CRITICAL PATH ANALYSIS

Critical path analysis and minimization are relevant considerations when designing digital circuits (e.g., application-specific integrated circuits, ASIC) to be incorporated in electronic products such as computer systems and telecommunication hardware. It should be mentioned that critical path analysis of digital circuit design is different from the critical path analysis method used in PERT networks, which is commonly applied to describe and control the workflow of engineering projects. The critical path analysis of a digital circuit may lead to the modification of the topology and structure of the circuit, as well as changes in the underlying algorithm that the circuit is implementing. These changes, when needed, are designed to decrease the overall critical path of the circuit, its implementation cost, and sometimes its power consumption.

The recent technological revolution in areas such as semiconductor processing, very large scale of integration (VLSI) circuit design, and computer-aided design (CAD) of electronic circuits motivated the development of several state-of-the-art methods for critical path analysis and minimization for digital circuit design. As a consequence of the present revolution in integration techniques, the area of an integrated circuit (IC) or chip is no longer the most critical design parameter. The throughput (i.e., the rate of processing of input samples) and the latency of a digital circuit (i.e., time needed to compute an output sample after all necessary inputs samples are available) have both become key design parameters. Figure 1 illustrates the concepts of latency and throughput of digital hardware in more detail, where the initiation interval is defined as the inverse of throughput of a digital circuit.

There is a direct relationship between the critical path (CP) of a digital circuit and the maximal possible throughput [i.e., the length of the critical path is a lower bound on the initiation interval (T_I) that is equal to the inverse of the throughput]. To find the critical path of a digital circuit with N input and M outputs and K delay elements (memory/registers), the following definitions are necessary:

1. A primary input (PI) is any one of the N inputs of the circuit, where N is greater or equal to 1.
2. A primary output (PO) is any one of the M outputs of the circuit, where M is greater or equal to 1.
3. Each output from a delay element of the circuit is called a pseudo primary input (PPI). A circuit without delay elements ($K = 0$) is called a memory-less (combinatorial) circuit and it has no PPIs.
4. Each input to a delay element of the circuit is called a pseudo primary output (PPO). A memory-less circuit has no PPOs.

The length of a path from a PI/PPI to a PO/PPO is measured by the number of clock cycles to perform the operations (e.g., additions, multiplications) in the path. The critical path of a circuit is the path with the longest length, where only paths without delay elements (memory/register) are considered (i.e., delay elements are either source or sink nodes to a path). The length of the critical path of a digital circuit can be computed, for example, by using variations of the all-pairs shortest paths algorithm, where each PI or PPI will be a source node and each PO or PPO a sink node. Figure 2 illustrates these concepts, where it is assumed that each operation is computed in one clock cycle for the sake of simplicity. Therefore, there is an understood delay after each operation and a corresponding delay in all parallel paths of the circuit.

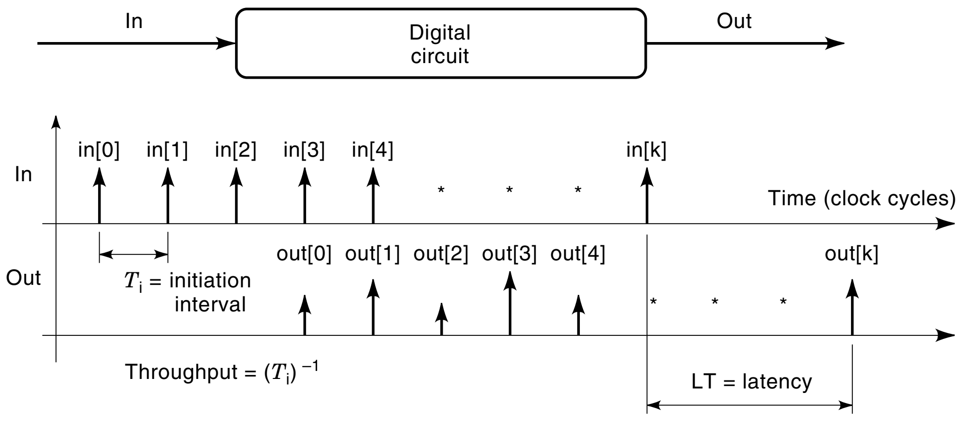

Figure 1. Latency and throughput of a digital circuit.

The latency (LT) of a digital circuit is a function of the length of the longest direct path from a PI to a PO, where explicit delay elements are included in the path. In this case the length of a path is measured as the product of the number of delay elements in the path times the initiation interval (critical path) measured in number of clock cycles. The latency of a memory-less circuit is equal to the critical path of the circuit.

Critical path minimization methods are, therefore, essential in throughput optimization. Proper module selection and scheduling are the basic design activities which apart critical path length. Module selection involves the assignment of modules (e.g., adders, multipliers) to each operation (e.g., additions, multiplications) in such a way that the length of the critical path length is traded against area and power consumption (i.e., faster modules usually require more area and power). Scheduling involves the proper time assignment of operations sharing the same module (i.e., deciding their order of execution whenever sharing of modules and other resources is a necessity due to constraints in the implementation cost [e.g., area]).

However, these two techniques have a limited efficacy because both assume that the algorithm underlying the digital circuit being designed is fixed. Increasing requirements for chips with higher throughputs and reasonable implementation costs have led to the development of algorithm transformation techniques (i.e., transformations to be applied at the high level, such as the register transfer level, RTL), which are representations of an algorithm specifying the circuit to be implemented. These transformations are widely recognized by the digital design community to be effective techniques in increasing the competitive advantages of the final chip design.

There is an abundance of published transformation techniques that efficiently make a tradeoff between latency and throughput (e.g., pipelining, Fig. 3). However, many real-time applications in areas such as flight, numerical control, robotics, and telecommunications have strict limits on the maximum allowed latency. This fact restricts the application of transformations, leading to additional latency in exchange for higher throughput.

In those situations, retiming (Fig. 3), a transformation technique introduced by Leiserson and Saxe, has proven to be a useful optimization technique to maximize throughput without an increase in latency. Retiming involves moving the delay (memory/register) elements in the circuit in such a way that the intended original computation is preserved, but parameters such as critical path length are optimized. However, the power of retiming is often limited by the structure of the computation to be mapped into the digital circuit being designed (i.e., by the particular description being used to specify the algorithm to be mapped in hardware). Also, algebraic and redundancy elimination transformations have been proposed several times, most often in the tree-height reduction framework, as an efficient technique for throughput optimization, without additional latency introduction. However, the power of algebraic and redundancy manipulation transformations is

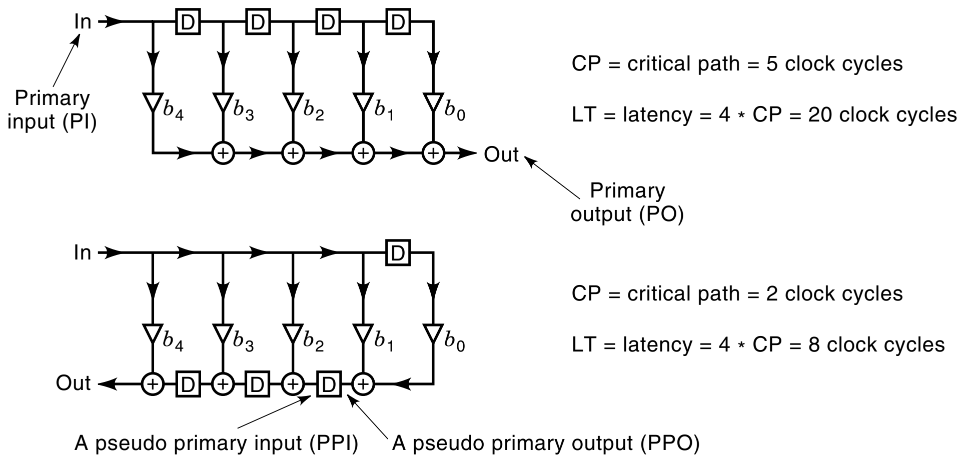
In
D
Primary input (PI)
b_4
b_3
b_2
b_1
b_0
Out
Primary output (PO)
CP = critical path = 5 clock cycles
LT = latency = 4 * CP = 20 clock cycles
CP = critical path = 2 clock cycles
LT = latency = 4 * CP = 8 clock cycles
A pseudo primary input (PPI)
A pseudo primary output (PPO)

Figure 2. Relationship among critical path, latency, and throughput of a circuit.

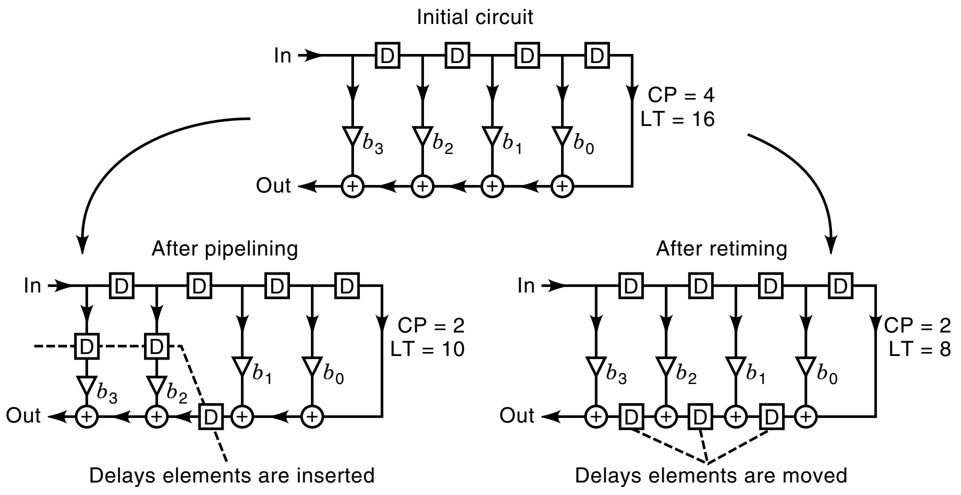

Figure 3. Pipelining and retiming.

limited by the initial position of delay (memory/register) elements. Only by targeting both functional restructuring of the computation to be mapped in hardware and retiming can the maximal possible throughput improvement be achieved. The linear predictor example (Fig. 4) illustrates this point, providing key insights about several issues addressed in this chapter. Figure 4(a) shows the original design, and Figs. 4(b), 4(c), 4(d), and 4(e) show variations in the computation structure.

There has been rapid evolution of semiconductor processing technology, improvements in integrated devices design methodology, and a widespread availability of CAD tools in the last two decades. This has created a climate in many application domains in which the importance of area as the most critical parameter of a design has been reduced. This (r)evolution has simultaneously elevated the optimization of throughput to the position of a key design parameter. The majority of optimization problems associated with throughput optimization in high-level synthesis are trivial as long as the implementation cost (e.g., area) is not a factor. For example, during module selection it is obviously sufficient to select the fastest components; in scheduling, any proper technique that honors the critical path is equally good for this goal. However, improvements due to this type of optimization are often not sufficient to meet ever-increasing requirements for higher throughput. In this situation, transformations have been widely recognized as an efficient way to provide a competitive advantage in design.

ALGORITHMIC TRANSFORMATIONS AND RETIMING

For simplicity, we will assume that each operation (either a multiplication with constant denoted as a triangle on the figures or an addition) in the linear predictor example takes one control cycle. The initial critical path is nine control cycles long [see Fig. 4(a)]. It is easy to conclude that retiming is not effective on this example and cannot reduce the critical path, because no delay can be moved along the critical path from the input through multiplication with constant b_4 and the chain of additions to the output. Associativity and commutativity, as used in several tree-reduction-based techniques, are slightly more effective: The critical path can be reduced to seven control cycles, as shown in Fig. 4(b).

However, if the transformation steps are applied, as shown in Figs. 4(c) to 4(e), we can obtain a significantly more drastic reduction in the length of the critical path. The idea is to identify the specific reasons why retiming is ineffective and how the structure of the computation has to be transformed, and then directly target and restructure those places in the control data flow graph structure, using algebraic transformations, so that retiming for critical path reduction is enabled. As we already mentioned, although there are eight delays in the linear predictor example, none of them can be used to reduce the critical path. The reason behind this situation is the fact that there is no delay on edges x and y, so no delay can be moved across the addition operation to the critical path. In this situation associativity and commutativity may help. We applied associativity and commutativity on two smaller chains of additions in the example. This application of associativity and commutativity can be denoted, respectively, in algebraic form as ($\alpha_1 + \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 = \alpha_2 + \alpha_3 + \alpha_4 + \alpha_5 + \alpha_1$) and ($\alpha_6 + \alpha_7 + \alpha_8 + \alpha_9 + \alpha_{10} = \alpha_7 + \alpha_8 + \alpha_9 + \alpha_{10} + \alpha_6$). The critical path is unchanged nine control steps long. However, now retiming is enabled and it reduces the critical path to four control steps. We see that associativity and commutativity are better utilized in this way than in the tree height minimization framework, although they did not directly reduce the critical path at all. The retiming that transforms the structure in Fig. 4(c) to the structure shown in Fig. 4(d) can be done in polynomial time using the Leiserson-Saxe retiming algorithm for critical path reduction.

The two-phase process, where algebraic transformations enable retiming, can be iterated to increase throughput further. This iteration brings even further throughput increase. After the application of associativity and commutativity for algebraic speedup (explained later in this article), we obtain the structure shown in Fig. 4(e). The critical path is reduced to three cycles. The Leiserson–Saxe retiming algorithm cannot bring an additional improvement but can reduce the number of delay elements, which correlates well with register requirements. So total improvement by a factor of 3 is achieved using retiming, associativity, and commutativity in the linear predictor example.

As we already saw in the iterated application of the basic procedure, in the general case, it will not be sufficient to use only associativity and commutativity in order to enable retiming. Also, although it is possible to address this small example manually, when the examples are large it is not possible to use simple ad hoc analysis to identify goals during the algebraic manipulation and redundancy manipulation phase.

The approach presented in this article generalizes the ERB (eliminating retiming bottlenecks) (1) method from logic synthesis research, which shows how to identify conditions that are sufficient to be met using algebraic and redundancy manipulation techniques so that retiming can reduce the critical path length provably in the transformed computational structure. A key component of the approach is the algebraic speedup technique, which combines the power of all algebraic and redundancy manipulation transformations to maximize the probability that an arbitrary set of required and arrival times are satisfied. An iterative improvement framework significantly enhances the power of the initial ERB algorithm.

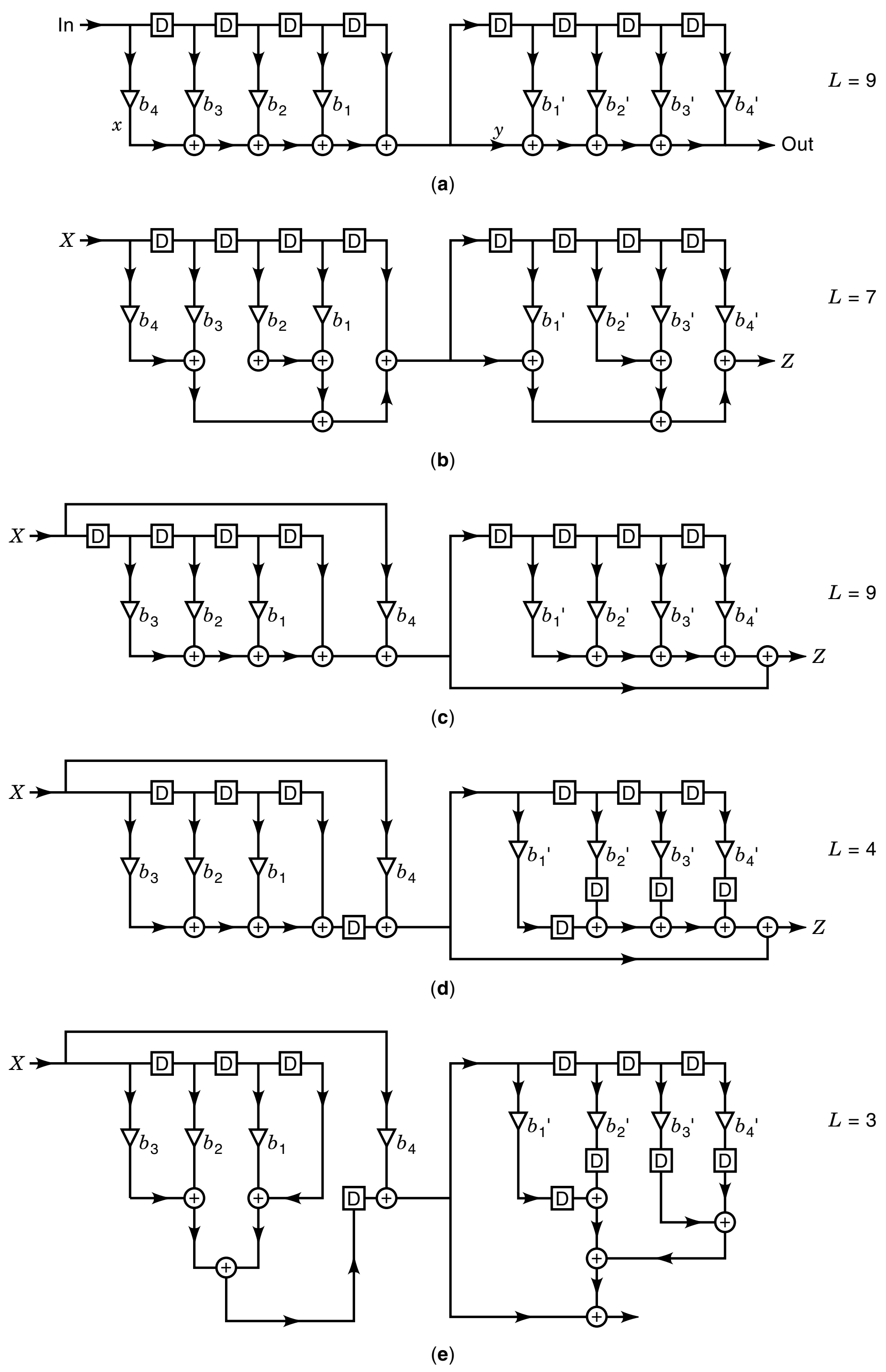

Figure 4. The linear predictor example: (a) the initial CDFG; (b) CDFG after application of tree height reduction; (c) after application of associativity and commutativity; (d) after retiming; (e) after reiterated application of algebraic transformations.

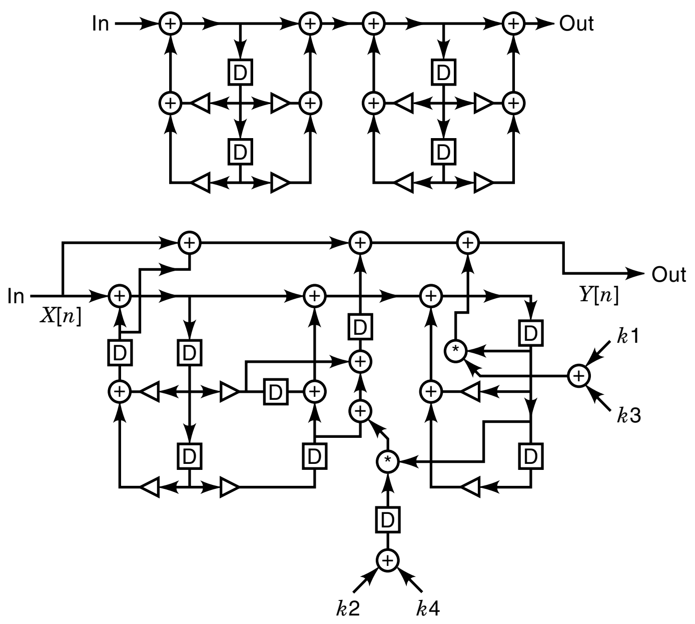

Figure 5. Fourth-order IIR filter: (a) The initial structure, (b) the final structure after the application of an iterative ERERB technique.

STATE OF THE ART IN TRANSFORMATION APPROACHES FOR HIGH-LEVEL SYNTHESIS

Recently, transformations have been rapidly establishing themselves as one of the most effective ways to improve a design's parameters. Transformations have been investigated and regularly used for performance enhancement is several engineering areas, most vigorously in software compilers and logic synthesis. Optimizing software compilers usually employ a set of transformations for speed improvement and code size reduction. More recently, compilers for parallel processing systems have been augmented with a set of transformations that explicitly target concurrency exploration. Detailed surveys of most popular transformations in compiler research and engineering can be found in diverse publications (2,3).

Transformations in logic synthesis are most often studied in two frameworks, combinational and sequential optimization. Combinational optimization uses algebraic, Boolean, and redundancy manipulation techniques, while sequential optimization augments the set of transformations with retiming. Besides the use of the Leiserson–Saxe algorithm for the minimization of the critical path and the number of delay elements, other important sequential optimization work includes research done by De Micheli (4), in which a subset of combinational transformations is combined with retiming; Bartlett (5), which addressed retiming of sequential circuits with level sensitive latches; Malik (6), who used peripheral retiming for optimization of pipelined circuits; and Shenoy (7), who extended retiming to cover circuits with level-sensitive latches. Finally, recently Dey (1) introduced the ERB algorithm, which efficiently uses combinational optimization to enable more effective retiming. See Fig. 5.

Although several high-level synthesis papers that describe research done in late 1970s addressed transformations (8–10), the use of transformations gained impetus recently. The most elaborate sets of transformations are given in several references (11–15). Trickey (11) strongly concentrated on several transformations in the Flamel system, including five variants of loop merging, loop unrolling, and tree height reduction and constant folding. He targeted the minimization of the critical path under area constraints, and although his optimization strategy was mainly greedy, Flamel achieved excellent experimental results on a benchmark set of 15 examples.

Walker and Thomas used in-line expansion, dead code elimination, several types of select transformations (conditional transformations in which code is moved across boundaries imposed by the control structure), and pipelining during preprocessing for both structural and behavioral partitioning in the System Architecture Workbench (SAW) (16). Haroun, in SPAID (13), used retiming for critical path reduction, pipelining, interleaving, substitution of multiplication by a constant with shifts and additions, associativity, commutativity, and distributivity. Hyper (15) combines more than 20 basic transformations in an optimization-intensive framework for optimization of both throughput and area. See Fig. 6.

Hyper-LP (16) modifies this framework and, by exploring the relationship between power consumption and transforma-

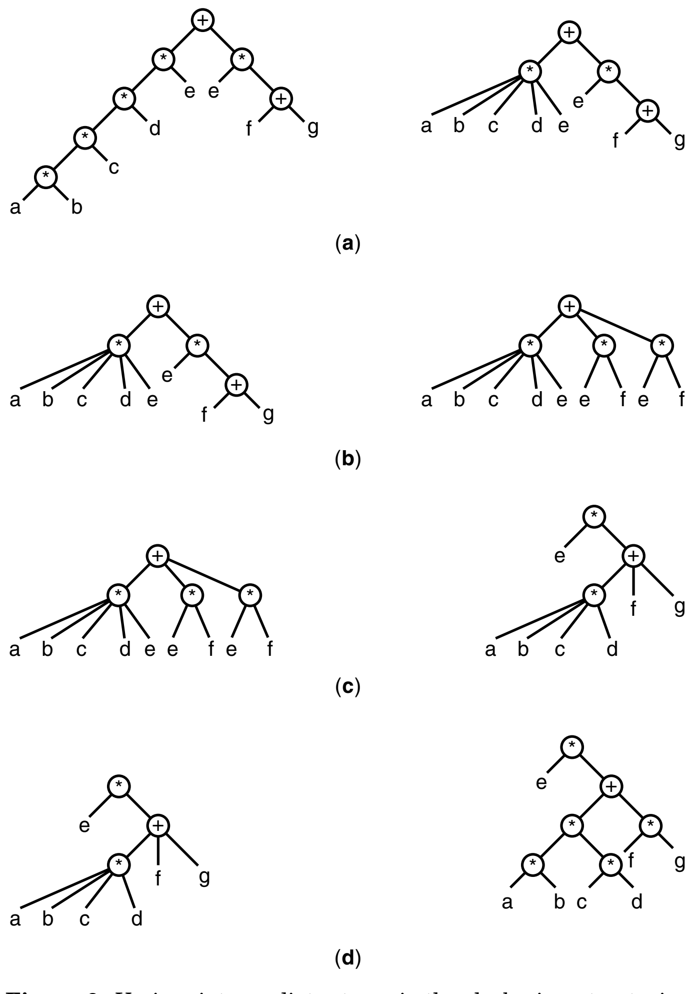

Figure 6. Various intermediate stages in the algebraic restructuring process: (a) common level extraction, (b) application of distributivity, (c) common subexpression elimination, and (d) application of associativity.

tions, achieves an order of magnitude improvement on a large set of examples. Several researchers, including Gyrczyc (17), Devadas (18), Goossens (19), and Hwang (20), addressed software pipelining during high-level synthesis. Wolf (21) addressed the use of several transformations in high-level synthesis for control-dominated designs. Functional pipelining, due to its extraordinary power to change dramatically the quality of design (in particular when there are no feedback loops), received special attention in high-level synthesis. Most often pipelining is addressed separately from the majority of transformations (note that since it introduces additional latency it is actually not a transformation in the strict sense). Several excellent in-depth studies of many important aspects of pipelining are available (22,23).

Retiming

Retiming is a computation technique in which delays are added at some places and removed from others in a such way that the input-output relationship of the circuits is not altered (24,25). Initially, retiming in high-level synthesis was strictly used for the optimization purposes (critical path and register minimization) outlined in the seminal Leiserson–Saxe paper (26,27). Initially, the Hyper system (15) addressed retiming for resource utilization by using a probabilistic sampling algorithm (28). Later, the same optimization algorithm and framework were generalized so that other transformations, such as associativity, commutativity, and inverse element law, are efficiently handled (29).

Algebraic and Redundancy Manipulation Transformations

Due to their simplicity of application, associativity and commutativity are two most often applied algebraic transformations in high-level synthesis. Several researchers applied them in the tree height reduction framework (27). Interestingly enough, common subexpression replication got earlier and more extensive coverage (30,31) than common subexpression elimination. Recently Potkonjak (31) showed an approach that guarantees the optimal application of all algebraic and redundancy manipulation transformations for critical path reduction. However, the application domain was restricted to the class of linear computations. Though that paper presented a heuristic that efficiently treats an arbitrary computation, the present approach has a significant advantage since it combines the power of algebraic transformations with retiming. Also, the proposed algebraic speedup algorithm is more general in the sense that it can be used not only to minimize the critical path, but also to satisfy an arbitrary specification of required and available times.

EXTENDING THE ERB TECHNIQUE TO HIGH-LEVEL SYNTHESIS

The ERB technique to enable retiming by changing the computational structure was introduced and applied in the logic synthesis domain by Dey (1). This section shows an application of the technique to minimization of the critical path of digital circuits in the domain of high-level synthesis.

Differences between High-Level Synthesis and Logic Synthesis

Although there are striking similarities in optimization tasks done in the logic synthesis and high-level synthesis domains, there has been surprisingly little effort to explore this relationship by adapting methods of one area and applying them to another. The major reason is that regardless of the mentioned similarities, there are also several sharp differences in computations done in the logic synthesis domain and the high-level synthesis domain, which prevent direct sharing of methodologies and give a different flavor to the problems. We discuss some of the similarities and differences between the two domains that need to be addressed to adapt a performance optimization technique like ERB from logic synthesis to high-level synthesis.

1. Logic synthesis techniques like ERB (eliminating retiming bottlenecks) are usually applied to Boolean networks or gate-level circuits, while high-level synthesis techniques are most commonly applied to control data flow graphs (CDFG). While the nodes of CDFGs represent algebraic operations and conditional operations, the nodes of logic circuits represent Boolean functions. The delay elements in a CDFG represent iteration boundaries, whereas the corresponding latches in a logic circuit are physical entities representing storage elements. Note that the values of both delays and latches represent the state of the computation. The structure of logic circuits and CDFGs are usually very different. A CDFG may usually display a high level of regularity and modularity. Many logic circuits, in contrast, are either highly complex and random, as in control circuits, or the information about regularity and structure is lost at the logic level.
2. The length of a path in a data flow graph is the number of cycles required to execute the operations along the path. The length of a path in a logic circuit reflects the time taken by a signal to propagate along the path. The length of a critical path in a CDFG determines the sampling period of the design. The length of a critical path of a sequential circuit determines the clock period of the circuit. Consequently, the performance goal of high-level synthesis is to optimize the sampling period of a given design, while the goal of logic synthesis is to minimize the clock period of a sequential circuit.
3. During the application of optimizing transformations in logic synthesis and high-level synthesis, the delay models are different. Gates in logic circuits can have more than two inputs, while most of the operations in CDFGs are assumed to be either unary or binary. While the simplest delay models of both domains consider each operation (node) to have unit delay, each domain has more elaborate delay models. High-level synthesis algorithms recognize the fact that fixed-point multiplication, for example, takes more units of time than fixed-point addition. Elaborate delay models for gate-level circuits, on the other hand, have to consider factors like the number of fanouts and the output load of a gate, not usually considered during high-level synthesis.
4. While hardware sharing is a standard methodology in high-level synthesis, it is not used in logic synthesis.
5. The functional operations in logic synthesis obey both Boolean and algebraic axioms. While algebraic axioms are also valid in high-level synthesis, Boolean laws are not applicable. Also, the different nature of primitive

operations in the two domains dictates sometimes different transformations.

6. Retiming is the only common transformation used in logic synthesis to exploit the sequential behavior of the circuit. In high-level synthesis, a variety of loop and conditional transformations are used, besides retiming.
7. Increase in latency is sometimes allowed in high-level synthesis to achieve various synthesis goals. In logic synthesis, latency is usually considered fixed.

The ERB Technique

The basic idea of the ERB technique is to identify the bottlenecks that prevent retiming from achieving the desired clock period of a sequential circuit, and eliminate the bottlenecks by satisfying a set of sufficient conditions. The sufficient conditions are satisfied using combinational transformations. Eliminating the bottlenecks ensures that retiming is enabled to achieve the desired clock period. In this section, we briefly describe the ERB technique, modified to be applicable to CDFGs in the high-level synthesis domain.

Retiming Bottlenecks. Let $d(p)$ denote the length of a path p or, equivalently, the number of cycles required to execute all the operations along path p. Let $w(p)$ denote the number of delays on the path p. Let L be the length of the critical path in the CDFG. Let the desired reduction in the critical path length be r. In the following discussion, we will use path to denote a simple path with no delays.

Retiming cannot achieve the desired critical path length $(L - r)$ if and only if at least one of the following retiming bottlenecks exists in the CDFG (1):

B1. A path p from PI to PO, such that $d(p) > (L - r)$

B2. A path p from PI to PO, such that $d(p) > (L - r)*(w(p) + 1)$, where $w(p) > 1$

B3. A critical loop C, such that $d(C) > (L - r)*w(p)$, where $w(p) > 0$

Sufficient Conditions for Eliminating Retiming Bottlenecks. Eliminating the retiming bottlenecks from a CDFG would enable retiming to achieve the desired critical path length. However, there may be an exponential number of retiming bottlenecks in a CDFG. Enumerating each bottleneck and eliminating it explicitly may be prohibitive. Instead, we identify a set of conditions that, if satisfied, eliminate all the retiming bottlenecks simultaneously.

A delay X is a forward slack delay (FSD) if the slack at its input, $s_i(X)$, is greater than the desired reduction r. An FSD X can be moved forward by a distance of $(s_i(X) - r)$ without making any path to X longer than the desired $(L - r)$. Moving FSD X forward by $(s_i(X) - r)$ increases the lengths of paths to X by $(s_i(X) - r)$ and decreases the lengths of paths from X by $(s_i(X) - r)$.

Similarly, a latch whose output slack, $s_o(X)$, is greater than r can be moved backward by $(s_o(X) - r)$ without making any path from X longer than the desired $(L - r)$. However, to guarantee that the final circuit after retiming has an equivalent initial state, we choose to use only the forward movement of FSDs during retiming. We derive a set of conditions on the paths of the circuit such that satisfying the conditions ensures that the FSDs in the transformed circuit can be moved forward (retimed) to achieve the desired critical path length $(L - r)$.

Let the inputs (outputs) of the CDFG be partitioned into those with FSDs and all other inputs (outputs) that do not have forward slack delays (NFSD). If a path beginning with an FSD, X [whose movement can decrease its length by $(s_i(X) - r)$] and ending with an FSD, Z [whose movement may increase its length by $(s_i(Z) - r)$] has length no greater than $(L - r + (s_i(X) - r) - (s_i(Z) - r))$, the length of the path can be reduced to the desired $(L - r)$ by moving delay X forward during retiming. Similarly, any path beginning with an FSD, X, and ending with an NFSD must have length no greater than $(L - r + (s_i(X) - r))$. Any path beginning with an NFSD and ending with an FSD, Z, must have length no greater than $(L - r - (s_i(Z) - r))$. Finally, paths between NFSDs must have length no greater than $(L - r)$, because retiming may not be used to reduce these paths. The following conditions are applicable to the CDFG before retiming:

B_{FF}. All paths from any FSD X to any FSD Z have length no greater than $(L - r + s_i(X) - s_i(Z))$.

B_{FN}. All paths from any FSD X to any NFSD j have length no greater than $(L + s_i(X) - 2r)$.

B_{NF}. All paths from any NFSD i to any FSD Z have length no greater than $(L - s_i(Z))$.

B_{NN}. All paths from any NFSD i to any NFSD j have length no greater than $(L - r)$.

Timing Constraints to Enable Retiming. In this section, we outline our approach to satisfy the conditions B_{FF} to B_{NN} to eliminate the retiming bottlenecks and enable retiming. We identify a set of nodes $\{v\}$, called cnodes, such that each path longer than $(L - r)$ has a cnode on it. We extract the cone of each cnode v, cone(v), which is the part of the CDFG comprising all paths from cnode v to the PI/PPIs. Our approach to satisfy conditions B_{FF} to B_{NN} is to identify the set of cnodes and satisfy a set of timing constraints for the corresponding cones.

For each cnode v, the timing constraints for cone(v) are specified in terms of the arrival times of its inputs (PIs and PPIs) and the required time of its output v.

For each input X of cone(v), the new arrival time $a(X)$ is as follows:

$$\hat{a}(X) = \begin{cases} 2r - s_i[X] \text{ is a forward slack delay (FSD) input} \\ r \text{ otherwise} \end{cases}$$

The new required time for cnode v is set to be its original required time. If the timing constraints are satisfied, all nodes and edges in cone(v), which are not in the transitive fanin of any other node besides v, are eliminated. The cone for v is now replaced by a new CDFG cone *Scone*(v), which satisfies the constraints. This operation preserves the functionality of the circuit.

The algorithm to identify the appropriate cnode and satisfy the timing constraints of the cnode is outlined in Ref. 1. The timing constraints of each cone are satisfied by an algorithm using several algebraic transformations, and will be described in the next section.

It has been shown that for a set of cnodes $\{v\}$ that forms a cutset of the paths longer than $(L - r)$, if the timing constraints are satisfied for each cone(v), then the conditions B_{FF} to B_{NN} are satisfied (1). Hence, satisfying the timing constraints of all the cones ensures that all retiming bottlenecks have been eliminated, and a subsequent retiming step is enabled to achieve the desired critical path length ($L - r$). During the application of the ERB algorithm, the maximal value of r is determined by using binary search.

Iterative High-Level Synthesis ERB Algorithm. From the seven aforementioned differences that influence the need for a different ways for the application of transformations in high-level synthesis and logic synthesis, hardware sharing, loop, and conditional transformations do not represent a problem during the application of the ERB algorithm in high-level synthesis. Also, since our primary goal is throughput optimization without introduction of additional latency, pipelining is not an attractive option. However, the different delay models, and in particular the different nature of functional transformations applicable in the two CAD domains, requires extensive modifications and enhancement of the logic synthesis ERB algorithm for application to high-level synthesis. The difference in types of applicable transformations required the development of a new algebraic speedup technique for CDFGs, which can satisfy given timing constraints. The algebraic speedup technique, which is described in the next section, checks and ensures during its application that retiming is not disabled due to nonunit delay of various types of operations.

Both the more regular and less involved structure of a CDFG compared to logic circuits make iterative application of the ERB algorithm feasible. As shown in the linear predictor example and in the next section on the truncated Volterra filter, iterative application of the ERB is beneficial. Due to regularity, the iterative ERB improved results significantly on all tested examples.

Finally, note that the iterative ERB algorithm can be adapted easily to treat combination functional pipelining with algebraic and redundancy manipulation transformations. A simple, well-known formulation of pipelining such that pipelining with N stages is equivalent to retiming, where the number of delays on all inputs is increased by N, directly implies that the only needed modification is to add an appropriate number of delays to all inputs. The simple preprocessing step, the application of the Leiserson–Saxe algorithm for critical path reduction, most often significantly reduces the number of needed iterations of the ERB algorithm for pipelining.

THE ERB-BASED ALGEBRAIC SPEEDUP ALGORITHM

To enable retiming, the ERB technique requires satisfaction of timing constraints on parts of the CDFG. In this section, we present a technique that uses an ensemble of different algebraic transformations to satisfy an arbitrary set of timing constraints on the CDFG. The technique can also be used to optimize the throughput of a CDFG.

The algorithm takes as input a dataflow graph along with the arrival times for the primary and pseudoprimary inputs, and tries to restructure the computation to meet the required time constraints of the outputs. The restructuring technique may even lead to an increase in the critical path length if such an increase meets the required time constraints (see the truncated Volterra example).

The algorithm can also be used as a critical path reduction technique. In this case the inputs are assigned unit arrival times and the algorithm tries to minimize the critical paths to get the least arrival times at the outputs.

The arrival and required times are assigned by the retiming timing constraints. The critical inputs are those that must be moved forward during algebraic resynthesis, and they are assigned higher (later) arrival times. The other inputs are assigned lower (earlier) arrival times, providing the algorithm freedom for restructuring the CDFG. Also, the common factors in the computation are pushed to a later stage in the computation process to make use of common subexpression elimination.

The algorithm begins by constructing an expression tree for the dataflow graph that preserves the order of computations in the dataflow graph. The next step is the extraction of common levels of the computation. This process helps to evaluate the level of flexibility in the dataflow graph and the amount of freedom available to schedule the critical inputs.

The next step is the application of distributivity to the dataflow graph, which further increases the chances for common subexpression elimination and the scheduling of the critical inputs. At this stage, we are able to make a tradeoff between the choice of signals (inputs) for common subexpression elimination and the critical inputs that directly affect the required times of the outputs. The critical signals take preference as they determine the satisfaction of the arrival (required) timing constraints, so in the case of a tie the critical inputs usually get the preference. However, if a candidate for common subexpression elimination still enables the timing requirements of the output to be met, then it is chosen, as the process eventually leads to a reduction of operations in the dataflow graph.

The common subexpression elimination process involves identifying inputs that are common to multiple operations at the same level of computation and moving them to the next higher level. This process results in a considerable reduction in the number of operations in the CDFG. Consider, for example, an input that appears in three different operations at the same level of computation. Since operation will eventually have only two inputs, the identification and subsequent transfer of the input to the next level of computation results in the reduction of instances of the particular operation from three to one.

After applying level reduction, distributivity, and common subexpression elimination, associativity is applied to restructure the CDFG into two-input operations, keeping in view the arrival and required time constraints. In the case of applying the algorithm to reduce critical paths, this process simply restructures the various inputs at the same level of computation as balanced binary trees so as to minimize their height. In the case where the goal is to satisfy a set of arrival and required time constraints, the inputs are restructured according to their arrival times: The inputs with earlier arrival times (having maximum freedom) are assigned first and the inputs with the latest arrival times (least freedom) are assigned in the end so as to meet the required time constraints of the output signals.

The truncated Volterra filter example serves to illustrate effectively the application of the ERB algorithm and the alge-

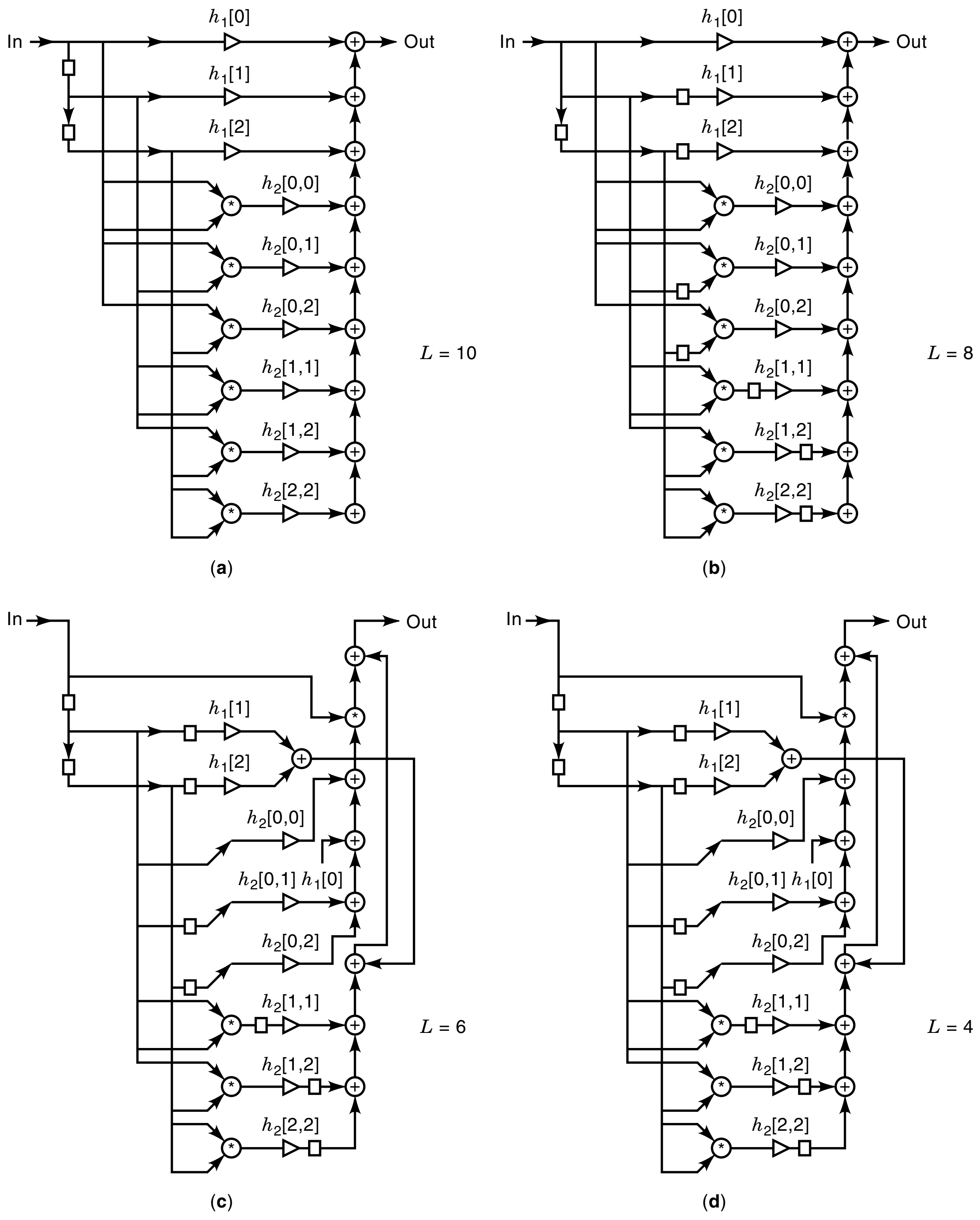

Figure 7. Truncated Volterra filter: the application of iterative ERB algorithm; (a) initial; (b) after application of retiming; (c) after algebraic speedup; (d) after repeated retiming; (e) after second algebraic speedup; (f) final structure.

braic restructuring algorithm on a real-life example. The particular example was chosen to illustrate effectively the process of restructuring of logic and the computation of the arrival time constraints that meet the retiming requirements.

The original example, shown in Fig. 7(a), has a critical path of 10 and contains 15 multiplications and 8 additions. The example, when retimed, has its critical path reduced to 8, as shown in Fig. 7(b). Retiming cannot improve the critical path further because of the presence of a path from primary input to the primary output of length 8.

The ERB algorithm applied to the graph with an *r* (desired reduction) value of 4 produces the dataflow graph as shown in Fig. 7(c). Note that although the critical path has been reduced by 2, this was by no means the objective of the transformation. The transformation was aimed at, and actually does, enable the CDFG for further retiming. Fig. 4(d) shows the

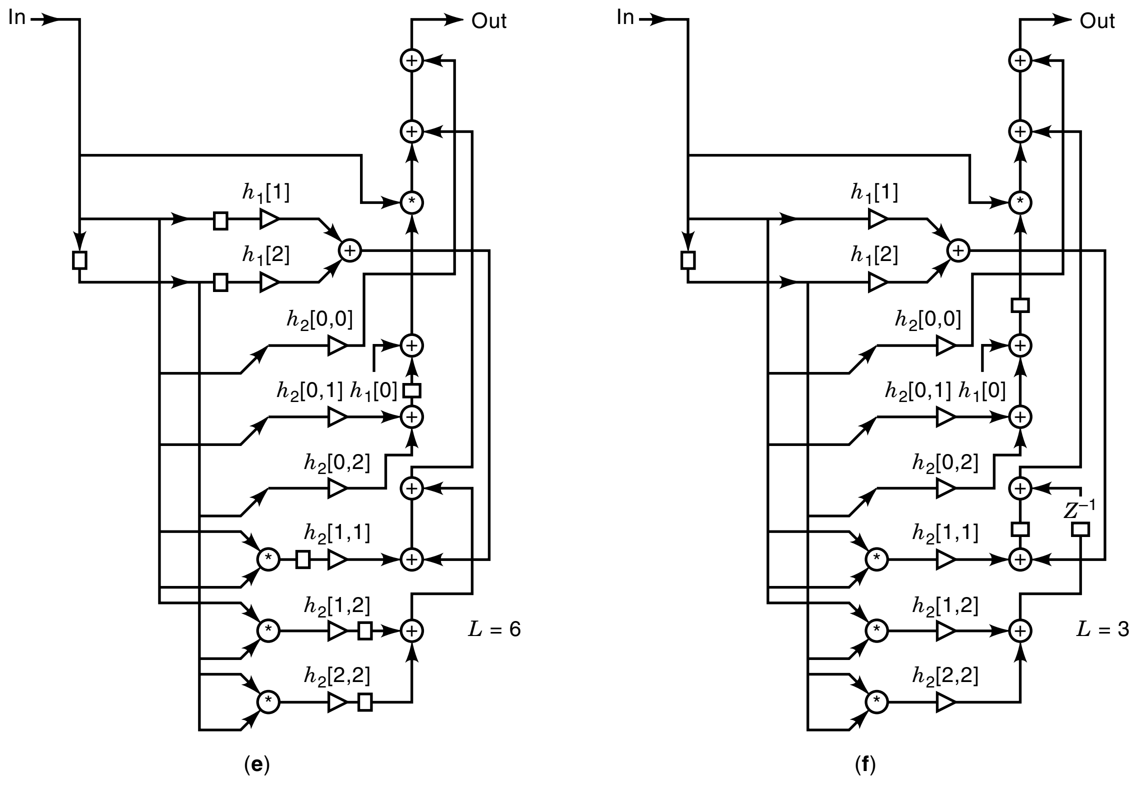

Figure 7. (*Continued*)

retimed flowgraph with a critical path of 4. It is worth noting that the transformed CDFG, Fig. 4(c), has 12 multiply operations as opposed to 15 in the original example, and 7 addition operations as compared to 8 in the original examples. The number of delays in the transformed graph [Fig. 7(c)] remains the same, though retiming reduces the number of delays by 1. Thus the increase/decrease in the number of delay elements in the dataflow graph is purely a function of the retiming process.

The CDFG shown in Fig. 7(d) has multiple paths from the primary inputs to the primary output and from delays to the primary output, which restrict the retiming algorithm from further optimizations in critical path length. The application of the ERB algorithm (with r of 1) on the CDFG represented by Fig. 7(d) yields a transformed graph, as shown in Fig. 7(e). The transformations in this particular case actually increase the length of the critical path from 4 to 6 while increasing the number of multiplication from 12 to 13, whereas the number of addition operations goes up from 7 to 8. Although the transformed flowgraph has a critical path length of 6, it guarantees the reduction of critical path by the r value of 1. Upon retiming, the resultant flow graph has a critical path of 3, as guaranteed by the ERB algorithm.

Hence, the final CDFG, shown in Fig. 7(f), has a critical path length of 3 as compared to the optimally retimed CDFG with critical path of length 8, and has 13 multiplication operations as compared to 15 in the original example. The reduction in the number of multiplication operations was made possible due to common subexpression elimination by the algebraic speedup technique while satisfying timing constraints. See Figs. 8 and 9.

EXPERIMENTAL RESULTS

Table 1 shows a set of six examples on which the proposed technique was tested. All examples properly simulate before and after the application of the iterative ERB algorithm. IIR filters are used in speech processing, the linear predictor is often used as generalization of FIR filters, nonlinear second-order Volterra filters are common in telecommunications (in particular for echo cancellation), and eighth-order Avenhaus bandpass filters are a standard benchmark in the digital filter design community. We excluded the popular fifth-order wave digital elliptical filter from benchmark suite since the proper set of coefficients was not available for simulation.

The average improvement in the length of the critical path in the final design was by a factor of 2.4 times. The average improvement over Leiserson–Saxe retiming is by a factor of

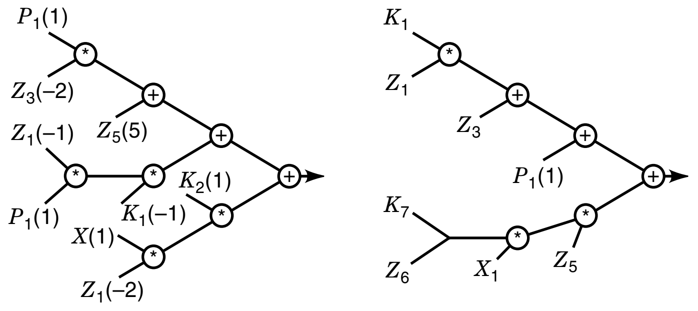

Figure 8. Application of algebraic transformation technique for satisfying arrival times (inputs) and required time (outputs) constraints.

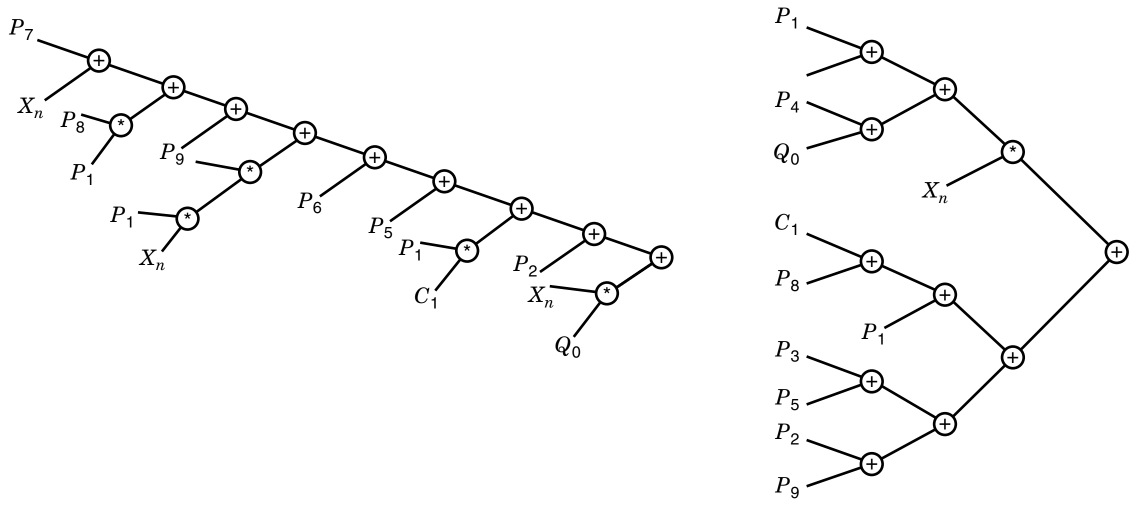

Figure 9. Application of algebraic restructuring technique for minimization of critical path.

2.16. This validates our initial assertion that a technique that utilizes the relationship between retiming and algebraic and redundancy manipulation techniques has a significant advantage over retiming alone. Recently, it has been shown that dramatic improvement in both throughput and parallelism is often correlated with a significant increase in the number of operations (29,30,32). Potkonjak and Rabaey (29) showed the surprising result that the throughput can be efficiently traded for latency, with no hardware overhead. The experimental results shows that even when latency is not introduced, the product of the number of operations and the critical path length (NC product), and therefore also AT (area–time) product, can be improved simultaneously with the throughput. Notice that the AT product is directly proportional to the product of the number of operations and the length of the critical path, for designs with no hardware sharing. When hardware sharing is involved, the correlation between the products is more involved, but still very strong. As shown in Table 1, the NC product is improved on average by 129%.

The average improvement in the length of the critical path in the final design was by a factor of 2.4 times. The average improvement over the Leiserson–Saxe algorithm is by a factor of 2.16. This validates our initial assertion that a technique that utilizes the relationship between retiming and algebraic and redundancy manipulation techniques has a significant advantage over retiming alone. The increase in the number of operations is rather small, only 22% and 6% for additions and multiplications, respectively.

FUTURE WORK AND CONCLUSIONS

An iterative ERB algorithm for critical path reduction in high-level synthesis was presented, using an algebraic speedup approach combining the power of all algebraic (associativity, commutativity, distributivity, zero, and identity elements laws) transformations with redundancy manipulation transformations. Experimental results show that the iterative ERB algorithm supported by an algebraic speedup subroutine outperforms the Leiserson–Saxe retiming algorithm by a factor greater than 2, with very small increase in hardware requirements, so that in all examples the product of the number of operations and the length of the critical path is improved significantly simultaneous with the throughput.

Many interesting and influential ideas come from compiler research and technology to both logic synthesis and high-level synthesis. Interestingly, previously no effort was made to explore the relationship between logic synthesis and high-level synthesis. Since the algebraic speedup algorithm is capable of addressing an arbitrary set of required and arrival times, it is expected that the modified ERB algorithm will find a wide range of application not only with retiming, but with other

Table 1. Experimental Results

	Initial			Retimed			Iterative ERB		NC Product	
Example	A	M	CP	CP	A	M	CP	I	R	ERB
4th-Order IIR filter	8	8	6	4	13	8	3	96	64	63
Adaptive IIR filter (4th Order)	8	8	6	6	10	8	5	96	96	90
Linear predictor	8	8	9	9	12	12	3	144	144	72
Volterra filter	10	17	12	12	10	13	4	324	324	92
Truncated Volterra filter	8	15	10	8	8	13	3	236	184	63
8th-Order Avenhaus bandpass filter	16	13	10	9	16	19	5	290	261	175

A—number of additions, M—number of multiplications, CP—critical path, NC—product number of operations and critical path. Note that retiming does not change the number of operations.

loop and conditional transformations. Also, the first experiments demonstrate the strong potential to solve, using algebraic speedup and the iterative ERB algorithm, not only critical path reduction but also other optimization objectives in high-level synthesis, such as area and power minimization problems.

BIBLIOGRAPHY

1. S. Dey, M. Potkonjak, and S. Rothweiler, Performance optimization of sequential circuits by eliminating retiming bottlenecks, *Proc. ICCAD-92,* paper 10C-1, 1992.
2. A. V. Aho and J. D. Ullman, *Principles of Compiler Design,* Reading, MA: Addison-Wesley, 1977.
3. C. N. Fischer and R. J. Le Blank, *Crafting a Compiler,* Menlo Park, CA: Benjamin/Cummings, 1985.
4. G. De Micheli, Synchronous logic synthesis: Algorithms for cycle-time minimization, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.,* **10**: 63–73, 1991.
5. K. Bartlett, G. Borriello, and S. Raju, Timing optimization of multiphase sequential logic, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.,* **10**: 51–62, 1991.
6. S. Malik et al., Retiming and resynthesis: Optimizing sequential networks with combinatorial techniques, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.,* **10**: 74–84, 1991.
7. N. Shenoy and R. K. Brayton, Retiming of circuits with single phase transparent latches, *Int. Workshop Logic Synthesis, MCNC,* Research Triangle Park, NC, 1991.
8. E. A. Snow, D. P. Siewiorek, and D. E. Thomas, A technology-relative computer-aided design system: Abstract definition, transformations and design tradeoffs, *15th ACM/IEEE Des. Autom. Conf.,* 1978, pp. 220–226.
9. A. E. Casavant, D. D. Gajski, and D. J. Kuck, Automatic design with dependence graphs, *17th ACM/IEEE Des. Autom. Conf.,* 1980, pp. 506–515.
10. M. C. McFarland and A. C. Parker, An abstract model of behavior for hardware descriptions, *IEEE Trans. Comput.,* **32**: 621–636, 1983.
11. H. Trickey, Flamel: A high-level hardware compiler, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.,* **6**: 259–269, 1987.
12. R. A. Walker and D. E. Thomas, Behavioral transformation for algorithmic level IC design, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.,* **8**: 1115–1127, 1989.
13. B. S. Haroun and M. I. Elmasry, Architectural synthesis for DSP silicon compilers, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.,* **8** (4): 431–447, 1989.
14. J. Bhaskar and H. Lee, An optimizer for hardware synthesis, *IEEE Des. Test Comput.,* **7** (5): 20–36, 1990.
15. J. Rabaey et al., Fast prototyping of data path intensive architecture, *IEEE Design and Test,* **8** (2): 40–51, 1991.
16. A. P. Chandrakasan et al., Hyper-LP: A design system for power minimization using architectural transformations, *IEEE Int. Conf. Comput.-Aided Des.,* paper 6c.3, Santa Clara, CA, 1992.
17. E. Gyrczyc, *Automatic generation of microsequenced data paths to realize ADA circuit description,* Ph.D. thesis, Carleton Univ., 1984.
18. S. Devadas and A. R. Newton, Algorithms for hardware allocation in data path synthesis, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.,* **8**: 768–781, 1989.
19. G. Goossens et al., An efficient microcode compiler for application specific DSP processors, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.,* **9**: 925–937, 1990.
20. C.-T. Hwang, Y.-C. Hsu, and Y.-L. Lin, Scheduling for functional pipelining and loop winding, *ACM/IEEE Des. Autom. Conf.,* San Francisco, 1991, pp. 764–769.
21. M. E. Wolf and M. S. Lam, A loop transformations theory and an algorithm to maximize parallelism, *IEEE Trans. Parallel Distrib. Syst.,* **2**: 452–471, 1991.
22. N. Park and A. C. Parker, Sehwa: A software package for synthesis of pipelines from behavioral specifications, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.,* **7**: 356–370, 1988.
23. P. G. Paulin and J. P. Knight, Force-directed scheduling for the behavioral synthesis of ASIC, *IEEE Trans. Comput.-Aided Des. Integr. Circuits Syst.,* **8**: 661–679, 1989.
24. C. E. Leiserson and J. B. Saxe, Optimizing synchronous systems, *J. VLSI Comput. Syst.,* **1** (1): 41–67, 1983.
25. C. E. Leiserson and J. B. Saxe, Retiming synchronous circuitry, *Algorithmica,* **6**: 5–35, 1991.
26. G. Goossens et al., An optimal and flexible delay management technique for VLSI, in C. I. Byrnes and A. Lindquist (eds.), *Computation and Combinational Methods in System Theory,* Amsterdam: North Holland, 1986, pp. 409–418.
27. R. Hartley and A. Casavant, Tree-height minimization in pipelined architectures, *IEEE Int. Conf. CAD,* 112–115, 1989.
28. M. Potkonjak and J. Rabaey, Retiming for scheduling, *VLSI Signal Process. Workshop,* San Diego, CA, 1990, pp. 23–32.
29. M. Potkonjak and J. Rabaey, Optimizing resource utilization using transformations, *IEEE Trans. Comput. Aided Des. Integr. Circuits Syst.,* **13**: 277–292, 1994.
30. D. A. Lobo and B. M. Pangrle, Redundant operation creation: A scheduling optimization technique, *28th ACM/IEEE Des. Autom. Conf.,* 1991, pp. 775–778.
31. M. Potkonjak and J. Rabaey, Maximally fast and arbitrarily fast implementation of linear computations, *IEEE Int. Conf. Comput.-Aided Des.,* paper 6c.4, Santa Clara, CA, 1992.
32. N. Jouppi and D. Wall, Available instruction-level parallelism for super-scalar and super-pipelined machines, *Proc. 3rd Int. Conf. Architectural Support Programming Languages Operating Systems,* Boston, 1989, pp. 272–282.

SUJIT DEY
University of California at San Diego

YOSEF G. T. GEFEN
Virginia Tech

ALICE C. PARKER
University of Southern California

MIODRAG POTKONJAK
University of California at Los Angeles