81]. Commutativity states that, in the set over an algebraic structure defined using an operation *, for every a, and b which are elements of the set, it holds that $a * b = b * a$.

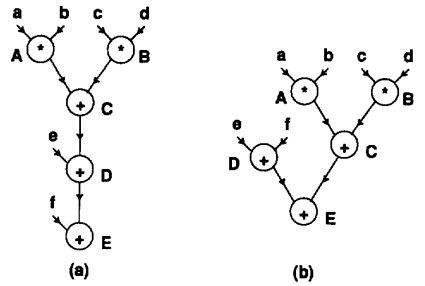

Fig. 4. Applying associativity to improve resource utilization: Flow graph (a) before and (b) after.

- ☐ *The Inverse Element Law* is also a common basic property of many algebraic structures [81]. This law expresses that, if a is in the set, then there is some element b in the set called an inverse of a such that $a * b = b * a = e$. The element e is called the identity element and satisfies for every a in the set the relationship $a * e = e * a = a$. We will use both commutativity and the inverse element law to enhance the power of associativity as discussed in Section 3.1.

A simple application of associativity is demonstrated in Fig. 4. Implementing the control data flow graph (CDFG) of Fig. 4(a) in four cycles (with both additions and multiplications taking one cycle) would obviously require two multipliers and one adder. However, after applying associativity on the adder chain (Fig. 4(b)), the critical path is reduced to three cycles and a single multiplier and adder are sufficient for the implementation. This example illustrates the most common use of associativity, namely for the so called *tree height reduction*, although it can contribute in other ways to improving the resource utilization, as will be discussed in Section 3.2.

The four mentioned transformations are not orthogonal. Their full power can only be exploited when applying them simultaneously. This statement will be discussed in more detail in Section 3.2. Some other resource utilization improving transformations (such as distributivity) and their incorporation in the proposed framework will be analyzed there as well.

The above examples illustrate that while transformations are not necessary for a bare minimum high level synthesis system, they are essential when trying to achieve a high quality solution or sometimes even just a feasible solution.

1.3. Previous Work

1.3.1. Transformations In General: The current state-of-the-art in research on transformations, as well as the closely related scheduling, can be traced in several areas: operation research, software compilers and in particular software compilers for parallel computers, high level synthesis and more globally CAD, DSP ASIC, and systolic array design, theoretical computer science, and numerical analysis and algebra.

Transformations are not discussed in operations research scheduling theory or practice, due to the different nature of the elementary task and the task dependencies [62]. While it is obvious that some transformations, e.g., associativity, are not applicable, it is very likely that operation research job

scheduling systems can benefit from, for instance, the usage of retiming.

Optimizing software compilers have been the subject of extensive and impressive research effort in the last two decades. Almost any compiler book contains a chapter or two on optimizing transformations [2], [21].

Among the most successful recent research projects are SUIF [55], [83] and Sharlit [78] at Stanford University, Superb at the University of Bonn [86], Parafrase [41], Parafrase 2 [70], Delta [64] and Impact [52] at the University of Illinois, Urbana Champaign, PFC [5], PTOOL [6] and ParaScope [37] at Rice university, Crystal at Yale University [13], PAT at Georgia Institute of Technology [76], PTRAN at IBM [3], and TINY Tool at the Oregon Graduate Institute [84]. Recent software compilers emphasize the use of loop transformations and background memory optimizing transformations. The latter is due to a the technological gap between the speed of processors and memory. The optimization algorithms used are usually simple and fast, because the application programs are large, and compilation time is often as important as run time.

In recent years, transformations have been getting increasing attention in CAD, mainly in logic and high level synthesis. Several logic synthesis systems are based on an extensive use of logic level transformations [14], [35], [28].

The use of transformations in high level synthesis was first proposed in the late seventies [77], [11], [60], [56]. The most comprehensive sets of transformations in high level synthesis are given in [79], [82], [31], [8], [73].

Flamel [79] uses five block level transformations (essentially two variants of loop merging and three forms of loop unrolling) followed by a greedy application of height reduction and constant folding. The goal is the minimization of the execution time under area constraints, which is a significantly simpler task than the problem of optimizing the area with the time as a constraint. The reason for this is that in the former formulation the minimization of each block run-time produces a globally minimal implementation time. The optimization task is hence easily divided into a number of simpler subtasks. Although the optimization strategy used is simple and at least partially greedy and the relationship between control and algebraic transformations is ignored, excellent improvements were achieved on 15 small examples.

The System Architect's Workbench (SAW) [82] uses inline expansion, dead code elimination, four types of selects (if-then-else transformations where the code is moved across boundaries imposed by the control structure) as well as pipelining, mainly to support behavioral and structural partitioning. SPAID [30] proposes the use of retiming as an independent transformation (as well as pipelining, which is a special case of retiming), interleaving, the expansion of multiplications into add/shift operations and algebraic transformations. Both SAW and SPAID are interactive frameworks, where the designer manually explores design trade-offs using a predefined transformation mechanisms.

Pipelining is by far the most often applied transformation in high level synthesis [67]–[69]. Hartley [31] combines pipelining and tree-high reduction (a special case of associativity), but his technique is applicable only to examples without feedback. Although pipelining is very powerful, it is not a transformation in the strict sense, since it increases algorithm latency. Its application domain is also often limited to nonrecursive algorithms [58].

Loop folding was first introduced by Girzyc [29]. Numerous synthesis systems combine a few transformations (most often loop unrolling, software pipelining and pipelining) with scheduling. For example, Devadas [16] combines scheduling with dynamic partial unwinding (unrolling) and functional pipelining with allocation, assignment, and scheduling using a simulated annealing algorithm. Goossens successfully treats both single and nested loop folding (software pipelining) during scheduling [27], while Hwang combines scheduling with functional pipelining and loop winding [33].

Besides the use of transformations in the synthesis of computationally intensive, data path dominated architectures and micro-processors, recent results have indicated that they also can be successfully applied in the design of control dominated machines. [85].

Transformations in general, and retiming and associativity in particular, are topics often discussed in the digital signal processing and systolic arrays literature, either in the context of the development of new algorithms or the mapping of an algorithm onto a particular architecture [10], [43]. Transformations are applied manually on application instances and often lead to impressive performance improvements. Classic example of this is the transformation of the DFT into the FFT algorithm. In a more recent development, pipeline interleaving is often used to improve the resource utilization in systolic arrays [43]. A recently popular topic is the manual application of a series of simple transformations (including retiming and associativity) to reduce the pipeline stage length in recursive algorithms, such as Viterbi coding and recursive filters [58], [65], [66], [20]. Liu applied retiming and interleaving to a recursive filter and showed how interleaving creates extra delays in the feedback loops, which can be retimed to reduce the critical paths [49].

Similarly, in numerical analysis and algebra, associativity is often used during the development of efficient algorithms, either manually or in systems for algebraic manipulations such as MACSYMA [51] and REDUCE [32]. Both the digital signal processing and the numerical analysis literature also discusses the effect of transformations on numerical stability and data representation [25], [24].

1.3.2. Retiming, Associativity, Commutativity and Inverse Element Law: Retiming has been successfully used in several areas of design synthesis and automation. Until recently, it has been exclusively used either to reduce the critical path in a circuit or graph [47], [48], to minimize the number of delays [47], [26] or to optimize sequential networks with the aid of combinational logic tools by temporarily moving delays to a periphery of a network [53].

In the area of logic synthesis, Bartlett presented a transformational approach for controllers of multi-phase CMOS logic that moves logic gates across latches to minimize the delay [7]. De Micheli demonstrated how circuits equation and register positions can be optimized simultaneously with the aid of a set of algorithms based on logic transformations

[15]. Dey combined partitioning, retiming and resynthesis using combinational optimization during the optimization of sequential logic [18]. Shenoy developed an algorithm for the retiming of single phase sequential circuits with level sensitive latches [75]. Ishii investigated strategies for reducing the clock period of multi-phase, level-clocked circuits by using clock tuning and retiming [34]. Recently, Bhatt reported the interesting application of retiming in the design for testability for sequential circuits [9].

As mentioned above, retiming has been used in a number of high level synthesis environments. One particular instance worth mentioning is the Hi-PASS system for DSP architecture synthesis, which efficiently uses bit level retiming in designs with no hardware sharing [19].

While associativity got only minor treatment in the software compiler literature after the conclusion that it does not improve code as much as some other transformations [40], it was successfully used in high level synthesis and theoretical computer science for critical path reduction. Most often it is applied in conjunction with other algebraic transformations such as commutativity and distributivity [36], [59], [61], [50], [80]. We are not aware of any reference in the high level synthesis literature which discusses the inverse element law.

1.3.3. Relationship to Other Transformations: While virtually all previous applications of retiming and associativity focused on the minimization of the critical path, this paper addresses their use for the minimization of the implementation area (or equivalently the resource utilization). An obvious question which can be posed is how retiming for resource utilization transformation relates to the well known *retiming for critical path* problem, as formulated by Leiserson [47]. Since a shorter critical path most often means a less constrained graph for a given execution time, a general correlation between the length of the critical path and the cost of the implementation can be observed. The critical path however is only one of the multiple ways of reducing the stress in the graph. For the majority of the examples, discussed in Section IV, the best solution was *not* the one with the smallest critical path.

While the retiming transformation is closely related to *pipelining*, it has some distinctive advantages: first of all, it does not change the latency of the algorithm. Secondly, retiming can be used in cases where pipelining is totally ineffective, namely in the optimization of recursive structures (such as IIR filters and recursive loops). It is however worth noting that the presented approach can be easily extended to cover *pipelining for resource utilization* as well.

Two other transformations, closely related to associativity, might also be employed to affect the resource utilization. The *distributivity* move could easily be incorporated in the presented framework. However, due to the fact that distributivity changes the number of nodes in the flow graph, some reformulation of the objective function might be necessary. The *identity element law* [81] also always changes number of operations.

Although the program performance indicates that even more complex transformations can be incorporated in the same transformation framework, it is our conviction that a better approach is to treat them separately. First of all, all transformations which operate on higher levels of hierarchy (such as loop unrolling and jamming) assume a mechanism to handle hierarchy. Operations on the sub-operator level (bit-level retiming, transformations from multiplications into add/shifts) need different objective functions.

1.3.4. What is New? The work presented here differs from the surveyed research in several major aspects. First of all, a novel optimization algorithm is used to optimize a statistically validated objective function which correlates well with the final resource utilization. The proof that the optimization task is an NP-compete problem is established. Next, for the first time in high level synthesis the inverse element law transformation is used. Also, the relationship between several transformations (e.g., retiming and associativity) is analyzed and it is shown that they enable and disable each other, and thus for the best effectiveness they have to be addressed simultaneously. Finally, although we mainly discuss application of four transformations (retiming, associativity, inverse element law and commutativity), the proposed framework is general and enables easy integration of new transformations mechanisms, new objective functions, and new optimization algorithms.

1.4. Problem Formulation

The resource utilization U_r for a resource r can be defined as the ratio of the number of control steps in which the resource is used to the available number of control steps. The total resource utilization U_{tot} of a design can then be defined as the weighted sum of the resource utilizations over all hardware primitives.

$$U_{\text{tot}} = \sum_{r \in R} w_r U_r \tag{1}$$

with R the total set of hardware resources used. The weights w_r are proportional to the hardware cost of the resource.

The cost of a design is inversely proportional to the resource utilization. Achieving a high resource utilization is in general equivalent to achieving a small design cost. During the design optimization process, it is generally easier to measure (or estimate) the resource utilization than the actual hardware cost. In the next section, we will discuss how the utilization can be estimated in an efficient and accurate way from the flow graph in combination with some detailed knowledge of the available hardware library.

The problem discussed in this section can now be defined as a constraint satisfaction problem:

Given: *A control data flow graph and a proposed hardware architecture.*

Goal: *Apply retiming, associativity, commutativity and inverse element law in such a way that the resource utilization of the resulting control data flow graph is maximized.*

The solution to the above problem can be used as a subroutine to address both (1) *the hardware minimization problem* formulation (given the time constraints) as well as (2) *the time minimization* formulation (given the available hardware). Although we will concentrate on the first formulation (which is the most appropriate for signal processing applications),

we will also discuss some experimental results for the time minimization formulation.

1.5. Paper Organization

The current section has discussed the application of transformations in a number of domains and has introduced the concept of transformations for resource utilization. Unfortunately, it turns out that applying these transformations optimally is, in general, of an NP-complete nature, as will be addressed in the appendix. An iterative probabilistic approach is therefore advocated. Achieving a high quality solution within such a framework requires the addressing of the following issues:

1) the derivation of a good objective function, i.e., how to estimate the implementation cost for a particular flow graph in a fast and accurate way.
2) how to efficiently reach the optimal flow graph, given the above objective function.

The objective function is discussed in Section II, while the algorithm which achieves the defined goals is described in Section III. The high quality of the proposed solution is demonstrated in Section IV, by statistically analyzing a number of examples. Finally, the proof that retiming for resource utilization is NP-complete problem on itself is presented in the appendix.

II. Objective Function

As stated, our goal is to apply retiming and associativity to achieve a high resource utilization (measured over all resources, namely execution units, memory and interconnect). A good objective function for the optimization should therefore be closely correlated to the final (unknown) hardware utilization. The objective function also has to be easily computable, since it is used in the optimization of an NP-complete problem, which will typically require multiple evaluations. A simple, yet effective objective function can be constructed, based on the following observations:

1) It is easier to achieve a high resource utilization when the timing constraints on the flow graph nodes are not strict;
2) It is advantageous to distribute flow graph nodes vying for the same resource over the time span. A resource is defined here as an execution unit, a register or an interconnection;
3) The critical path should be shorter than the available time;
4) The number of variables which are alive at the same time should be smaller than the number of available registers.

Those observations can be quantized in the following way:

Any operation A has to be scheduled in the interval between its *As Soon As Possible* (ASAP_A) and its *As Late As Possible* (ALAP_A) times [69]. Those times are easily computed using leveling according to the input and to the output. The slack of a node is defined as the difference between the ALAP and ASAP times, incremented by 1. A flow graph, containing a lot of operations with a small slack, represents a highly constrained scheduling problem and will often result in a poor resource utilization. However, even when the average slack is high, scheduling can still be difficult to achieve if a number of flow graph nodes with a small slack are present. Therefore, in order to properly quantify the expected chances of finding a feasible schedule, we define for each flow graph node a measure, called the *expected scheduling difficulty* (SD). SD is defined as the inverse of the slack of the node. The total scheduling difficulty of a flow graph, composed of the node-set S, is defined as the sum of the scheduling difficulties over all flow graph nodes:

$$\mathrm{TSD} = \sum_{A \in S} \frac{1}{\mathrm{ALAP}_A - \mathrm{ASAP}_A + 1} \tag{2}$$

At the same time, it is important that operations which compete for the same resource (same type of execution unit, same interconnect, or registers in the same register file) are separated in time. The probability, that two operations A and B, which need the same resource class, will happen simultaneously and therefore might require another instance of the resource, is proportional to the overlap of their ASAP-ALAP intervals (O_{AB}). This probability can therefore be approximated using the following formula:

$$\mathrm{OL}_{AB} = \frac{O_{AB}}{\mathrm{SL}_A \times \mathrm{SL}_B \times \mathrm{IR}_{AB}} \tag{3}$$

where SL_A is the slack of operation A, SL_B is the slack of operation B, and IR_{AB} is number of available resources of this class. Note, that if two nodes A and B are data or control dependent on each other, they can not be scheduled simultaneously, regardless of their potential ASAP–ALAP time overlap, and therefore they are not considered as competitors. A total overlap measure, called TOL, is defined over all nodes of the graph:

$$\mathrm{TOL} = \sum_{A \in S, B \in S, A \neq B} \mathrm{OL}_{AB} \tag{4}$$

where S is the set of all nodes in the flow graph. Notice that TSD and TOL can be conflicting measures. It is therefore important to incorporate both of them in the cost function for the transformation.

The retiming and associativity transformations can change the critical path of the graph. Obviously, no feasible schedule exists if the critical path is longer than the available time. Retiming changes the number of delays in the graph. All variables which are associated with delays must be stored simultaneously (during the first cycle) in registers. No feasible schedule is available if the number of delays exceeds the available number of registers. Both the critical path and the number of delays are incorporated in the objective function, such that it becomes infinity when one of those constraints is violated. Empirical results indicate, furthermore, that a correlation exists between the number of delays and the number of registers in the final solution. Therefore, it is useful to add the number of delays (ND) to the objective function as a measure of the total register cost.

All those factors can be combined into a global objective function, as seen in (5) on the following page.

Weight factors can be explicitly set by the user. For example, a large value for α_3 often results in fewer registers, but more

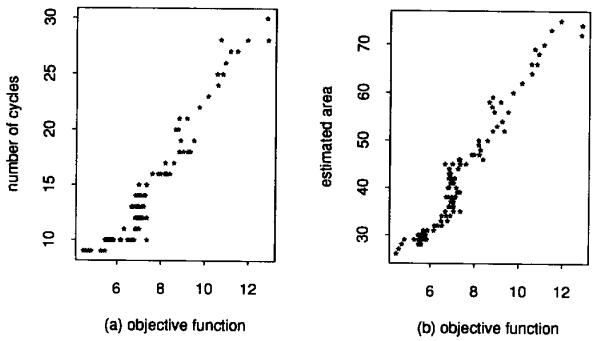

Fig. 5. Correlation between objective function and solution quality.

interconnect and execution units. For all examples discussed in the experimental section, we used the following default settings: $\alpha_1 = 0.8$, $\alpha_2 = 0.1$ and $\alpha_3 = 0.1$.

The close correlation between our objective function and the quality of the solution obtained is depicted in Fig. 5 for the example of an 11th order FIR filter. The x-axis shows the value of the objective function, while the y-axis addresses the corresponding number of control steps, necessary to execute the graph on a fixed amount of hardware (Fig. 5(a)) or the amount of hardware needed to execute the graph in a fixed amount of time (here 13 cycles) (Fig. 5(b)), respectively. The plotted data is obtained after the execution of the complete synthesis process. It is easily observed that a small value of the objective function invariantly predicts a high quality solution. Only in 3% of the examples did a larger objective function result in a better solution (over a benchmark set of 20 examples).

III. Probabilistic Sampling Algorithm

While the traditional retiming problems (for critical path and minimum number of delays) have a polynomial complexity, we have proven that the retiming for resource utilization algorithm is an NP-complete problem [71]. It is therefore very unlikely that a worst case polynomial complexity algorithm exists.

In order to efficiently solve the problem, a new probabilistic iterative improvement algorithm has been developed. This section describes first of all the set of basic moves applied during the iterative improvement, followed by a discussion of the proposed optimization strategy. Experimental results are presented and analyzed in the next sub-section.

3.1. Basic Moves

Two classes of basic moves are defined: retiming and generalized associativity moves. The retiming move was discussed in Section I and illustrated in Fig. 3(a).

TABLE II
Generalized Associativity Moves

initial	transformed
a+(b+c)	(a+b)+c
a+(b-c)	(a+b)-c
a-(b+c)	(a-b)-c
a-(b-c)	(a-b)+c
a*(b*c)	(a*b)*c
a*(b/c)	(a*b)/c
a/(b*c)	(a/b)/c
a/(b/c)	(a/b)*c

In order to enhance the power and the application range of associativity, we have expanded its definition, such that it addresses several additional cases. Those extensions are a direct consequence of two well known algebraic axioms: commutativity and the inverse element law. The generic associativity move was introduced in Fig. 3(b) and corresponds to entries 1 and 5 in Table II. Note that the application of associativity is disallowed when node A has other fanout besides node B. The basic moves have been extended with six additional transformations using the inverse element law, as shown in Table II. Cases 3, 4, 7, and 8 are especially interesting, since they allow for a trade-off between the number of additions and subtractions and the number of multiplications and divisions in the flow graph. Furthermore, by incorporating commutativity in the definition of generalized associativity, additional possibilities are obtained. For example, applying commutativity on the first row of generic moves in Table II expands it into 12 equivalent form; for the expression $a + (b + c)$, the resulting formulations are $a + (b + c) = (a + b) + c = a + (c + b) = (a + c) + b = b + (a + c) = (b + a) + c = b + (c + a) = (b + c) + a = c + (a + b) = (c+a)+b = c+(b+a) = (c+b)+a$. Notice also that for each move, defined in Table II, an inverse transformation exists. We call the inverse transformations the *reverse moves*, in contrast to the moves of Table II, which are called the *forward moves*.

3.2. Associativity—Retiming Relationship

One might wonder why we particularly chose to combine generalized associativity which from now on we will simply refer to as associativity and retiming in the same framework. As already mentioned in the introduction, retiming and associativity are not orthogonal. If we want to exploit the full power of those transformations, a simultaneous application is actually mandatory. For instance, the associativity transforma-

$$\text{Objective} = \begin{cases} \infty, & \text{if } t_{\text{crit}} > \text{ available time or} ND > \text{ number of registers} \\ \alpha_1 \times \text{TSD} + \alpha_2 \times \text{TOL} + \alpha_3 \times ND, & \text{otherwise} \end{cases} \quad (5)$$

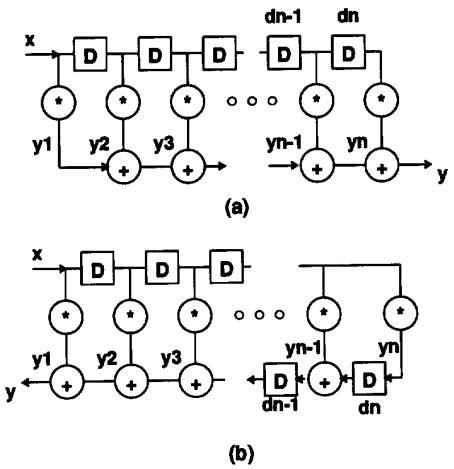

Fig. 6. Combining associativity and retiming (FIR filter).

tion cannot be applied on the following expression:

$$F = D(a + b) + c$$

Moving the delay first using retiming and applying associativity next, results in the following equation, which might produce better a resource utilization:

$$F = D(a) + (D(b) + c)$$

A more dramatic example is shown in Fig. 6. Fig. 6(a) shows the direct implementation of an nth order FIR filter. The direct form implementation displays a long critical path. Retiming alone does not help to solve this problem: note that neither delay d_{n-1} nor d_n can be moved across additions. However, after applying associativity $(n-1)$ times on the accumulation chain, we can reverse the data flow direction in the chain:

$$\begin{aligned} y &= (((y_1 + y_2) + \cdots + y_{n-1}) + y_n) \\ &= (y_1 + (y_2 + \cdots + (y_{n-1} + y_n))) \end{aligned}$$

In this way, we obtain the structure of Fig. 6(b), where delays can be freely moved and recombined (as is shown for d_{n-1} and d_n). The retiming operation can be repeated until all delay operators are moved to the accumulation path, resulting in the reciprocal FIR filter structure. Although the most visible change in the computational structure is a reduction in the critical path, it is easy to see that the retiming enabled by associativity will provide a better resource utilization too.

3.3. Probabilistic Sampling Algorithm

When applying the above transformations on a typical example, a large number of moves is possible at every point in time. Because even more basic moves might be introduced in the future, time-efficient algorithms are definitely necessary. We first tried the popular simulated annealing technique [39]. Although the results were satisfactory, the run times were excessive for large examples, even when adaptive cooling [1], [12], [44], [74] and a rejectionless approach [28], [71] were applied. Therefore, we decided to use a new and faster probabilistic approach.

TABLE III
CORRELATION AMONG SOLUTIONS ON DISTANCE OF M MOVES

m	1	2	3	4	5	6
min	0.971	0.932	0.919	0.902	0.866	0.808
average	0.978	0.952	0.927	0.909	0.878	0.841
max	0.994	0.978	0.959	0.932	0.907	0.879

The technique is organized as a two phase process. In the first phase, the solution space is scanned in an organized fashion to detect areas where the objective function has a small value. Those areas are then used in the second phase as the starting points for a more elaborate search towards a final solution.

The goal of the first phase is thus to discover k solutions where the objective function has the smallest value. In order to achieve this goal, we will traverse the search space a number of times, each time favoring one particular direction of traversal. For instance, we will first (probabilistically) favor moves in the forward direction (moving the delays from the inputs to the outputs for the retiming and favoring the forward associativity moves). After 1/4 of the optimization process, the preferred direction is reversed: moving the delays from output to input as well as the reverse associativity moves are now probabilistically favored. Finally, for the last 1/4 of the time, the forward moves are favored anew.

At every point in the optimization process, a move is selected in a probabilistic fashion, in correspondence with the favored move direction. No selected moves are ever rejected. Moves which increase the objective function can be selected during the sampling phase. To increase the speed of the design space search, the objective function is only evaluated every four steps, resulting in a speed-up of approximately four times. This is acceptable, since neighboring solutions in the design space (solutions which can be reached in at most m moves) tend to display a strong correlation in their objective function values. Since the exact location of a local minimum is only determined in the second phase, no degradation of the solution quality was observed as a result of this simplification. Table III shows the minimum, average, and maximum correlation among neighboring solutions for several examples.

The first phase results in k starting points for the second phase. Those are used as the seeds for a greedy search towards the final solution. The objective function is now observed at each step and for all possible moves. The move offering the best decrease in the objective function is automatically selected. For each starting point, the search is concluded when a local minimum is reached. The best of those minima is selected as the final solution.

The number of starting points k is set to 10 for the examples discussed in the next section. The length of the first phase is such that the number of moves during the first forward traversal equals 10 times the number of nodes in the graph. Moves in the forward direction are preferred with a ratio 4:3. We have varied these values over substantial ranges and did not notice a significant change in the quality of the

TABLE IV
SAVINGS AGAINST THE INITIAL AND RANDOM IMPLEMENTATIONS (IN %)

	EXU		MEMORY		INTERCONNECT	
	Random	Initial	Random	Initial	Random	Initial
average	40.1	32.5	25.4	23.6	33.8	29.7
median	38.5	33.3	26.6	22.2	36.6	30.0
min	20.0	9.1	3.6	6.9	8.3	10.0
max	70.8	50.0	44.4	46.0	63.3	50.0

solution, although the effects on the run times were significant. The presented approach can easily be augmented with a cooling mechanism (phase 1) or backtracking. Experiments have shown, however, that those extensions do not produce any significant improvements and have a detrimental effect on the run time.

It is interesting to notice that the presented approach resembles, to some extent, the simulated annealing approach [39]. While phase 1 uses an iterative, probabilistic improvement technique, similar to simulated annealing, some major differences with the annealing approach can be observe. First of all, the presented technique uses a directed search, while annealing executes random moves. The major difference, however, is the combination of probabilistic exploration and greedy solution generation.

IV. EXPERIMENTAL RESULTS

The ultimate proof of the usefulness and effectiveness of a proposed transformation or optimization algorithm is to apply it on real life examples. We have applied our technique on 40 flow graphs, which include common DSP, communication and error-correcting code examples, such as FIR, IIR (direct form, parallel and cascade form, ladder wave digital form), nonlinear polynomial and median filters, adaptive filters and simultaneous polynomial division and multiplication. For each of those examples, we varied the available time. The smallest example had only eight operations, while the largest example had 596 operations. The number of delays varied from 2 to 127. We also varied the relative execution lengths of the operators, such as shifts, adds, multiplications and multiplexers.

The primary objective was to measure how much hardware is saved by applying the transformation. A secondary task was to evaluate the potential of the transformation to minimize the execution time of an algorithm without increasing the latency (in contrast to pipelining). While improvements in execution units and memory cost can be measured exactly, the cost of interconnect could only be estimated, since precise numbers are only available after time consuming and tool dependent tasks such as floorplanning and routing. We have estimated interconnect cost based on the number of busses and assumed that all busses have the same cost.

The ratio of the implementation cost after the application of the transformation over the cost of the initial implementation for all benchmark examples is plotted in Fig. 7(a), (b), and (c) (for execution units, memory and interconnect costs respectively).

Comparing the final solution with the initial flow graph, provided by the designer, has only limited significance since this highly depends on the amount of manual optimization, applied by the designer. We therefore compared the cost of the generated solutions against the median cost of 20 random solutions (generated by randomly applying retiming and associativity moves for a long period of time) as well. Fig. 8(a)–(c) shows the ratios of the optimized solutions against the random ones. The average and median savings against the initial implementation and the median random implementation are tabulated in Table IV.

The generation of the above results required the use of a particular set of scheduling, assignment and allocation tools. Since those problems are NP-complete as well, it might be argued that the obtained improvements were not the result of the transformation by itself, but are due to the fact that the heuristic scheduler performed better on the transformed graphs. This argument can be discarded using the following simple procedure. For each instance of a flow graph, it is possible to establish sharp minimum bounds on the resources, by observing that no candidates are available for is some control steps [72] These bounds can not be outperformed, regardless of the scheduling method used. If the application of a transformation results in a decrease of those bounds, then this improvement is a pure consequence of the transformation. The ratios of the final implementation cost versus the estimated min-bounds of the initial solutions for execution units are plotted in Fig. 9. The values of the median, average and maximum improvement for execution units area respectively equal 21.6%, 21.7% and 47.1%. Only once was the min-bound not reached.

To evaluate the effects of the transformation on a well known benchmark, we have applied the technique on the popular fifth order elliptical wave digital filter example [17]. The results are tabulated in Table VI for the available times ranging from 15 to 19 clock cycles, in correspondence with the standard benchmark, we assume that a multiplication and an addition take respectively 2 and 1 clock cycle. As can be

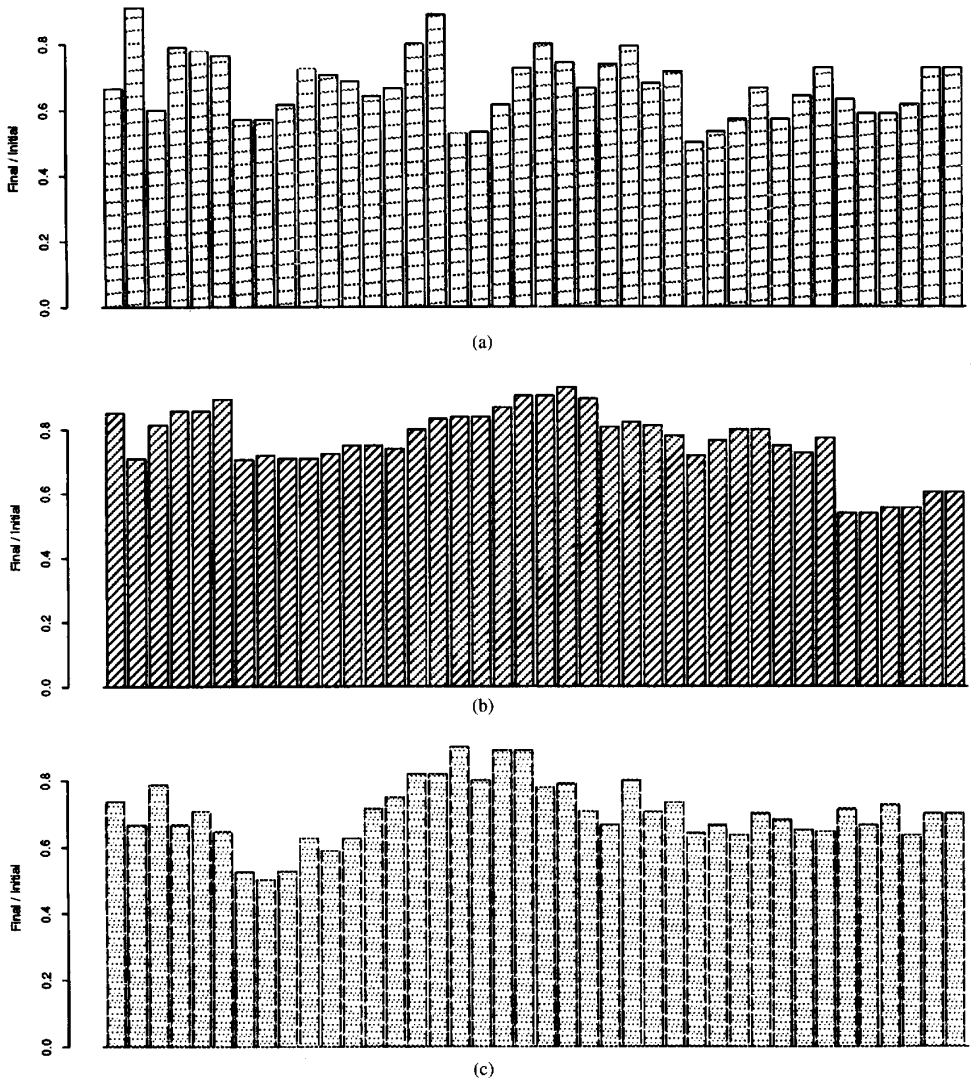

Fig. 7. Ratio of implementation cost of final versus initial implementation for benchmark example set. (a) Execution units. (b) Memory. (c) Interconnect.

observed from the table, the average hardware savings due to the transformations are impressive. Finally, to demonstrate how the transformation succeeds in optimizing the resource utilization, the flow graph and the schedule of the filter after the transformation are plotted in Figs. 10 and 11, assuming that the available time equals 16 cycles. The schedule is also presented in Table V. It is interesting to notice that the fastest solution after retiming for critical path still needs 16 clock cycles [31]. The combination of retiming and associativity succeeds in producing a solution which requires only 15 cycles.

The effectiveness of the algorithm is illustrated by the fact that, even for graphs with several hundred nodes, the run time was shorter than one minute. Table VII shows the run times for several examples (measured in seconds).

In some cases, retiming and associativity for resource utilization result in the spectacular improvements. For example in the case of the second order Volterra filter [54] the proposed transformation reduces the area by a factor of three. Figs. 12 and 13 show the layouts before and after the application of transformations. While the initial implementation required four multipliers, six barrel shifters, two adders and two subtractors, one multiplier, one barrel shifter, one adder and one subtractor were sufficient after. The number of registers was reduced from 38 to 28. Also, the critical path, which was initially 16 control steps, was reduced to 15 control steps.

4.1. Application Range

Although the new approach showed a significant improvement on a large set of test examples, one might wonder about the application range of the transformation. The majority of the examples presented in the previous section are filters. Filters are obviously the ideal target for the proposed technique, since they normally have real time constraints imposed on

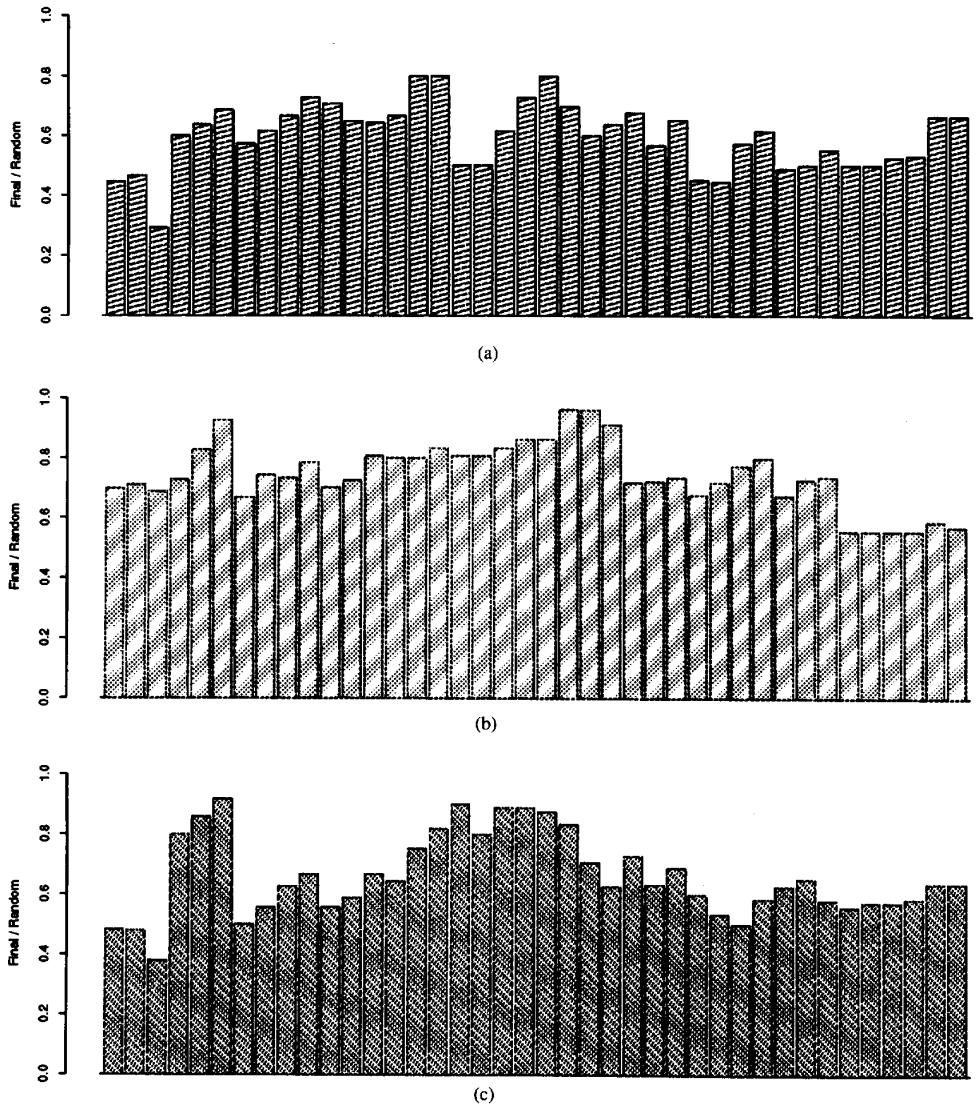

Fig. 8. Ratio of implementation cost of final versus random implementation for benchmark example set. (a) Execution units. (b) Memory. (c) Interconnect.

TABLE V
SCHEDULE FOR FIFTH ORDER ELLIPTICAL FILTER, USING TWO ADDERS (A_1 AND A_2) AND TWO MULTIPLIERS (M_1 AND M_2) IN 16 CONTROL STEPS

CS	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15	16
A1	0	1	4			6	7	19	22	9	25	10	12	17	18	13
A2	3	2				21	23	26	14	25	27	31		29	30	33
M1				5	5			8	8			16	16	11	11	
M2				20	20			24	24			28	28	32	32	

them, since they operate on infinite streams and since they combine information from subsequent samples. As a net result, delays are naturally present in all filter structures and hence, retiming can be very effective. It should be mentioned that filters are still the most important signal processing components, even though more complex, nonlinear or multi-dimensional operations are becoming increasingly important [10].

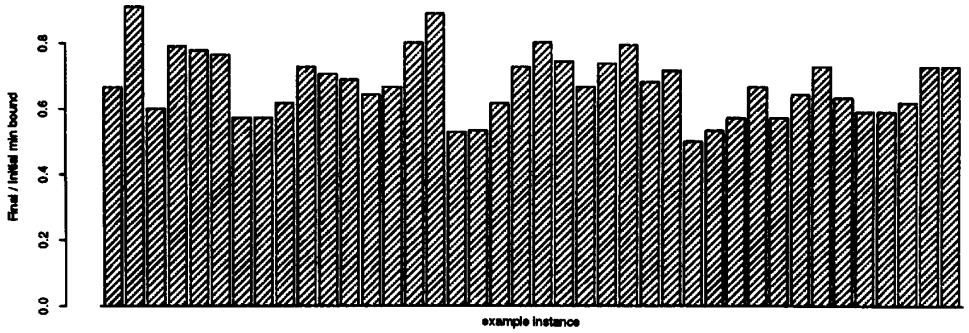

Fig. 9. Ratio of implementation cost of final versus min bounds of initial solutions for benchmark example set (execution units).

TABLE VI
NUMBER OF ADDERS AND MULTIPLIERS USED FOR IMPLEMENTATION OF FIFTH ORDER ELLIPTICAL FILTER BEFORE AND AFTER APPLYING RETIMING AND ASSOCIATIVITY. THE ROW BEFORE SHOWS THE BEST KNOWN SOLUTIONS TO THE FIFTH ORDER ELLIPTICAL FILTER (E.G., [69]), WITHOUT USE OF TRANSFORMATIONS

	Number of Control Steps	15	16	17	18	19
BEFORE	# of adders / # of multipliers	NA	NA	3/3	3/2	2/2
AFTER	# of adders / # of multipliers	3/3	2/2	2/2	2/2	2/1

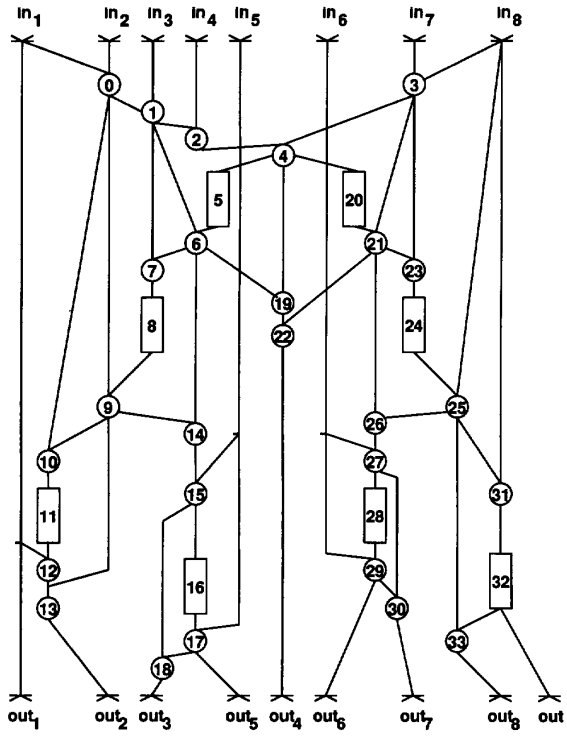

Fig. 10. Flow graphs and schedules of fifth order elliptical filter before transformations.

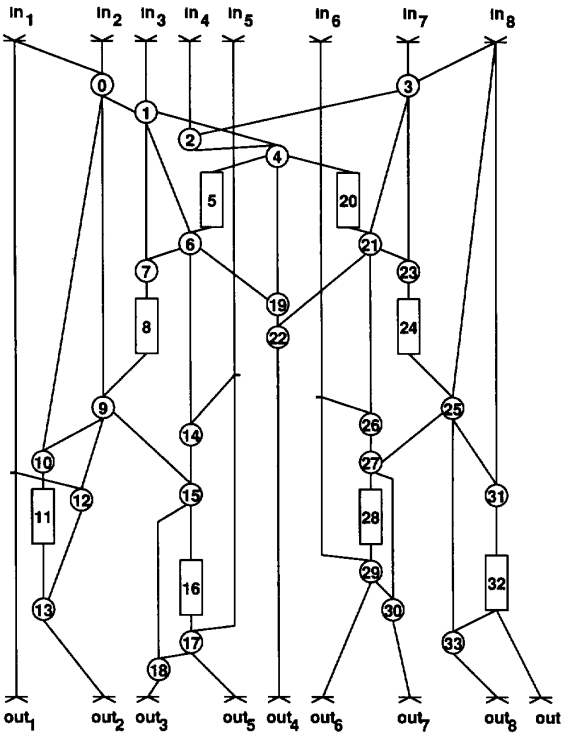

Fig. 11. Flow graphs and schedules of fifth order elliptical filter after transformations.

It is important to stress that the application of retiming and associativity is by no means restricted to filters. While this is obvious for the application of associativity, it is important to note that the richest source of delays in a program is the loop construct. Whenever a loop body uses information from a previous iteration, a delay is introduced. In fact, the stream oriented nature of signal processing applications is nothing else than an infinite loop over time. Therefore, retiming is

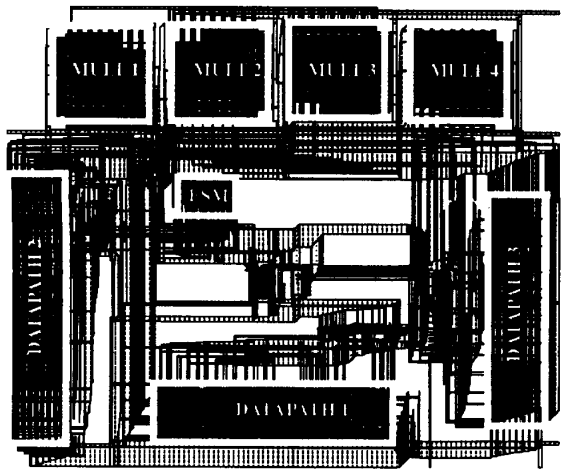

Fig. 12. Volterra filter before retiming and associativity for resource utilization.

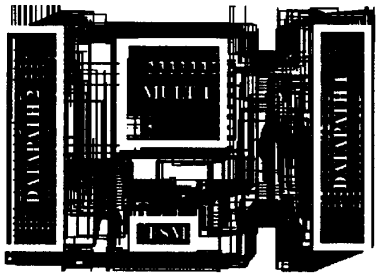

Fig. 13. Volterra filter after retiming and associativity for resource utilization.

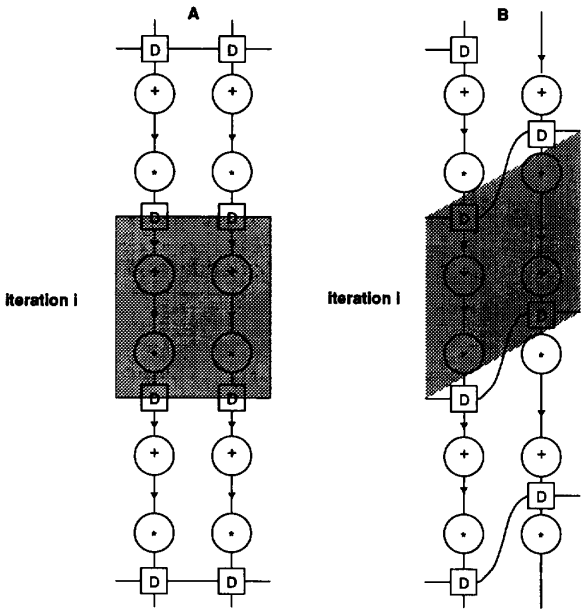

Fig. 14. Software retiming: (a) Before and (b) after.

applicable to almost all problem instances which employ iteration or recursion. The application of retiming in those cases can be called *software retiming* in correspondence with the well known *software pipelining* transformation [45], [46]. The proposed transformation is therefore also effective in a multitude of other signal processing applications, which rely on the extensive execution of a tight inner loop. Examples of those can be found in the areas of multi-dimensional signal processing, sonar, speech recognition (Markov Modeling), telecommunications (Viterbi search), various signal transformations such as the DFT and DCT and digital audio (error correction, adaptive interpolation). A simple example of the usage of *loop retiming* is shown in Fig. 14. Assuming that two clock cycles are available per iteration and that both a multiply and an add take one cycle, it is easily observed that the first instance requires two multipliers and two adders, while the retimed graph only needs one unit of both.

Two other important questions both for a user of the design system as well as for the development of automatic transformations can be formulated:

1) When can this transformation be applied?
2) What is the potential improvement which can be achieved by applying the transformation?

In general, finding an answer to the former question is extremely difficult. It is for instance unanswerable for a loop transformation such as loop jamming [6]. However, in the case of retiming and associativity for resource optimization, the answer is straightforward: there exists no restriction on the application. The most promising targets are flow graphs with a lot of delays and structures with chained associative operators.

This brings us to the second question. In the previous section, we have already discussed that the transformation lowers the minimal bounds on the resources, hence making it possible to obtain cheaper solutions. The question raised here is what is the maximum improvement achievable by the transformation. In a way, this can be predicted by comparing

TABLE VII
RUN TIMES FOR FOUR EXAMPLES (SPARCSTATION 2, 48 MB) FOR VOLTERRA FILTER, FIFTH ORDER ELLIPTICAL WAVE DIGITAL FILTER, AND SEVENTH ORDER IIR FILTER BEFORE AND AFTER APPLICATION OF MULTIPLICATION TO SHIFT AND ADDITION TRANSFORMATION

Number of Nodes	32	34	40	113
Run Time [s]	5	7	11	34

the estimated minimum bounds of the initial flow graph with the *absolute min-bounds*. These bounds can be predicted using the following formula:

$$N_R^{\text{abs}} = \left\lceil \frac{o_R \times d_R}{t_{\text{available}}} \right\rceil \tag{6}$$

with O_R the number of operations using resource R, d_R the duration of operator R (in clock-cycles) and $t_{\text{available}}$ the available time. The absolute min-bound N_R^{abs} estimates the number of instances needed of the resource R under the assumption that a 100% utilization is achieved. Obviously, this is the best result which can be obtained with the transforma-

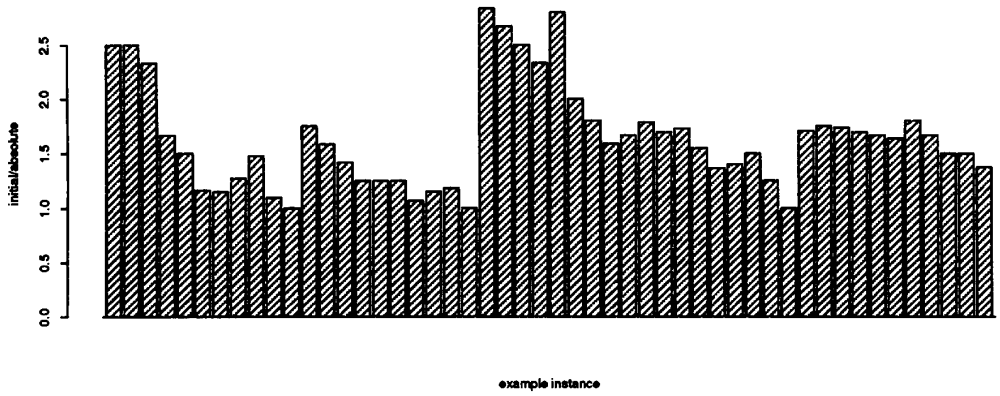

Fig. 15. Ratio of initial min bounds versus absolute min bounds for execution units (48 examples).

tional approach presented here[1]. The potential improvements can hence be measured by comparing the absolute bounds with the estimated min-bounds of the initial flow graph. The results for some of the benchmarks are displayed in Fig. 15. The average potential improvement for those 48 examples is 35%, median 38%, maximal 65%, and minimal 0%. For the 40 examples were we tested the new approach the average potential improvement was 40% percent, median 42%, maximal 65% and minimal 13.3%. Therefore, the probabilistic sampling algorithm realized around 80% of the potential improvement, what is a very high ratio, taking into account that absolute bounds assume an ideal hardware utilization.

It should be mentioned that even when this comparison predicts no improvement at all, it still might be useful to attempt the transformation: as discussed above, the transformation relaxes the constraints in the graph and makes it easier to produce a feasible schedule using the minimal hardware.

V. Conclusion

A transformational approach, aimed at improving the resource utilization in high level synthesis, has been introduced. The current implementation combines retiming and associativity in a single framework. This combination of transformations results in considerable area improvements as is amply demonstrated by the benchmark examples. A novel *probabilistic sampling* iterative improvement probabilistic algorithm has been developed and is used to resolve the associated NP-complete combinatorial optimization problem. The proposed algorithm has proven to be very effective in reaching the optimal solution both for hardware resource usage and for run-time.

Appendix
Computational Complexity

As mentioned before, one of the most important properties of the proposed transformation is its NP-completeness. In this appendix, this will be proven for the retiming for resource utilization transformation. More precisely, we will prove that the following simplified version is NP-complete. Other more realistic and complex versions, which take into account the cost of interconnect and registers, are also NP-complete. This can be proven without any difficulty by restricting them to this version [23].

[1] A word of caution: some of the associativity moves interchange additions for subtractions and multiplications for division. This effectively changes the absolute min-bounds.

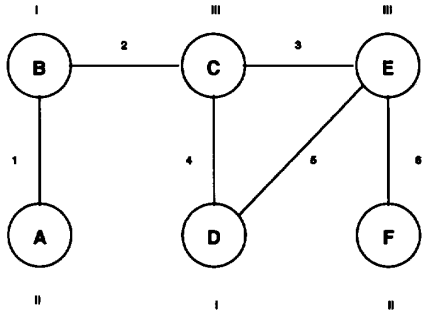

Fig. 16. MAX 3-CUT problem.

PROBLEM: RETIMING FOR RESOURCE UTILIZATION:

INSTANCE: Signal flow graph SFG = (SV, SG), available time T, cost $c(v) \in Z^+$ for each $v \in$ HV, positive integer L.

QUESTION: Is there a retiming for resource utilization of SFG such that its implementation is at most L in cost?

The problem is in NP because a feasible schedule which requires implementation with the cost L, if it exists, can be exhibited and checked for feasibility quite easily.

We shall now polynomially transform the *MAX 3-CUT* problem to *RETIMING FOR RESOURCE UTILIZATION* to finish NP-completeness proof.

PROBLEM: MAX 3-CUT [23]:

INSTANCE: Graph $G = (V, E)$, weight $w(e) \in Z^+$ for each $e \in E$, positive integer K.

QUESTION: Is there a partition of V into disjoint sets V_1, V_2, V_3 such that the sum of weights of the edges from E that have its endpoints in different sets is at least K?

Suppose we are given an arbitrary graph G as shown in Fig. 16. We will first polynomially transform G to a corresponding SFG so that finding a solution in polynomial time to the

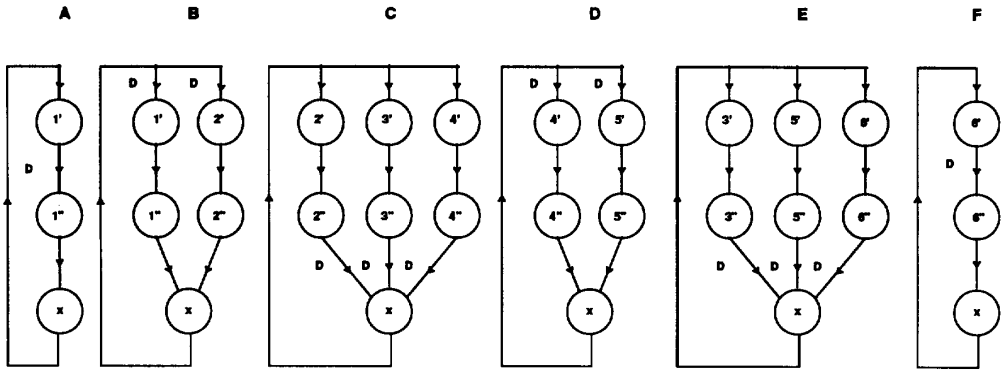

Fig. 17. Retiming for resource utilization.

retiming for resource utilization problem implies a polynomial time solution to the MAX 3-CUT problem.

For each node V in G, the SFG contains a disjoint subgraph, which contains as many cycles as the node V has incident edges. An illustration is shown in Fig. 17. For example, subgraph B in the SFG contains two cycles, corresponding to the two incident edges (1 and 2) to node B in G. Cycle C, corresponding to C in G, contains three nodes of different types C', C'', and x. Node x is part of every cycle in the subgraph and acts as enforcer [23]. Each operation takes one control cycle, and the available time is three control cycles. Operations $c(C')$ and $c(C'')$ has cost equal to the weight $w(C)$ in G.

The enforcer node x has cost 0. It enforces all delays in a subgraph to be at the same level. Otherwise, no feasible schedule exists

The problem has been constructed such that in order for two operations of the same type to share the same resources, the delays in their subgraphs have to be positioned at different levels. For instance, the delays in subgraphs A and B have been placed at different levels. Therefore, nodes $1'$ from both subgraphs can be scheduled on the same hardware in cycles 1 (subgraph B) and 3 (subgraph A). Putting delays on different levels corresponds to putting nodes, incident to corresponding edge in G, in different disjoint sets.

A retiming with no hardware sharing obviously has the cost $2M$, where M is the sum of all edge weights in G. This corresponds to a 0 cut for the MAX 3-CUT problem since all nodes are in the same set. Moving one delay (e.g., on the edge with nodes $5'$ and $5''$ in the subgraph D) will reduce the cost of the implementation by $w(5)$ and increase cutset value by the same amount. Therefore, if we can achieve the retiming for resource utilization solution with the cost $L = 2M - K$, this corresponds to the solution of the MAX 3-CUT problem with cost K. Consequently, we have demonstrated that MAX 3-CUT polynomially transforms to retiming for resource utilization.

REFERENCES

[1] E. H. L. Aarts and J. H. M. Korst, *Simulated Annealing and Boltzmann Machine*. Chichester, United Kingdom: Wiley, 1989.

[2] A. A. Aho and J. D. Ullman, *Principles of Compiler Design*. Reading, MA: Addison Wesley, 1977.

[3] F. E. Allen, M. Burke, P. Charles, R. Cytron, and J. Ferrante, "An overview of the PTRAN analysis system for multiprocessing," *J. Parallel and Distributed Computing*, pp. 617–640, 1988.

[4] R. Allen, D. Baumgartner, K. Kenedy, and A. Porterfield, "PTOOL: A semi-automatic parallel programming assistant," in *Proc. Int. Conf. Parallel Processing*, 1986, pp. 164–170.

[5] R. Allen and K. Kenedy, "Automatic translation of FORTRAN programs to vector form," *ACM Trans. Programming Languages and Systems*, vol. 9, no. 4, pp. 491–542.

[6] U. Banerjee, Dependence Analysis for Supercomputing. Norwell, MA: Kluwer, 1988.

[7] K. Bartlett, G. Borriello, and S. Raju, "Timing optimization of multiphase sequential logic," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 51–62, 1991.

[8] J. Bhaskar and H. Lee, "An optimizer for hardware synthesis," *IEEE Design and Test*, Oct. 1990.

[9] S. N. Bhatt, F. R. K. Chung, and A. R. Rosenberg, "Partitioning circuits for improved testability," *Algorithmica*, vol. 6, no. 1, pp. 37–48, 1991.

[10] R. E. Blahut, Fast Algorithms for Digital Signal Processing. Reading, MA: Addison Wesley, 1985.

[11] A. E. Casavant, D. D. Gajski, and D. J. Kuck, "Automatic design with dependence graphs," in *Proc. 17th ACM/IEEE Design Autom. Conf.*, 1980, pp. 506–515.

[12] F. Catthoor, H. DeMan, and J. Vanderwalle, "SAMURAI: A general and efficient simulated-annealing schedule with fully adaptive annealing parameters," *Integration*, vol. 6, pp. 147–178, 1988.

[13] M. Chen, Y. Choo, and J. Li, "Compiling parallel programs by optimizing performance," *J. Supercomputing*, vol. 1, no. 2, pp. 171–207, 1988.

[14] A. Darringer, W. H. Joyner, L. Berman, and L. Trevillyan, "Logic synthesis through local transformations," *IBM J. Res. Develop.*, vol. 25, no. 4, pp. 272–280, 1981.

[15] G. De Micheli, "Synchronous logic synthesis: Algorithms for cycle-time minimization," *IEEE Trans. Computer-Aided Design*, vol. 10, pp. 63–73, 1991.

[16] S. Devadas and A. R. Newton, "Algorithms for hardware allocation in data path synthesis," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 768–781, 1989.

[17] P. Dewilde, E. Deprettere, and R. Nouta, "Parallel and pipelined VLSI implementation of signal processing algorithms", in *VLSI and Modern Signal Processing Algorithms*, S. Y. Kung, H. J. Whitehouse, T. Kailtah, ed. 1985, pp. 257–264.

[18] S. Dey, F. Brglez, and G. Kedem, "Partitioning sequential circuits for logic optimization," *IEEE Int. Conf. Computer Design*, 1991, pp. 70–76.

[19] P. Duncan, S. Swamy, S. Sprouse, D. Potasz, R. Jain, N. Gafter, W. Cammack, Y. Wong, and W. Gass, "HI-PASS: A Computer aided synthesis system for fully parallel digital signal processing ASIC's, " in *Proc. Int. Conf. Acoustic, Speech & Signal Processing*, Mar. 1992, San Francisco, CA.

[20] A. Fetweis, H. Meyr, and L. Thiele, "Algorithm transformations for unlimited parallelism," *IEEE Int. Symp. Circuits and Systems*, New Orleans, LA, 1990, pp. 1756–1759.

[21] C. N. Fisher and R. J. LeBlanc, Jr., *Crafting a Compiler*. Menlo Park, CA: Benjamin/Cummings, 1988.

[22] D. D. Gajski, "An algorithm for solving linear recurrence systems on parallel and pipelined machines," *IEEE Trans. Computers*, vol. 30, pp. 190–206, 1981.
[23] M. R. Garey and D. S. Johnson, "Computers and Intractability: A Guide to the Theory of Np-completeness. New York: W. H. Freeman 1979.
[24] D. Goldberg, "What every computer scientist should know about floating-point arithmetic," *ACM Computing Surveys*, vol. 23, no. 1, pp. 5–48, 1991.
[25] G. H. Golub and C. F. van Loan, *Matrix Computations*. Baltimore, MD: Johns Hopkins University Press,.
[26] G. Goossens, R. Jain, J. Vandewalle, and H. De Man, "An optimal and flexible delay management technique for VLSI," in *Computation and Combinational Methods in System Theory*, C. I. Byrnes, A. Lindquist, ed. Amsterdam, The Netherlands: North Holland, 1986, pp. 409–418.
[27] G. Goossens, J. Rabaey, J. Vandewalle, and H. De Man, "An efficient microcode compiler for application specific DSP processors," *IEEE Trans. Computer-Aided Design*, vol. 9, pp. 925–937, 1990.
[28] D. Gregory, K. Bartlett, A. de Geus, and G. Hachtel, "SOCRATES: A system for automatically synthesis and optimizing combinational logic," in *Proc. 23rd ACM/IEEE Design Autom. Conf.*, 1986, pp. 79–85.
[29] E. Gyrczyc, "Automatic Generation of Microsequenced Data Paths to Realize ADA Circuit Description," Ph. D. dissertation, Carleton University, 1984.
[30] B. S. Haroun and M. I. Elmasry, "Architectural synthesis for DSP silicon compilers," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 431–447, 1989.
[31] R. Hartley, A. Casavant, "Tree-height minimization in pipelined architectures," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 112–115, 1989.
[32] A. C. Hearn, "REDUCE User's Manual: Version 3.3," RAND Publication CP78, The Rand Corporation, Santa Barbara, CA, 1987.
[33] C.-T. Hwang, Y.-C. Hsu, and Y.-L. Lin, "Scheduling for functional pipelining and loop winding," in *Proc. ACM/IEEE Design Autom. Conf.*, San Francisco, CA, 1991, pp. 764–769.
[34] A. T. Ishii and C. E. Leiserson, "A timing analysis of level-clocked circuits," in *Advanced Research in VLSI: Proc. Sixth MIT Conf.*, 1190, pp. 113–130.
[35] W. H. Joyner, Jr., L. H. Trevillyan, D. Brand, T. A. Nix, and S. C. Gundersen, "Technology adaptation in logic synthesis," in *Proc. 23rd ACM/IEEE Design Autom. Conf.*, 1986, pp. 94–100.
[36] R. M. Karp and V. Ramachandran, "Parallel algorithms for shared-memory machines," in *Handbook of Theoretical Computer Science, Volume A: Algorithms and Complexity*, J. van Leewen, ed. Cambridge, MA: MIT Press/Elsevier, 1990, pp. 869–941.
[37] K. Kennedy, K. S. McKinely, and C. Tseng, "Interactive parallel programming using the parascope editor," *IEEE Trans. Parallel and Distributed Systems*, vol. 2, pp. 329–341, 1991.
[38] B. W. Kernighan and S. Lin, "An efficient heuristic procedure for partitioning graphs," Bell System Tech. J., vol. 49, pp. 291–307, 1970.
[39] S. Kirkpatrick, C. D. Gellat, Jr., and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.
[40] D. J. Kuck, *The Structure of Computers and Computations*. New York: Wiley, 1978.
[41] D. Kuck, R. Kuhn, B. Leasure, and M. Wolfe, "The structure of an advanced vectorizer for pipelined processors," in *Proc. COMPSAC 80, The 4th Int. Computer Software and Application Conf.*, 1980, pp. 709–715.
[42] S. Y. Kung, "On supercomputing with systolic/wavefront array processors," *Proc. IEEE*, vol. 72, pp. 867–884, 1984.
[43] S. Y. Kung, VLSI Array Processors. Englewood Cliffs, NJ: Prentice Hall, 1988.
[44] J. Lamm and J.-M. Delosme, "Performance of a new annealing schedule," in *Proc. 25th ACM/IEEE Design Autom. Conf.*, Anaheim, CA, 1988, pp. 306–311.
[45] M. Lam, "Software pipelining: An effective scheduling technique for VLIW Machines," *ACM SIGPLAN*, 1988.
[46] M. S. Lam, *A Systolic Array Optimizing Compiler*. Norwell, MA: Kluwer Academic, 1989.
[47] C. E. Leiserson, F. M. Rose, and J. B. Saxe, "Optimizing synchronous circuits by retiming," in *Proc. Third Conf. VLSI*, 1983, pp. 23–36.
[48] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica*, vol. 6, no. 1, pp. 5–35, 1991.
[49] L. Liu, T. Yoshino, S. Sprouse, and R. Jain, "An interleaved/retimed architecture for the lattice wave digital filter," *IEEE Trans. Circuits and Systems*, vol. 38, pp. 344–347, 1991.
[50] D. A. Lobo and B. M. Pangrle, "Redundant operation creation: A scheduling optimization technique," in *Proc. ACM/IEEE Design Autom. Conf.*, San Francisco, CA, 1991, pp. 775–778.
[51] VAX UNIX MACSYMA Reference Manual, Version 11, Symbolics, Inc.
[52] S. A. Mahlke, N. J. Warter, W. Y. Chen, P. P. Chang, and W. W. Hwu, "The effect of compiler optimizations on available parallelism in scalar programs," in *Proc. Int. Conf. Parallel Processing*, pp. 142–145, 1991.
[53] S. Malik, E. M. Sentovich, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming and resynthesis: Optimizing sequential networks with combinational technique," *IEEE Trans. on Computer-Aided Design*, vol. 10, pp. 74–84, 1991.
[54] V. J. Mathews, "Adaptive polynomial filters," *IEEE Signal Processing Mag.*, vol. 8. pp. 10–26, July 1991.
[55] D. E. Maydan, J. L. Hennessy, and M. S. Lam, "Efficient and exact data dependance analysis," in *Proc. ACM SIGPLAN '91 Conf. Programming Language Design and Implementation*, Toronto, ON, Canada, 1991, pp. 1–14.
[56] M. C. McFarland and A. C. Parker, "An abstract model of behavior for hardware descriptions," *IEEE Trans. Computers*, vol. C-32, pp. 621–636, 1983.
[57] M. C. McFarland, A. C. Parker, and R. Camposano, "The high-level synthesis of digital systems," *Proc. IEEE*, vol. 78, pp. 301–317, Feb. 1990.
[58] D. Messerschmitt, "Breaking the recursive bottleneck," in *Performance Limits in Communication Theory and Practice* Norwell, MA: Kluwer Academic, 1988.
[59] G. L. Miller, V. Ramachandran, and E. Kaltofen, "Efficient parallel evaluation of straight-line code and arithmetic circuits," *SIAM J. Computing*, vol. 17, no. 4, pp. 687–695, 1988.
[60] G. S. Miranker, "The Use of Conflicts in the Translation and Optimization of Hardware Description Languages," Ph. D. dissertation, Massachusetts Inst. Technology, May 1979.
[61] A. Nicolau and R. Potasman, "Incremental tree height reduction for high level synthesis," in Proc. ACM/IEEE Design Autom. Conf., San Francisco, CA, 1991, pp. 770–774.
[62] O'hEigeartaigh, J. K. Lenstra, and A. H. G. Rinnooy Kan, *Combinatorial Optimization: Annotated Bibliographies*. New York: Wiley, 1985.
[63] D. A. Padua and M. J. Wolfe, "Advanced compiler optimizations for supercomputers," *Commun. ACM*, vol. 29, no. 12, pp. 1184–1201, 1986.
[64] D. Padua, "The delta program manipulation system, preliminary design," CSRD Tech. Rep. 880, Center for Supercomputing Research and Development, Univ. of Illinois at Urbana-Champaign, June 1989.
[65] K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters—I: Pipelining using scattered look-ahead and decomposition," *IEEE Trans. Acoust. Syst. Signal Processing*, pp. 1099–1117, July 1989.
[66] K. K. Parhi and D. G. Messerschmitt, "Pipeline interleaving and parallelism in recursive digital filters—II: Pipelined incremental block filtering," *IEEE Trans. Acoust. Syst. Signal Processing*, pp. 1118–1134, July 1989.
[67] N. Park and A. C. Parker, "Sehwa: A program for synthesis of pipelines," in Proc. ACM/IEEE Design Autom. Conf., Las Vegas, NV, 1986, pp. 454–460.
[68] N. Park and A. C. Parker, "Sehwa: A software package for synthesis of pipelines from behavioral specifications," *IEEE Trans. Computer-Aided Design*, vol. CAD-7, pp. 356–370, 1988.
[69] P. G. Paulin and J. P. Knight, "Force-directed scheduling for the behavioral synthesis of ASIC," *IEEE Trans. Computer-Aided Design*, vol. 8., pp. 661–679, 1989.
[70] C. D. Polychronopolous, M. Girkar, M. R. Haghighat, C. L. Lee, B. Leung, and D. Schouten, "Parafrase-2: An environment for parallelizing, partitioning, synchronizing, and scheduling on multiprocessors," in *Proc. Int. Conf. Parallel Processing*, 1989, pp. 39–48.
[71] M. Potkonjak and J. Rabaey, "Retiming for scheduling," *VLSI Signal Processing Workshop*, San Diego, CA, Nov. 1990, pp. 23–32.
[72] J. Rabaey and M. Potkonjak, "Resource driven synthesis in the HYPER system," in *Proc. ISCAS-90*,New Orleans, LA, May 1990, vol. 4, pp. 2592–2595.
[73] J. Rabaey, C. Chu, P. Hoang, and M. Potkonjak, "Fast prototyping of data path intensive architecture," *IEEE Design and Test*, vol. 8, pp. 40–51, 1991.
[74] A. Sangiovanni-Vincentelli, "Editor's foreword," *Algoithmica*, Special issue: "Simulated Annealing," vol. 6, no. 3, pp. 295–302, 1991.
[75] N. Shenoy, R. K. Brayton, and A. Sangiovanni-Vincentelli, "Retiming of circuits with single phase transparent latches," in *Proc. IEEE Int. Conf. Computer Design*, 1991, pp. 86–90.
[76] K. Smith, and W. F. Appelbe, "PAT—An interactive FORTRAN parallelizing assistant tool", in *Proc. Int. Conf. Parallel Processing*, 1988, vol. II: Software, pp. 58–62.
[77] E. A. Snow, D. P. Siewiorek, D. E. Thomas, "A technology-relative

computer-aided design system: Abstract definition, transformations and design tradeoffs," in *Proc. 15th ACM/IEEE Design Autom. Conf.*, 1978, pp. 220–226.
[78] S. W. K. Tjiang and J. L. Hennessy, "Sharlit—A tool for building optimizers," in *Proc. ACM SIGPLAN' 92 Conf. Programming Language Design and Implementation*, San Francisco, CA, pp. 82–93.
[79] H. Trickey, "Flamel: A high-level hardware compiler," *IEEE Trans. Computer-Aided Design*, vol. CAD-6, pp. 259–269, 1987.
[80] L. Valiant, S. Skyum, S. Berkowitz, and C. Rackoff, "Past parallel computation of polynomials using few processes," *SIAM J. Computing*, vol. 12, no. 4, pp. 641–644, 1983.
[81] B. L. Van der Warden, *Modern Algebra* New Yrok: Frederick Ungar, 1950.
[82] R. A. Walker and D. E. Thomas, "Behavioral transformation for algorithmic level IC design," *IEEE Trans. Computer-Aided Design*, vol. 8, pp. 1115–1127, 1989.
[83] M. E. Wolf and M. S. Lam, "a loop transformation theory and an algorithm to maximize parallelism," IEEE Trans. Parallel and Distributed Systems, vol. 2, pp. 452–471, 1991.
[84] M. J. Wolfe, "The tiny loop restructuring research tool," in Proc. Int. Conf. Parallel Processing, 1991, vol. II, pp. 47–53.
[85] W. Wolfe, A. Takach, and T-C. Lee, "Architectural optimization methods for control dominated machines," in *High-Level VLSI Synthesis*, W. Wolfe and R. Camposano, ed. Norwell, MA: Kluwer Academic, 1991, pp. 231–254.
[86] H. P. Zima, H. J. Bast, and M. Gerndt, "Superb: A tool for semi-automatic SIMD/MIMD parallelization," *Parallel Computing*, vol. 6, pp. 1–18, 1988.
[87] H. P. Zima and B. Chapman, *Supercompilers for Parallel and Vector Computers*. New York: ACM Press, 1991.

Miodrag Potkonjak (S'90–M91) received the B. S. and M. S. degrees in electrical enginering from the University of Belgrade, Yugoslavia and the Ph. D. degree in electrical engineering and computer science from the University of California, Berkeley.

He has been a research staff member at Computer and Communication Research Laboratories, NEC USA, Princeton, NJ, since 1991. His main research interests include computer-aided analysis, synthesis and evaluation of system level designs, parallel and distributed computations and interaction between high performance application specific computations and communications.

Jan Rabaey (S'80–M'83–SM'92) received the E. E. and Ph. D. degrees ("summa cum laude") in applied sciences in 1978 and 1983, respectively, from the Katholieke Universiteit Leuven, Belgium.

In 1987, he joined the faculty of the University of California, Berkeley, where he is now a professor. His current research interests are the study of architectures, the computer aided analysis and the automated design of digital signal processing systems. He has authored or co-authored more than 100 papers in the area of signal processing.

In 1986, Dr. Rabaey received the 1985 IEEE TRANSACTIONS ON COMPUTER-AIDED DESIGN Best Paper Award (Circuits and Systems Society). In 1989, he received the Presidential Young Investigator award. He also was a recipient of the Analog Devices Career Development Professorship award. He has served as associate editor of the IEEE JOURNAL OF SOLID STATE CIRCUITS and he is and has been on the program committee of the ISSCC, EDAC, ICCD, ICCAD, High Level Synthesis, and VLSI Signal Processing Conferences.