Considering Testability at Behavioral Level: Use of Transformations for Partial Scan Cost Minimization Under Timing and Area Constraints

Miodrag Potkonjak, *Member, IEEE,* Sujit Dey, *Member, IEEE,* and Rabindra K. Roy, *Member, IEEE*

Abstract—**We address the problem of transforming a behavioral specification so that synthesis of a testable implementation from the new specification requires significantly less area and partial scan cost than synthesis from the original specification. The proposed approach has three components: a library of relevant transformation mechanisms, an objective function, and an optimization algorithm. The most effective transformations for testability optimization are identified by analyzing the fundamental relationship between transformational mechanisms and topological and functional properties of the computations that affect testability. A dynamic, two-stage objective function that estimates the area and testability of the final implementation, and also captures enabling and disabling effects of the transformations, is developed. Optimization is done using a new randomized branch and bound steepest descent algorithm. Application of the transformation algorithm on several benchmark examples demonstrates significant simultaneous improvement in both area and testability of the final implementations.**

I. Introduction

A. Motivation

RECENTLY, the importance and advantages of addressing testability early in the design process has been established. The topic has attracted attention in both the testing and high-level synthesis research literature. A summary of the developments in this emerging field can be found in [3]. While several scheduling, assignment, and allocation algorithms for testability enhancement have been proposed, only limited attention has been paid to exploring the relationship between other high-level synthesis tasks, like transformations and partitioning, and testability. To the best of our knowledge, the use of only a very specific transformation (addition of deflection operations) for testability optimization has been reported [14].

At the same time, efforts to optimize other design metrics, such as area and throughput, clearly indicate that efficient application of transformations often has significantly higher impact on the quality of the final implementation than other high-level synthesis tasks, like scheduling and assignment. Techniques using transformations to optimize area [33], throughput [32], throughput and latency [36], power [5], and transient [19] and permanent [16] fault tolerance report improvements between factor of two and several orders of magnitude. On the other hand, only minor improvements in these goals have been reported when only scheduling and assignment are used [39].

Clearly, it is important to evaluate the potential of transformations for testability improvements during the high-level design process. In this paper, we present a technique which uses a variety of transformations to reduce the area overhead required by Design-For-Testability (DFT) techniques to make the final implementation highly testable. During testability optimization, area minimization and timing (throughput) constraints are simultaneously targeted.

B. Transformations as Testability Optimization Tools: Motivational Example

In the last few years, numerous high-level synthesis approaches which address testability at the behavioral level have been reported [3]. While several systems target Built-In-Self-Test (BIST) and hierarchical testability as the test strategy, a majority of high-level synthesis techniques explore the relationship between hardware sharing and sequential Automatic Test Pattern Generation (ATPG) methods. The presence of loops in a sequential circuit is a major source of problems for sequential ATPG. Partial scan is an effective technique to break loops in the circuit by scanning a subset of flip-flops (FF's) [6], [23]. Empirical evidence shows that breaking all nontrivial (with at least two FF's) sequential cycles is an effective heuristic for making a circuit highly testable [6], [23]. In this paper, we minimize the number of scan registers required to break all nontrivial loops in the datapath, and use this number as a measure of testability.

It has been demonstrated that a design can be synthesized from a high-level specification in such a way that a relatively small increase in the area of a datapath can often be sufficient to significantly reduce the cardinality of the Minimum Feedback Vertex Set (MFVS), and hence the number of scan-registers needed to break all loops [13]. This is achieved by alternating classical scheduling and assignment algorithms in such a way that the testability overhead is taken into account [13].

We illustrate the use of transformations for testability optimization using the control data flow graph (CDFG) of the fourth-order parallel IIR filter, shown in Fig. 1. Due to its short critical path, compact and regular structure, and excellent

Manuscript received September 1, 1994; revised January 25, 1995. This paper was recommended by Guest Editors W. Maly and Y. Zorian.

The authors are with C&C Research Laboratories, NEC USA, Princeton, NJ 08540 USA.

IEEE Log Number 9410364.

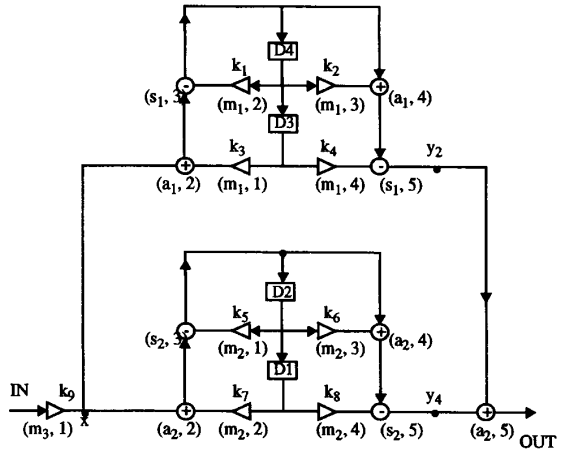

Fig. 1. Motivational example for demonstrating use of transformation for testability: fourth-order parallel IIR filter.

numerical stability properties (thereby requiring a short word-length), the parallel structure is one of the most frequently used IIR filters, mainly in audio applications [10], [28]. It is assumed that each operation takes one clock cycle. The available time is six control cycles. The critical path is also six cycles long. To meet the available time of six clock cycles, the minimal resource allocation requires 3 multipliers (three multiplications have to be scheduled in the first control step), 2 adders (two additions must be scheduled simultaneously in the second control step), and 2 subtractors (two subtractions must be scheduled in third control step simultaneously).

The result of scheduling and assignment, using an existing behavioral test synthesis system, BETS [12], is shown in Fig. 1. For instance, the first operation, a multiplication by k_9, is assigned to multiplier m_3, and scheduled in control cycle 1, shown by the tuple (m_3, 1). BETS performs allocation, scheduling, and assignment considering simultaneously testability overhead and area of the implementation, to generate an easily-testable design shown in Fig. 2 [12]. The resulting minimum feedback set contains four scan registers shown shaded in Fig. 2. As reported in Table I (row 4IIR.20), the resulting circuit is highly testable.

It can be shown that it is impossible to reduce the number of scan registers using any other scheduling and assignment. Consider the lower biquad in Fig. 1. In order to break the loops in the corresponding part of the datapath, at least two scan registers are required. If the register file of the transfer unit on which delay D1 is assigned is selected, then all the CDFG loops are broken. However, in this case another scan register is required to break assignment loops which are formed when the two '+' operations have to be assigned to the same adder A2. If the input variable to the transfer unit is not selected for scanning, then it is obvious that at least two scan registers are needed just for breaking the CDFG loops. Similarly, it can be shown that a minimum of two scan registers are needed for the upper biquad in Fig. 1. Note that the variables corresponding to delays can not share the same scan register because they are simultaneously alive in the first control step.

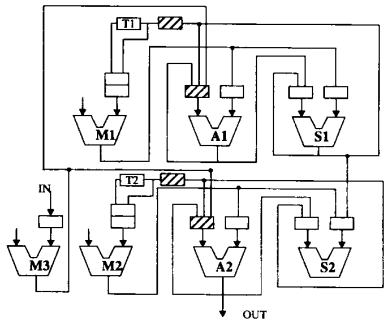

Fig. 2. The datapath of the initial fourth-order parallel IIR filter. When the shaded registers are scanned, all loops in the datapath are broken.

Consider now how transformations can be used to simultaneously reduce the area and testability cost of the design. A sequence of transformations shown in Fig. 3(a)–(c) is applied, so that eventually the CDFG shown in Fig. 3(c) is obtained. First, algebraic transformations are applied to the CDFG in Fig. 1 to obtain the functionally equivalent CDFG shown in Fig. 3(a). It can be shown that there is a correspondence between the coefficients c_i and k_i used in the two structures. The next transformation applied is the scaling operation [33], where two feedforward cuts are multiplied by β and $1/\beta$, to obtain the CDFG shown in Fig. 3(b). Finally, retiming is used to relocate the delay D3 to three new positions D5, D6, and D7. In addition, the CDFG is pipelined by introducing a delay D8, which is retimed back across the multiplication by c_9. The final CDFG, shown in Fig. 3(c), is significantly more suitable for BETS to produce a testable implementation.

The schedule and assignment obtained by BETS is shown in Fig. 3(c). Note that although not targeted, the throughput is also improved, since the critical path is reduced to five control steps, and the schedule uses five control steps. The resulting datapath is shown in Fig. 4. As shown in row 4IIR.20 in Table I, only one scan register is sufficient to break all loops in the datapath, and the datapath is highly testable. The hardware requirements are reduced to only two multipliers, one adder and one subtracter. Simulation using Hyper [35] indicates that all the computational structures of the fourth–order parallel IIR filter, shown in Fig. 1 and Fig. 3(a)–(c), have identical numerical properties and therefore identical word-length requirements.

C. Paper Organization

The rest of the paper is organized in the following way. In the remainder of this section, we outline the computational and hardware models used, the testing strategy targeted, and present the problem formulation. After a survey of the related work in Section II, we give an overview of the new approach in Section III. We explain the transformational mechanism which facilitates simultaneous optimization of area/time and testability characteristics of a design in Section IV. In the next three sections, we present two components of the optimization approach: an objective function and an optimization algorithm, and describe the HITS (HIgh-level synthesis system for TeStability and area optimization) system, which is the CAD

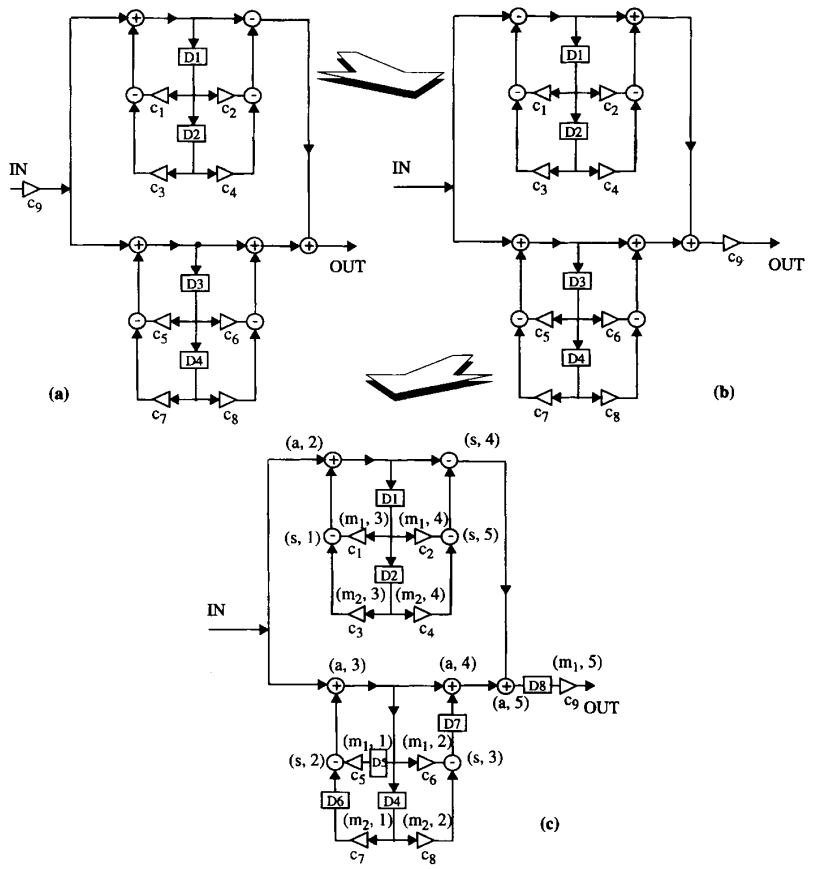

Fig. 3. The sequence of transformations for the simultaneous optimization of testability and area. (a) First, associativity and the inverse element law are applied. (b) Next, scaling transformation. (c) Finally, retiming. The datapath corresponding the final CDFG is shown in Fig. 4.

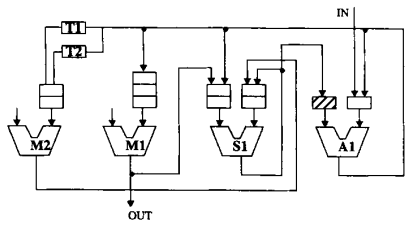

Fig. 4. The datapath of the transformed fourth-order parallel IIR. Only one scan register (shaded) is required to break all loops in the datapath, compared with four scan register in the initial design.

implementation of the proposed transformation approach. Finally, we demonstrate the effectiveness of HITS on a set of benchmark examples and draw conclusions.

D. Computational Model, Hardware Model, and Testability Assumptions

We assume a synchronous data flow computation model [22], which is often used in high-level synthesis. This model assumes a periodic computation done on a seminfinite stream of data along the time loop. A majority of DSP, video, control, and graphics applications follow the selected computational model. Synchronous data flow computation model is also well suited for the application and CAD treatment of transformations. This is mainly because the relatively strict timing discipline imposed by the model provides a convenient basis for the evaluation of changes in the structure of the computations, and enables accurate and rapid predictions of the properties of the final implementation.

Besides the assumed computation model, the selection of targeted hardware model also has numerous consequences. We adopt the dedicated register-file model, in which all registers are grouped in a number of register files. Each register file is connected to only one input of an execution unit, while each execution unit can send data to an arbitrary number of registers files. There are several reasons behind the decision to select the register file model of architecture. First, the model dictates grouping of single read, single write register files which enables area-efficient layout. This is the main reason that the register file model is widely used in general purpose architectures [31] as well as custom ASIC designs [11], [35]. Next, it has been demonstrated that the dedicated register file model reduces the number of interconnect at the expense of a somewhat higher number of registers. As the technology scales, the reduction of interconnect is becoming more important, making the dedicated register file model even more attractive. Finally, availability of accurate and

computationally inexpensive area prediction models [5] is an added attraction to use the register file model.

It is also important to explicitly state assumptions about the targeted testability methodology. We target testability assuming a single stuck-at fault model, and gate-level sequential ATPG. It is widely accepted that the elimination of all sequential loops, beside self-loops, is most often sufficient to achieve high testability. We target the elimination of all nontrivial sequential loops only in the data-path, assuming full scan of the control logic. For majority of numerically-intensive designs, the data-path completely dominates the area and FF requirements of the design [5], [35]. An effective DFT technique to make circuit testable is partial scan where a set of flip-flops (FF's) are scanned which breaks all nontrivial loops in the circuit [6], [23]. We assume employment of partial scan DFT methodology, and consider the number of partial scan FF's needed to make circuit highly testable as the measure of testability overhead.

In the rest of the paper, we will restrict our attention to the synchronous data flow model of computation, dedicated register file model, and sequential gate-level ATPG of the datapath. It is important to emphasize that all the transformations presented in this paper can be directly explored in other popular computational and hardware models, as long as the elimination of nontrivial sequential loops is a dominant measure of testability.

E. Problem Formulation and Key Novelties

For the first time, we explore the application of transformations to minimize the hardware overhead needed to make the implementation testable, under area and throughput constraints. The first goal is to identify the transformations which are most effective for simultaneous optimization of area and testability. Once the transformations are identified, the goal is to develop an optimization algorithm which transforms an arbitrary CDFG, so that an existing behavioral test synthesis system BETS [12] can produce a testable implementation with significantly reduced test hardware (partial scan) overhead. Since the computational intractability of the problem is obvious, the goal is to develop a practical heuristic which is sufficiently general to show high performance on a number of diverse examples. In particular, the goal is to experimentally demonstrate that transformations provide a significant advantage over scheduling and assignment techniques when testability is targeted. The key innovation is the employment of transformations for the new task, the optimization of testability (partial scan) overhead. We also introduce several novel modeling and optimization techniques. For example, we develop a concept of run-time-dependent prediction function, which is particularly well suited for properly addressing an enabling effect (see Section V) during the application of transformations.

II. Previous Related Work

Related work is outlined along two lines of research: high-level synthesis techniques for testability and transformations.

The mandatory tasks during high-level synthesis are allocation, scheduling, and assignment [27], [39], all of which have been shown to have significant impact on the testability of the synthesized designs. Existing high-level synthesis for testability techniques can be broadly classified according to the testing methodology targeted: BIST, gate-level sequential ATPG, or hierarchical test pattern generation. Four most notable efforts which target BIST have been reported from Waterloo University [15], Case Western University [9], [30], Stanford University [2], and University of California at San Diego [17].

Several research groups [12], [24], [26] have developed high-level synthesis systems which target sequential ATPG testability. These systems synthesize datapaths without loops, by using proper scheduling and assignment, and scan registers to break loops [12], [24]. A datapath synthesized from a behavioral specification may contain several types of loops, e.g.: *CDFG, assignment, and sequential false loops* [12]. A CDFG loop is formed in the datapath when there exists a cycle consisting of data-dependency edges in the CDFG. The other types of loops are introduced in the datapath during behavioral synthesis, specifically due to hardware sharing. For instance, when operations along a CDFG path from operation u to operation v are assigned to n separate modules, with u and v assigned the same module, an assignment loop of length n is created in the datapath. A comprehensive analysis of the formation of loops, in circuits synthesized by high-level synthesis techniques, is presented in [12], and a technique to estimate their formation is briefly reviewed in Section V.

Recently, Bhatia and Jha reported high-level synthesis techniques which target hierarchical test pattern generation [4]. Other works addressing testability during high-level design is related to use of testability analysis to guide test statement and test hardware insertion [7], [38].

Transformations are changes in the structure of a computation so that a particular objective is achieved, while the functional and timing dependencies between the inputs and the outputs are preserved. Transformations have been successfully used in a number of computation-related areas, including compilers, databases, VLSI algorithms, parallel algorithms, and computer architecture. Transformations have been successfully used in high-level synthesis for optimization of variety of goals [5], [16], [19], [21], [32], [33], [36].

Recently, a new transformation technique was developed which increases the complexity of the behavioral description while reducing the structural complexity of the resulting datapath [34]. Application of the new transformation technique to reduce the partial scan overhead for generating easily testable datapaths was demonstrated [14].

III. Simultaneous Area and Testability Overhead Improvement Using Transformations

The proposed behavioral synthesis approach for simultaneous optimization of area and testability was mainly influenced by the following four development aspects:

- *Reusability of existing tools:* The development and the support of transformation library is a tedious and time

consuming task. Therefore, we decided to reuse already available high-level synthesis transformations, and restrict our effort in this domain to combinations and modifications of existing transformations libraries [18], [35], [36]. As the back end of the synthesis system needed to verify the effectiveness of transformations, we employed the BETS behavioral test synthesis system [12] which provides allocation, scheduling, and assignment algorithms targeting the area and sequential ATPG testability of the final implementation. BETS also provides an interface to logic synthesis and physical layout tools.

- *Modular high-level synthesis tool:* The fast prototyping of the proposed system was greatly helped by integrating parts of already available modular tools. Our goal was to make the new system as modular as possible, so that its parts can be reused in other tools in the future.
- *Easy and flexible mechanism to enable trade-off between runtime and quality of results:* It has been our experience that potential users differ significantly in their utility preferences when high-level synthesis tools are used. While for some users a short runtime is the key issue, for others only the final quality of result matters. Our goal is to develop an approach in which runtime-quality of results can be easily changed at user request.
- *User-interaction:* It is becoming apparent that unlike CAD tools for lower levels of design, user interaction is crucial for achieving desirable results in a high-level design process. Our goal is to develop an optimization algorithm and objective function that the user can conveniently alter and possibly improve the synthesis process using additional synthesis knowledge and insight provided by any new approach.

The outlined goals resulted in the following organization (Fig. 5) of the HITS system. HITS has three main components: library of transformations, objective function calculation routines, and the optimization algorithm.

The search mechanism invokes the move generation algorithm. The move generation algorithm generates short sequences of transformations, applies them using the library of transformations, and evaluates them using the fast objective function. Finally, it selects one of the sequences and propose it to the search mechanism. The search mechanism considers the proposed aggregate transformations, and after optional use of accurate objective functions, decides about accepting the transformation, and updates if needed the best current solution. The search is continued or terminated by the search mechanism strategy. The detailed description of library of transformational mechanisms, objective function calculation routines, and optimization algorithm are presented in the next three sections.

IV. Transformational Mechanisms

In this section, we introduce transformations which were found to be particularly effective in addressing testability at the behavioral level. Transformations are classified into three groups: *atomic*, *global*, and *integrated*. Two criteria are used for classification: the number of axiomatic transformations which are employed and the scope of their application.

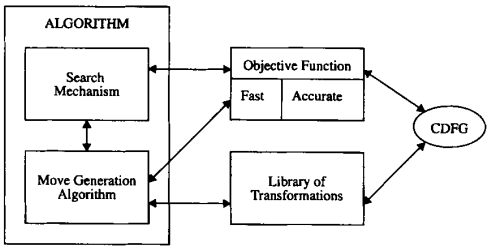

Fig. 5. The conceptual structure of the new approach for testability optimization using transformations. For the detailed explanation see Section III.

A. Atomic Transformations

Atomic transformations consists of a single axiomatic transformation rule at a single position in the CDFG. All atomic transformations can be categorized into three groups: *algebraic laws*, *redundancy manipulation techniques*, and *control flow transformations*. In this subsection, we describe atomic transformations which were found to be the most effective in addressing testability at the behavioral level and incorporated in our high-level synthesis system. For each transformation an example is presented which illustrates how the transformations can be used for testability enhancement. In all examples presented in this section, for the sake of simplicity, it is assumed that each operation takes one control cycle.

1) Algebraic transformations: These are defined over algebraic structures on which computation is conducted. We currently use the following four atomic algebraic transformations: commutativity, associativity, distributivity, and the inverse law. A more formal and comprehensive treatment of algebraic transformation can be found in [37].

Commutativity: An operation $*$ is commutative if for every x and y in a set over which the algebraic structure is defined, $x * y = y * x$. Although remarkably simple, commutativity often has significant transformation power. Under the dedicated register-file model used, two variables can share the same scan register if and only if the following two conditions are satisfied: a) the variables are input to operations of the same type; and b) they have disjoint lifetimes [12]. The aim of the transformation is to maximize sharing of the scan registers by fulfilling the above conditions as much as possible. Consider the example shown in Fig. 6(a). The available time is three control steps, and the critical path is three control steps long. The CDFG has two loops. It is easy to see that one multiplier and one adder are sufficient for the implementation with 100% resource utilization. At least two scan registers are required for breaking the loops because the left loop has the right input of multiplications and the left input of the addition as edges, while the right loop has right inputs to additions and the left input to multiplications as edges. The selected hardware model precludes the sharing of scan registers to break both the loops.

The application of commutativity on $+_3$, as indicated by C in Fig. 6(b), allows the same register in the left register file of the adder to be used for storing the variables (denoted by the crossed edges) which break both the loops. The required

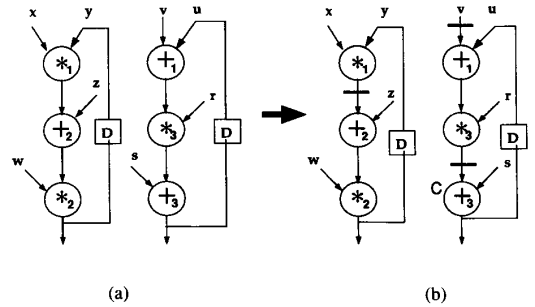

Fig. 6. Testability optimization using commutativity. The crossed edges denote variables which are assigned to the same scan register. (a) The initial structure needs at least two scan registers. (b) The final structure needs only one.

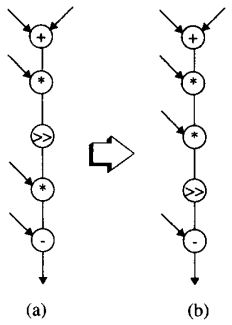

Fig. 7. Use of associativity for elimination of the need for scanning. The need for scan registers is eliminated after the application of associativity.

number of execution units is not altered, while the transformed design can be implemented with one less interconnect and one scan register.

Associativity: Associativity is one of the basic postulates in all algebraic structures for all defined operations starting from groups to vector fields and matrix algebras. An operation $*$ is associative, by definition, if for every three elements x, y, and z in a set over which an algebraic structure is defined, $(x * y) * z = x * (y * z)$.

Fig. 7(a) illustrates one of many ways in which associativity can be used to reduce the need for scan registers. We assume that the available time is five control steps. The minimum hardware implementation uses one each of multiplier, shifter, adder, and subtracter. The corresponding datapath is shown in Fig. 8(a). Due to hardware sharing of the two $*$ operators, an assignment loop containing the execution units $*$ and $\gg$ is created, shown by thicker lines, which can be broken by the scanning the shaded register.

The assignment loop can be eliminated, and therefore the need for a scan register avoided, if associativity is applied as shown in Fig. 7(b). The corresponding datapath is shown in Fig. 8(b). Note that the size of datapath is not altered, while the need for scan is eliminated.

Distributivity: Distributivity is the only axiom in the majority of algebraic structures which involves two different operations. Distributive law states that, for any three elements x, y, and z, and operations $*$ and $+$ in the algebraic structure, $x * (y + z) = x * y + x * z$.

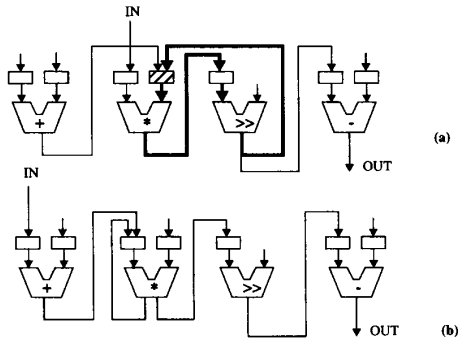

Fig. 8. The datapaths corresponding to the CDFG's shown in Fig. 7.

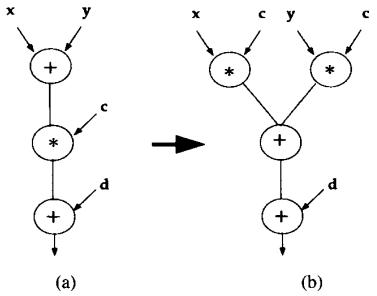

Fig. 9. The application of distributivity for testability optimization.

Fig. 9(a) and (b) illustrates the application of distributivity for the treatment of testability. The assumed available time in both examples is four control cycles. If we use the minimum amount of hardware (one adder and one multiplier) to implement the CDFG shown in Fig. 9(a), then it is easy to see that an assignment loop will be created in the corresponding datapath. However, after the application of associativity, one adder and one multiplier are still sufficient for the implementation of the functionally equivalent computation shown in Fig. 9(b), while the resulting datapath has no loops other than self-loops.

Note that if the available time was three cycles long then the transformed CDFG will result in a datapath which requires two multipliers. In this case, with the application of distributivity, the reduction in the testability overhead is achieved at the higher expense of one constant multiplication.

However, distributivity often simultaneously reduces partial scan overhead and datapath hardware. An example shown in Fig. 10(a) and (b) illustrates this type of application of distributivity. For the available time of three control steps, the transformed structure (Fig. 10(b)) can be scheduled using only one adder and one multiplier, while the initial structure needs two multipliers and one adder. The datapath of the transformed CDFG does not contain any cycles, whereas the initial datapath requires one scan register to break the loop involving one multiplier and the adder.

Inverse element law: In the scheme, shown in Fig. 11, the power of the inverse law is combined with two other algebraic transformations, commutativity and distributivity. The available time is 3 control steps. Note that the two CDFG loops do not have any operation of the same type.

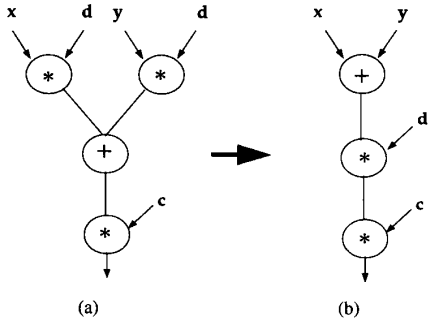

Fig. 10. The application of distributivity for testability optimization.

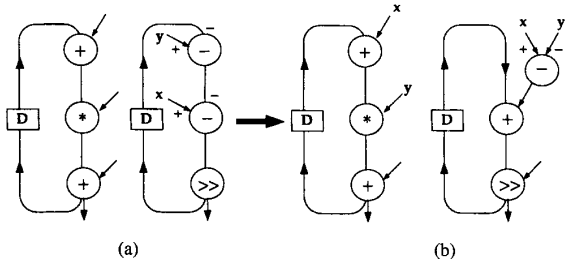

Fig. 11. The application of the inverse element law, augmented with associativity and commutativity, for testability optimization. For scanning the datapath which corresponds to the transformed CDFG, one scan register (the left input of adder) is sufficient.

Therefore, to break loops in the corresponding datapath, when the dedicated register file model is used, at least two scan registers are required. The computation can be transformed to the computation shown in Fig. 11(b) by using the combination of the inverse element law and other algebraic transformations.

Fig. 11(b) shows the transformed CDFG. It is easy to see that only one scan register is required for breaking all loops. Note that we still need only one functional unit of each type in the datapath.

2) Redundancy manipulation transformations:

Common subexpression replication: Several compilers and high-level synthesis systems use common subexpression elimination in order to reduce the number of operations and the area of the implementation. However, recently it has been demonstrated that the inverse transformation, common subexpression replication, is a potentially powerful optimization tool for throughput and area optimization [32].

The common subexpression replication concept and its application for testability improvement is shown in Fig. 12. The initial computational structure is shown in Fig. 12(a). Both the available time and critical path are five control steps long. Note that at least two scan registers are required to break the CDFG loop and the assignment loop formed if one multiplier is used for minimal area implementation.

In Fig. 12(b), the same computation is shown after the application of common subexpression replication to the + operation and consecutive application of commutativity. Now the same scan register can be used to break both the CDFG and the assignment loops. Note that due to hardware sharing the required number of resources remain unchanged.

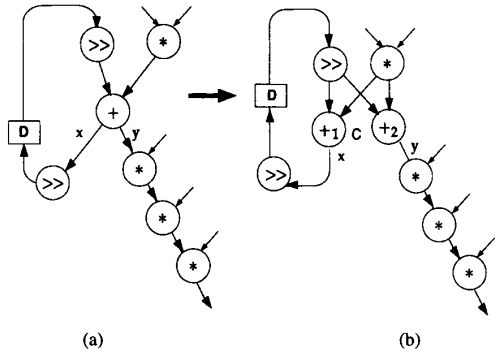

Fig. 12. The application of common subexpression replication for testability optimization.

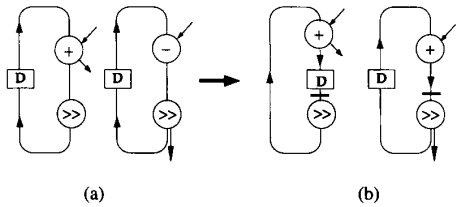

Fig. 13. Retiming for testability optimization. The crossed edges in the transformed CDFG are variables stored in the same scan register. While the implementation of initial CDFG requires two scan registers, that corresponding to the retimed CDFG requires only one.

3) Control flow transformations:

Retiming: The synchronous dataflow computation model introduces the notion of a state (delay). In the DSP literature, state is most often called delay, and denoted by D. A delay (state) denotes boundaries between two consecutive iterations of the body of the computation done on seminfinite stream of data.

The most convenient formal way to introduce the retiming transformation is to treat delay as an operation. Retiming, as a transformation, postulates the distributivity property of the delay operator over any other operation. Retiming states, that $D(a) * D(b) = D(a * b)$, where $*$ is an arbitrary operation. Delays are most often represented by rectangles in the CDFG graphical representation. Intuitively, retiming is a transformation where the number of delays on each output edge of an operation is reduced by one, while the number of delays on each input edge is increased by one, or vice versa.

Fig. 13 introduces the application of retiming for testability. The assumed available time is two control cycles. The only type of operation which belongs to both loops is shift. However, due to the timing constraints imposed by the data dependencies, both shifts must be scheduled in the second control step and their input variables cannot share the same scan register.

Consider now the functionally equivalent CDFG after the application of retiming, where in the left loop the delay is moved across the shift operation. Now the two shift operations can be scheduled in two different control steps, thereby allowing sharing of the scan registers, denote by the crossed edges in Fig. 13(b). Therefore, the test overhead is reduced from two scan registers to one.

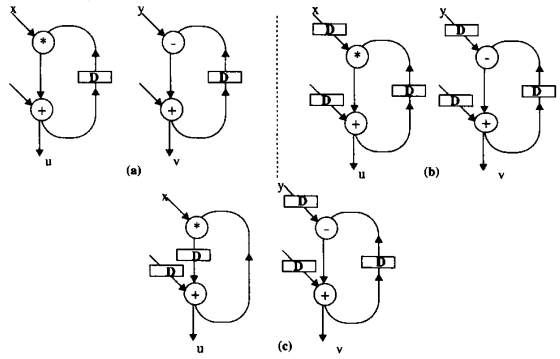

Fig. 14. The application of (b) pipelining and (c) subsequent retiming for testability optimization. After the application of the transformation one scan register is sufficient to break all loop in the resulting datapath.

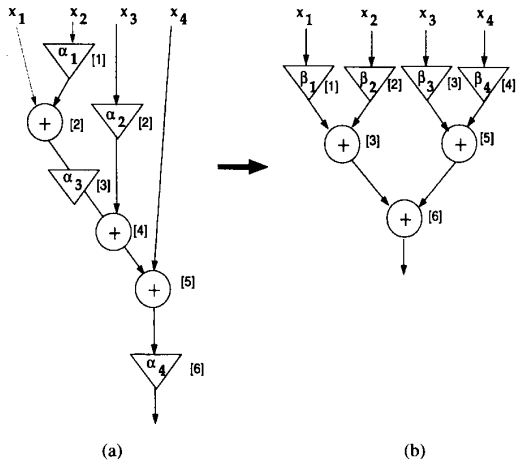

Fig. 15. The application of algebraic-speed up transformation [Iqb93] for testability optimization. The number in the brackets are the control step in which corresponding operations are scheduled.

Pipelining: Pipelining can be defined as a preprocessing step of retiming, where an arbitrary but equal number of delays is introduced on all primary input or all primary output edges of the CDFG. Retiming is most often automatically applied after pipelining. Fig. 14 illustrates the use of pipelining (with retiming) for reduction of testability overhead. Fig. 14(a) shows the initial CDFG. The available time and the critical path are two cycles long. The graph has two CDFG loops and only addition operations are common in both the loops. However, it is not possible to use the same register for breaking both loops, since the corresponding input variables to two additions have overlapping lifetimes. Hence at least two scan registers are required.

Note that retiming can not be applied, because of the blocking inputs x and y and the blocking outputs u and v. After the application of pipelining, the bottlenecks are eliminated, and now retiming can be used to move the boundary of iteration in one of the parts. After the application of retiming (see Fig. 13) on the left part of the CDFG, the same scan register can be used to break both loops in the resulting datapath. Note that as a side effect the hardware requirements are also reduced. While the initial design required two adders, the pipelined design requires only 1 adder.

B. Global Transformations

Global transformations consists of application of a single transformation mechanism at several places of the computation. Similar to the case of atomic transformations, three different transformational mechanism can be identified: algebraic laws, redundancy manipulation, and control flow alternations. The development and implementation of global transformations is a significantly more complex engineering task than the treatment of the corresponding atomic transformations. From each group of global transformations, we identified one global transformation.

Algebraic speed-up (CZAR): In the previous subsection, we demonstrated the use of several algebraic and redundancy manipulation transformations. In many cases, it is advantageous to use them in a particular order to achieve best results. Recently, it has been demonstrated that one conceptually simple, yet often very effective way is to use transformation in a particular order, so that the initially applied transformations enable the consequently applied. Algebraic-Speed-Up (CZAR) [18] is one such script which uses all algebraic axiomatic transformations (commutativity, associativity, distributivity, the inverse element law, and the zero element law) and several redundancy optimization techniques to optimize the length of critical paths to satisfy an arbitrary set of timing constrains imposed on arrival times of the inputs and the required times of the outputs. The side effect of the script is that it reorders operations in a particular fashion, so that all multiplications are done first, followed by addition and subtractions, and eventually divisions. This ordering of operations often reduces the testability overhead, because many assignment and false loops are eliminated.

Consider the CDFG from Fig. 15(a) and its functionally equivalent computation shown in Fig. 15(b). The critical path is six control steps long. Triangles represent multiplications with constants. The numbers in the brackets next to operations indicate the control step in which the operation is scheduled. Since no two operations of the same type are scheduled in the same control step, it is clear that one adder and one multiplier are necessary and sufficient execution unit resources. Therefore, from a resource utilization point of view there is no need for transformations. The corresponding datapath will have several assignment and false loops. At least one scan register is required to break the datapath loops.

The CDFG after the application of the CZAR program is shown in Fig. 15(b). The corresponding schedule of the transformed computation is shown in the same figure. Since all multiplications are done in the beginning and all addition at the end of the computation, no assignment loops, except adder self-loops, are formed. Therefore, the need for using scan registers in eliminated.

Time loop unfolding: Time loop unfolding is a popular control flow transformation. While initially only one iteration of the computation is considered, after the unfolding by factor

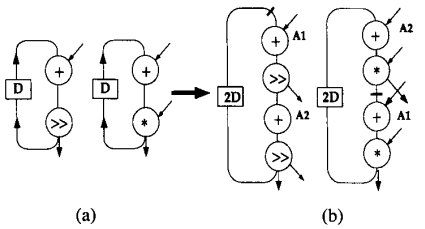

Fig. 16. Unfolding for scheduling. Although unfolding is mainly an enabling transformation, in several scenarios it directly results in the reduction of the required number of scan registers. The crossed edges in the unfolded CDFG can be stored in the same scan register.

k, $(k+1)$ iteration are simultaneously considered during the consequent high-level synthesis steps. Until now, loop unfolding is mainly used as an enabling transformation [5].

Time loop unfolding can directly reduce testability overhead. Fig. 16 supports this claim. Initially, the assumed available time is 2 control steps. At least two scan registers are needed. This is so, because the only common operation present in both CDFG loops is addition. Since both the additions must be scheduled in the first control step, they can not share the same execution unit, and their input variables cannot share the same register.

However, if two iterations are considered simultaneously, as provided by time loop unfolding and shown in Fig. 16(b), only one scan register is required to break all loops in the corresponding datapath. After the application of unfolding, four control steps are available for scheduling two iterations of the computations. Note that unfolding enabled two variables from two different iterations of the two loops to share the same scan register, since they do not have overlapping lifetimes.

Leiserson–Saxe retiming: We already defined retiming as a local transformation. It has been earlier reported that when retiming is used in this fashion, significant reduction in area is often achievable [33]. However, a more popular application of retiming in CAD techniques is when retiming is treated as a global transformation. Leiserson and Saxe [25] developed two optimal polynomial-time complexity algorithms for the minimization of the critical path and the number of states using retiming.

We observed that both the Leiserson-Saxe retiming algorithms can be used for reducing testability cost. The effect of the Leiserson–Saxe algorithm for the critical path minimization is mostly indirect. The resulting high slacks of operation in the CDFG can be used to reduce scheduling difficulty during the application of scheduling and assignment which targets testability. The Leiserson–Saxe algorithm for the minimization of the number of registers (LSR) is most effective when CDFG loops are targeted. Some high-level synthesis systems target only delay variables for breaking the CDFG loops [24]. If the LSR algorithm is used as a preprocessing step, a significant reduction in the required number of scan registers can be achieved.

Even when a more general strategy is used for breaking CDFG loops where no restriction on selecting only boundary variables is imposed, in many cases the Leiserson–Saxe algorithm produces minimal or near minimal solutions.

C. Integrated Transformations

Integrated transformations consist of several transformations being simultaneously applied on a part of the computation or the whole computation. While integrated transformations impose the most strict requirements on the computational structure in order to be applied, the improvement due to their application is the highest. The key advantage of integrated transformations is that they combine enabling effects of several transformations by employing knowledge about effective ways of transformation ordering.

Currently, we use only one integrated transformation, originally presented in [36]. The application domain of this transformation is linear time invariant computations. An arbitrary linear computation (after the application of this transformation) is transformed into one of several canonical forms (direct form, cascade form, or parallel form) which generates high-performance and low-cost implementations. While, the canonical forms do not necessarily result in a highly testable design, it can be shown that the consequent application of local transformations almost always result in CDFG's which have highly testable implementations.

Fig. 17 shows the example of this integrated transformation, which shows the initial Gray–Markel IIR filter, which can be transformed to the cascade form. Note that Gray–Markely filter has very involved cyclic structure of computation. After application of a few additional transformations only one scan register is sufficient for achieving 100% test efficiency (the last row in Table I).

D. Summary: How to Use Transformations to Improve Testability

We presented several examples to show how a significant number of transformations can be used to reduce the requirements for partial scan. However the application of transformations for testability improvement, in all presented examples, is based on a very few, common principles. In this subsection we summarize the key knowledge related to improving testability at the behavioral level, and thereby provide a platform for incorporating new transformations for this task.

It is possible to reduce the MFVS which has to be scanned so that all loops are broken by:

1) alternating lifetime of variables which can share the same scan register;
2) adding an operation in a CDFG loop of a particular type. The added operation is selected so that at least one of input variables can be shared;
3) removing or replacing an operation of a particular type from a CDFG loop or a path in the CDFG which influences creation of a loop during high-level synthesis in the corresponding datapath;
4) reordering of operations so that corresponding datapath loops are eliminated;
5) radical alternation of the structure of computations;
6) reducing word-length requirement.

Hardware-sharing of scan-registers is often the most effective way for reducing testability cost. However, the scheduling

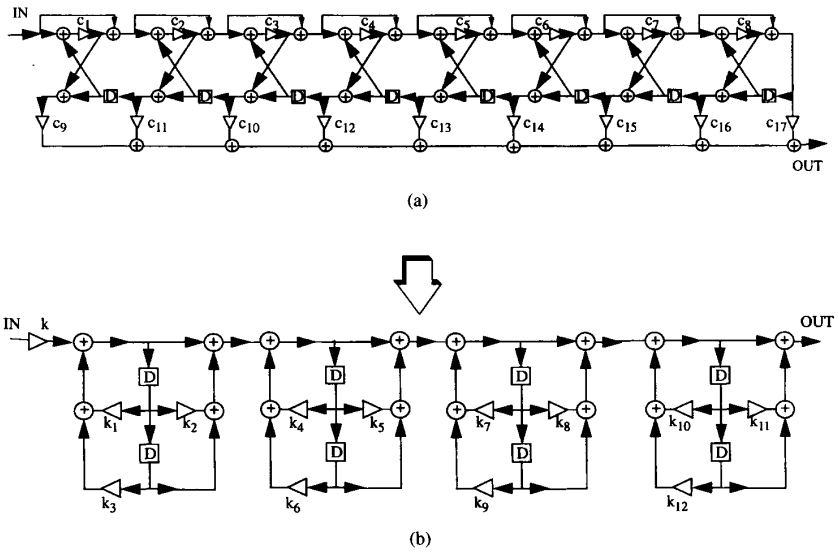

Fig. 17. Using the transformation procedure presented in [36], an arbitrary linear time invariant system can be transformed into a (b) cascade form which is a good starting point for generating high-performance, low-cost, and highly testable implementation.

and assignment degree of freedom for sharing scan registers is often limited by the data and control dependencies of the computation. Transformations are most often the tool of choice for removing scheduling and assignment constraints, and enabling better hardware sharing of scan registers.

The first group of transformations for testability target changing the lifetimes of the variables so that the efficient use of scan registers is improved. During this process, the primary goal is to make the lifetimes of the variables, which have to be stored in scan registers, maximally disjoint, while the reduction of their lifetimes is only of secondary importance. The importance and validity of the above criteria is explained in [12]. Typical examples from this group of transformations are retiming, pipelining, and common-subexpression elimination.

Note that the same transformation can have multiple mechanism for addressing testability. For example, we demonstrate how distributivity can be used to reorder operation. The following example shows how distributivity can be used to alter lifetimes of variables, so that testability is improved. Fig. 18 shows the initial example which has two CDFG loops. The assumed available time is three control steps. At least two scan registers are required, since the only type of operation present in both CDFG loops are additions (bound to be scheduled in the second control step) and multiplication (bound to the scheduled in the first control step. Therefore, their input variables cannot share the same scan register.

Fig. 18(b) shows the same example after the application of distributivity. Now additions are bound to be scheduled in two different control steps, and therefore can share the same scan register. Simultaneously the resource utilization of execution units is improved, since only one multiplier is required.

The addition of one or more operations of a particular type in the CDFG loop or on some path is motivated by the same goal as the alternation of the lifetimes of variables. The aim is to use the same scan register to store several variables.

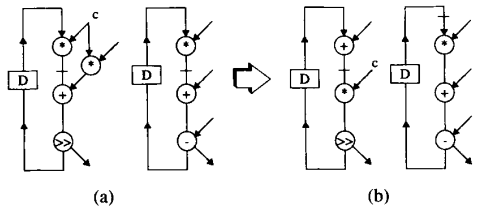

Fig. 18. Use of distributivity to alter lifetimes of variables which have to be stored in the same scan register. The crossed edge corresponds to scan variables. Note that after the (b) transformation, only one scan register is required to break all loops in the corresponding datapath.

Therefore, when the operation of a particular type is added, it is important to ensure that an input of the operation of this type has a high potential to be stored in a register which breaks many loops in the datapath, and the lifetime of the input data to the operation has a lifetime, disjoint with other variables which are targeted for assignment to the same register.

Note that most often it is easier to add operation of a particular type, than to remove all operations of a targeted type. The advantages of removing operations of a particular type is that in some situation it is possible to completely eliminate the need for partial scan.

In some situations when neither addition nor removal of operations is possible, the reordering of operation can be the best alternative. Reordering of operations is exceptionally effective for reducing or even eliminating a need for scan registers. Typical transformations from this category are associativity, distributivity, and algebraic-speed up. This type of transformations, unfortunately, often has a negative influence on the resource utilization, and therefore has be exercised with caution.

The fifth transformation mechanism often facilitates highest impact on testability. Currently, the only single transformation of this type in our framework is the integrated transformation

presented in Section IV-C. However, note that in many cases the successive application of local and global transformation can drastically alter the structure of the computation and its suitability for producing highly testable datapath.

The final mechanism, in some sense, has the most indirect effect. This mechanism is related to the reduction of the number of bits in design. In this case, even if the required number of registers is not reduced, the number of FF's is reduced due to the fact that each register contains fewer FF's. Although many atomic and global transformations influence the required number of bits, this effect is most pronounced when integrated transformations are applied. For example, an arbitrary direct form IIR filter can be transformed to the corresponding parallel form, using the integrated transformation described in Section IV-C.

As a final remark, it is important to stress that many of the transformation mechanisms, while primarily targeting testability, simultaneously reduce area.

V. Objective Function

Two contradictory objectives influence the development of the objective function which attempts to capture both testability prediction, and throughput and area constraints. The first goal is to develop an objective function which accurately predicts testability, throughput, and area metrics of the design which will be consequently scheduled. The second goal is to make the objective functions as computationally inexpensive as possible, because the objective function has to be evaluated numerous times during synthesis using transformation. It is well known that often the computational effort to compute objective function dominates runtime of the optimization algorithm. The situation is further complicated by the fact that behavioral synthesis is followed by several high-level synthesis tasks, such as resource allocation, scheduling, assignment, and hardware mapping as well as low-level synthesis stages which includes logic synthesis, floorplanning, and layout, the impact of which is difficult to predict. In order to satisfy these demanding and contradictory requirements, we selected an objective function which explicitly targets the three key aspects of evaluating the design at the behavioral level.

1) **Testability Cost (MFVS):** Estimated test hardware requirements, in our case the size of the minimum feedback vertex set of registers which break all loops in the datapath.
2) **Scheduling Difficulty (SD):** Estimated scheduling difficulties due to area (amount and structure of datapath components) and timing (throughput) goals and constraints.
3) **Transformation Difficulty (TD):** Suitability of the CDFG for the consequent application of effective transformations.

It is well known and widely documented that transformations can significantly alter numerical properties of a computation, and therefore alter bit-width requirements (see Section IV-E). Currently, the only technology available for deducing the required bit-width is concurrent high-level and bit-true simulation and result comparison. Since, this procedure is most often very time consuming, we developed two objective functions: fast and accurate. The fast objective function (**FOF**) assumes that since the previous evaluation of the bit-width requirements, no change in word-length requirements has occurred. Therefore, in this case the simulation is avoided. The accurate objective function (**AOF**) takes, as the starting point the RT-level resources indicated by the FOF, and conducts simulation in order to deduce the required number of bits in the word-length. The search for the minimal word-length is conducted using binary search starting from the word-length indicated by the FOF.

The datapath synthesized from a CDFG can contain several types of loops, due to the presence of loops in the CDFG (data-dependency loops), as well as due to hardware sharing during the assignment phase. A comprehensive analysis of the formation of the different types of loops in the datapath is provided in [12]. Since we want to apply the transformations before the high-level synthesis tasks of scheduling and assignment, and hence before actual formation of the assignment loops in the datapath, we need to use an estimation technique to predict the formation of loops, and thereby estimate the MFVS of the loops formed. A technique was introduced in [13] to estimate the formation of loops before the actual assignment process, which is briefly outlined here for completeness.

The formation of different types of loops in the datapath during the assignment phase can be captured by using a Data-Dependency and Compatibility Graph (DDCG). The DDCG models the data dependencies and the compatibilities of the operations of the CDFG. Each node in the graph represents an operation of the CDFG. There is a (undirected) compatibility edge between two operations if there is a nonzero probability that both the operations can be assigned to the same module. There is a (directed) data-dependency edge from operation v to operation w if operation w depends on data produced by operation v. We will denote a compatibility edge between v and w as $c(v, w)$, and a data-dependency edge from v to w as $d(v, w)$.

Fig. 19(a) shows segments of two paths in a CDFG, and Fig. 19(b) shows the corresponding data-dependency and compatibility graph. Assuming that each operation in the CDFG takes one control cycle, the longest path is 3 control steps long. For a schedule which requires 3 control steps, the As Soon As Possible (ASAP) and the As Late As Possible (ALAP) control steps in which each operation can be scheduled [39] is shown by the tuple [ASAP,ALAP] in Fig. 19(a). In Fig. 19(b), each compatibility edge $c(v, w)$, shown dotted, is weighted by an estimate of the compatibility, $\text{comp}(v, w)$, between two nodes v and w, as given by

$$\text{Comp}(v, w) = \begin{cases} 1, & \exists d(v, w) \\ 1 - \dfrac{\text{overlap}(v, w)}{(\text{mobility}(v) * \text{mobility}(w))}, & \text{otherwise.} \end{cases}$$

The compatibility between nodes $+_1$ and $+_3$, $\text{Comp}(+_1, +3) = 1 - (\frac{1}{2}) = 0.5$. All the compatibility edges $c(v, w)$ are weighted by $\text{Comp}(v, w)$, as shown in the DDCG in Fig. 19(b).

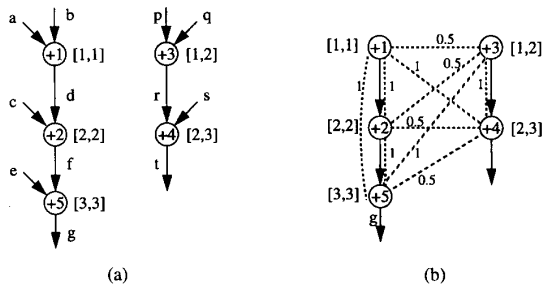

Fig. 19. Data dependence and compatibility graph concept. (a) CDFG. (b) Corresponding DDCG.

The paths of the DDCG can be represented using regular expressions. d^+ represents a path consisting of a sequence of one or more data-dependency edges d. The concatenation of two paths p and q is represented by $p.q$, or simply, pq. For example, the regular expression cd^+ represents a path starting with a compatibility edge c, followed by one or more occurrence of data-dependency edges. In Fig. 19(b), the path $(+_5, +_1), (+_1, +_2), (+_2, +_5)$ can be represented by cd^+.

A CDFG loop is formed in the datapath if there exists a cycle of the form d^+ in the DDCG. An assignment loop is formed in the datapath if there exists a cycle of the form cd^+ in the DDCG, and the compatibility edge c is used during module assignment. Similarly, if there is a cycle of the form $(cd^+)(cd^+)^+$ in the DDCG, and the compatible edges c in the cycle are used during module assignment, a sequential false loop is formed in the datapath.

Following the reusability guidance criteria, the software for the construction of the DDCG for a given CDFG is directly imported from the BETS system [12].

Testability cost (MFVS): Testability cost is the number of feedback vertices required to break all loops besides self-loops, in the DDCG. The DDCG has information about all possible loops that can be formed during assignment. We consider only those loops that have a high possibility of formation during assignment, as reflected by the high values of comp (v, w) on their c edges. We ignore those loops that have a low possibility of forming, reflected by low values of comp (v, w) on their c edges. To achieve this, we delete all those compatibility edges which have compatibility estimate less than a threshold. The cardinality of the MFVS is calculated using the greedy heuristic, similar to one used in [6], [23].

Scheduling difficulty (SD): Scheduling difficulty is proportional to the expected cost of the datapath area. First, the expected number of primary datapath components (execution units, registers, and interconnects) are estimated using the compatibility edges of the DDCG. For each type of hardware unit, the estimated number of instances is proportional to the number of corresponding instances in the DDCG and the inverse value of the cumulative sum of the weights of all the corresponding compatibility edges. This RT-level description of the datapath is used as the input in the statistical model for layout prediction developed in [5].

Transformation difficulty (TD): Transformation difficulty is proportional to the weighted change in the number of further transformations which can be applied on the graph after the application of a particular transformations. The weight is proportional to the importance of applying the consequent transformation on the particular node of the CDFG. Experimentally derived formula for quantifying the importance has the following as input parameters: the slack of the operation, size of the strongly connected component to which it belongs, the cardinality of the transitive fan-in and transitive fan-out set, and the cost and delay (number of control steps required by the operation to be executed) of the functional unit on which the operation is bound to be assigned.

The three components are weighted by three empirically derived factors. The weight factor of the TD component is time dependent, monotonically decreasing. The idea behind this dependency is that as the optimization process is close to finish, it is less and less important to plan for future improvement, and more important to optimize the current best solution.

The typical time required to calculate FOF of a moderately sized computation of about 50 nodes is between 20 and 100 ms on a SUN SPARCstation2. On the same platform and the same CDFG the evaluation of the AOF usually takes between 1 and 5 s. We conclude this section by a pseudo-code description of the fast objective function.

$$
\begin{aligned}
&\textit{Fast\_Objective\_Function FOF}\,(\textit{CDFG})\,\{\\
&\quad \textit{Form DDCG};\\
&\quad SD = \textit{Scheduling\_Difficultiy}();\\
&\quad MFVS = \textit{Testability\_Cost}();\\
&\quad TD = \textit{Transformation Difficult}();\\
&\quad FOF = \gamma_1 * SD + \gamma_2 * MFVS + \gamma_3(t) * TD;\\
&\}
\end{aligned}
$$

VI. Optimization Algorithm

The development of the HITS optimization algorithm for simultaneous optimization of area and testability overhead using transformations was influenced by the following four observations.

- *Solution space size:* The solution space of the problem is exceptionally large. Even for CDFG's of relatively moderate size, there are numerous possibilities at each step. For instance, even if we consider only commutativity, and if there are only 30 commutative operations in the CDFG, we have more than a billion combinations (2^{30}), each of them representing different functionally equivalent computations.
- *Solution space topology:* The solution space of the problem has very complex and nonregular topology. The situation is additionally complicated by the fact that three different components of the cost function and constraints (testability, area, and throughput) are often affected in different ways by the application of a particular transformation. While some transformations have global consequences, others have mainly local influence. For example, local retiming moves often have global influence which significantly changes the critical path, area,

testability, and may enable or disable the application of other transformation, while commutativity has only local influence.

- *Computationally expensive objective function:* Although a considerable effort was dedicated to a develop computationally inexpensive objective function, it is still relatively expensive to evaluate the exact objective function. The calculation of the exact objective function involves time consuming high-level (double precision floating point) and bit-true simulation. At the same time, the calculation of the approximate objective function, although computationally much less expensive, still requires significant computational effort.
- *Transformations ordering:* The specific order in which transformations are applied can greatly influence the optimization of one or more design metrics. While certain transformations do not directly contribute to the improvement of the quality of design, they may enable the application of other transformations which initially were not applicable. The application of the initially inhibited transformations may greatly improve the quality of the solution [32].

Additional requirements are imposed on the optimization algorithm by the specifications of the general approach described in Section III. It is apparent that no single heuristic or random search is effective enough to handle all listed requirements. Therefore, we opted to develop a composite approach where efficient randomized search is augmented by knowledge about the solution space. The current algorithm performs well on many examples, but many parameters of the algorithm can be altered, if needed, by future experience with new examples and new transformations. The fully modular nature of the approach facilitates easy modifications. In this framework, the set of requirements was addressed in the following way.

- *Solution space size:* The key for fast scanning of the solution space is the use of a fast deepest-decent search algorithm [1]. In order to enable avoidance of local optima, we superimposed randomization on the top of the primary strategy. The randomization level is user specified, and facilitates trade-off between search time and quality of the solution. In the current version of the program, the deterministic and randomized components initially have the same weight factor, and after each step the random component weight is reduced geometrically by 20%. Also we prevent the oscillation effect (when on the same place in the CDFG the pair of direct and its reverse transformation are applied) by assigning low probability to the application of the same type of transformation on the particular node (or set of nodes) many times. Finally, it is empirically observed that some parts of the CDFG are more important for area and testability than others. In particular, we assign higher probability to select nodes which belong to cycles with small slack, and nodes with large transitive fan-out and transitive fan-in.
- *Solution space topology:* The knowledge of the topology of the solution space is used in several ways. First of all, the integrated move is attempted only once in the beginning. Second, in the beginning of the search process, high impact transformations with global consequences are invoked with high probability. This group contains all global transformations, except unfolding, retiming, common subexpression elimination/replication. At the end of the search, transformations with more local effects (mainly algebraic transformations and common subexpression elimination/replication) are mostly invoked. Finally, pipelining is invoked as the last transformation in all cases where a significant number of nodes in the CDFG is outside cycles and expensive execution units are used. The motivation behind this strategy is explained in [32].
- *Computationally expensive objective function:* The AOF objective function is invoked only when the chances for a new best solution are high. Even the FOF objective function is invoked only after a set of moves are applied. If the solution is significantly worse, the probabilistic branch and bound rule is used to terminate the sequence of transformation moves which are more recently applied, and the search is restarted from the previous best solution.
- *Transformations ordering:* In the beginning, the selection of enabling transformation is more favored, while in the end mostly optimizing transformations are used. For the exact order of mutual enabling relationship among various transformations, see [33]. The Leiserson–Saxe algorithms for retiming are invoked at most once. Unfolding is invoked only if all other options have been tried and the user criteria for stopping search has not been achieved.

As we already described in Section III, the optimization algorithm has two independent components: move selection algorithm and search strategy. The first component, move selection algorithm, is captured by the following pseudo-code.

Move_Selection_Algorithm $(CDFG)$ {
Randomly select the number of global and local moves;
Select and apply global moves;
Evaluate FOF; *If the FOF increased by more than 10% terminate move*;
Select and apply global moves;
Evaluate FOF;
}

The search algorithm uses a dynamic randomized deepest decent search mechanism and is described using the following pseudocode.

Search_Strategy_Algorithm{
Initialization(); *Integrated Transformation*();
if (*new_FOF* < *Current AOF*) {
if (*new_AOF* < *Current AOF*) *update Best_Cost and Current_Best_Solution*; }
Initialize criteria1, *criteria2*;

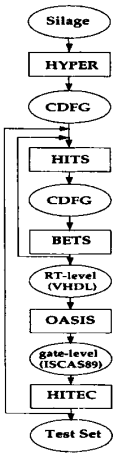

Fig. 20. The design flow of the new approach for addressing testability at the behavioral level. For detailed explanation, see Section VII.

while (criteria1 is satisfied) {
while (criteria2 is satisfied) {
Move_Selection_Algorithm (CDFG)
if (new_FOF < Current AOF) {
if (new_AOF < Current AOF) update Best_Cost and Current_Best_Solution; }
update criteria 2;
}}}

The Initialization routine sets as the best current solution the initial solution, and sets as the best cost the initial cost. Criteria for stopping the Move_Selection_Algorithm (CDFG) and the whole search are either user dictated time limit or requested size of the datapath and the number of scan registers.

VII. HITS and Experimental Results

The proposed transformation mechanisms, objective functions and optimization algorithms are incorporated as the core of HITS. The synthesis flow of the HITS system is shown in Fig. 20. The input to HITS is a functional specification of the application in an applicative stream-oriented Silage language [35]. The input is parsed using the Hyper parser, and the computation is stored in the Hyper CDFG format [35]. HITS operates on the Hyper CDFG format, and generates a transformed CDFG, after applying all the transformations mandated by the algorithms described in Section VI. At any moment of the transformation process, as required by the optimization process or the user, the Hyper simulator is invoked.

The transformed CDFG produced by HITS is given as input to the BETS behavioral test synthesis system [12], which performs allocation, scheduling, and assignment, targeting simultaneously area and testability while satisfying throughput constraints. The output of BETS is an annotated CDFG, where each node and edge attributes have corresponding scheduling and assignment information. The annotated CDFG is further processed by the Hyper hardware mapping tool. The output of the hardware mapper is an RT-level structural VHDL description of the design. The VHDL description is translated using the BETS translator to the Logic III format, which is the input to the OASIS system [20]. A gate-level netlist (ISCAS89 format) and physical layout are produced using the OASIS system. The gate-level netlist is used as the input to the HITEC sequential ATPG tool [29].

TABLE I
Experimental Result of the Application of the HITS System on 5 Benchmark Examples

Design Name	Design Version	Hardware	Sample Period	Scan/Tot FF	Faults	F.C (%)	T.E. (%)	ATPG Time (sec)
4IIR.20	BETS	2M,2A,2S	5	80/340	9716	96	100	224.9
	HITS	2M,1A,1S	5	20/280	8936	95	99	503.6
5EWF.20	BETS	3M,2A	17	60/460	10916	98	100	233.2
	HITS	2M,2A	17	20/340	7023	99	100	128.1
LIN3.16	BETS	5M,3A	13	144/400	14308	99	100	193.6
	HITS	3M,2A	13	16/224	12157	95	99	4221.5
ELLIP4.24	BETS	5M,4A	8	120/480	12986	99	100	162.2
	HITS	3M,2A	8	24/336	8478	98	100	146.9
8IIR.14	BETS	2M,3A,1S	34	182/462	9964	100	100	135.1
	HITS	1M,1A	34	60/132	6233	99	100	187.7

To evaluate the proposed transformation methodology, objective function and optimization algorithms, HITS was applied to the behavioral descriptions of the following designs: 4IIR-fourth-order parallel IIR filter, 5WDF—fifth-order elliptical wave digital filter, LIN3—fifth-order linear controller, ELLIP4—GE controller, and 8IIR—eigth-order Avenhaus Gray-Markel bandpass IIR filter. 5WDF is by far the most popular high-level synthesis benchmark [39]. We implemented the maximally fast 20-b version of the filter which conforms to CCITT G712 PCM standard. Two controllers, LIN3 and ELLIP4, are from the GE line of products. LIN3 is a 5-state controller, while ELLIP is a 4-state controller. Finally 8IIR is an acoustical filter which is a widely used benchmark in the DSP literature [10].

Table I shows the results of applying HITS and BETS to the designs. In the sequel, *BETS* refers to the implementation of the original CDFG using BETS when both testability and area are considered during allocation, scheduling, and assignment. *HITS* refers to the implementation obtained by transforming the original CDFG using HITS, and synthesizing the transformed CDFG using BETS. The numerical suffix after the name of design indicates the word-length of the implementation. In all examples, except the last one, the *BETS* and the *HITS* design required the same number of bits. For the 8IIR example the word-length reduces after the application of transformations from 14 to 12 b. The columns *Hardware* and *Sample Period* shows the number of execution units (multipliers, adders, and subtracters), and the number of clock cycles needed for both the *BETS* and *HITS* designs. Note that while the number of clock cycles taken by both the initial and final designs are maintained the same, the hardware needed by the final design is always less than the corresponding initial designs. The runtime taken by the HITS

system is under user control. To obtain the results reported in Table I, the runtimes on a SUN SPARCstation2 range from 5 minutes for the smallest example (4IIR) to 15 minutes for the largest example (8IIR).

The column *Scan/Tot FF* reports the number of scan FF's needed by BETS to synthesize a loop-free testable circuit, and the total number of flip-flops, for the *BETS* and *HITS* implementations. As can be seen, the number of scan FF's needed by BETS for the transformed CDFG's produced by HITS (*HITS*) is significantly smaller than the number of scan FF's needed by BETS for the original CDFG's (*BETS*). The reduction in the number of scan FF's ranges from a factor of 3 (5EWF) to a factor of 9 (LIN3). The average and median reductions are by factors of 4.81 and 4.00, respectively.

The last four columns of Table I demonstrate that the designs obtained using BETS and HITS are consistently highly testable using the gate-level sequential ATPG tool HITEC [29]. The total number of faults (column *Faults*), the fault coverage obtained (column *FC*), the test efficiency obtained (column *TE*), and the ATPG time taken in seconds on a SUN SPARCstation2 (column *ATPG Time*) are shown in Table I. Fault coverage, Test Efficiency, and ATPG times remain close for both initial and final designs. Only in the case of LIN3, the test generation time was significantly higher for the final design than for the initial design, by a factor of more than 20 times, which can be attributed to the sharp drop in the number of scan FF's used for the final implementation. Note that the *HITS* design needed significantly smaller number of scan FFS than the *BETS* design to achieve comparable high test efficiencies.

Finally, to illustrate that the number of scan registers produced by BETS is indeed required for high testability, and that HITS produces a significantly more testable design for a fixed number of scan registers, we conducted the following experiment on the 4IIR example. We applied the gate-level partial scan tool OPUS [8] on the *BETS* design of the fourth-order IIR parallel filter, restricting OPUS to scan at most 20 FF's, as used by the HITS generated design. After running for more than 5 h (18,651.3 s) on the OPUS-generated circuit, HITEC reported 24% fault coverage and 28% test efficiency, clearly indicating the advantage of applying transformations by HITS at the behavioral level for testability optimization.

VIII. Conclusion

This paper presented a comprehensive analysis of the effect of transformations of the behavioral specification on the testability of the resultant datapath. The analysis is the basis for a transformation-based high-level synthesis approach for simultaneous optimization of testability and area under throughput constraints. The proposed algorithms are integrated with existing high-level synthesis, low-level synthesis, and testability tools in a CAD system which is used to show the effectiveness of the approach on several benchmark examples. Experimental results demonstrate significant savings in partial scan overhead when the original specifications are transformed by HITS before using the behavioral test synthesis system BETS [12] to synthesize 100% testable designs.

References

[1] A. V. Aho, J. E. Hopcroft, and J. D. Ullman, *The Design and Analysis of Computer Algorithms.* Reading, MA: Addison–Wesley, 1974.

[2] L. J. Avra, "Allocation and assignment in high-level synthesis for self-testable data paths," in *Proc. Int. Test Conf.,* 1991, pp. 463–472.

[3] L. J. Avra and E. J. McCluskey, "High level synthesis of testable designs: An overview of university systems," in *ITC Test Synthesis Seminar,* 1994, pp. 1.1.1.– 1.1.6.

[4] S. Bhatia and N. K. Jha, "Genesis: A behavioral synthesis system for hierarchical testability," in *Proc. European Design & Test Conf.,* 1994, pp. 272–276.

[5] A. P. Chandrakasan *et al.,* "Hyper-LP: A design system for power minimization using architectural transformations," in *Int. Conf. Computer-Aided Design,* Nov. 1992, pp. 300-303.

[6] K. T. Cheng and V. D. Agrawal, "A partial scan method for sequential circuits with feedback," *IEEE Trans. Comput.,* vol. 39, no. 4, pp. 544–548, 1990.

[7] C. H. Chen and D. G. Saab, "BETA: Behavioral testability analysis," in *Proc. Int. Conf. Computer-Aided Design,* 1991, pp. 202–205.

[8] V. Chickermane and J. H. Patel, "An optimization based approach to the partial scan design problem," in *Proc. Int. Test Conf.,* Sept. 1990, pp. 377–386.

[9] S. S. K. Chiu and C. Papachristou, "A built-in self-testing approach for minimizing hardware overhead," in *Proc. Int. Conf. Comput. Design,* 1991, pp. 282-285.

[10] R. E. Crochiere and A. V. Oppenheim, "Analysis of linear networks," *Proc. IEEE,* vol. 63, no. 4, pp. 581–595, 1975.

[11] H. De Man *et al.,* "Synthesis of DSP systems at Leuven," *Proc. IEEE Int. Conf. Comput. Design,* 1987, pp. 133–145.

[12] S. Dey, M. Potkonjak, and R. K. Roy, "Exploiting hardware-sharing in high level synthesis for partial scan optimization," in *Int. Conf. Computer-Aided Design,* 1993, pp. 20–25.

[13] ——, "Synthesizing designs with low-cardinality minimum feedback vertex set for partial scan application," in *IEEE VLSI Test Symp.,* 1994, pp. 2–7.

[14] S. Dey and M. Potkonjak, "Transforming behavioral specifications to facilitate synthesis of testable designs," in *Proc. Int. Test Conf.,* Oct. 1994, pp. 184-193.

[15] C. H. Gebotys and M. I. Elmasry, "Integration of algorithmic VLSI synthesis with testability incorporation," *IEEE J. Solid-State Circuits,* vol. 24, no. 2, 1989.

[16] L. Guerra, M. Potkonjak, and J. Rabaey, "High level synthesis for reconfigurable datapath structures," in *Proc. Int. Conf. Computer-Aided Design,* Nov. 1993, pp. 26–29.

[17] I. G. Harris and A. Orailoglu, "SYNCBIST: Synthesis for concurrent built-in self-testability," in *Proc. IEEE Conf. Comput. Design,* 1994, pp. 101-104.

[18] Z. Iqbal, M. Potkonjak, S. Dey, and A. Parker, "Critical path minimization using retiming and algebraic speed-up," in *30th ACM/IEEE DAC Design Automation Conf.,* June 1993, pp. 573–577.

[19] R. Karri and A. Orailoglu, "Transformation-based high-level synthesis of fault-tolerant ASICS," in *Proc. 29th ACM/IEEE Design Automation Conf.,* 1992, pp. 662–665.

[20] K. Kozminski, Ed., *OASIS Users Guide.* Research Triangle Park, NC: MCNC, 1991.

[21] D. Ku and G. De Micheli, *Constrained Synthesis and Optimization of Digital Circuits from Behavioral Specifications.* Norwell, MA: Kluwer, 1992.

[22] E. A. Lee and D. G. Messerschmitt, "Static scheduling of synchronous dataflow programs for digital signal processing," *IEEE Trans. Comput.,* 1987, pp. 24-35.

[23] D. H. Lee and S. M. Reddy, "On determining scan flip-flops in partial-scan designs," in *Int. Conf. Computer-Aided Design,* pp. 322–325, 1990.

[24] T. C. Lee, N. K. Jha, and W. H. Wolf, "Behavioral synthesis of highly testable data paths under nonscan and partial scan environment," in *Proc. Design Automation Conf.,* 1993, pp. 292–297.

[25] C. E. Leiserson and J. B. Saxe, "Retiming synchronous circuitry," *Algorithmica,* vol. 6., no. 1, pp. 5–35, 1991.

[26] A. Majumdar, K. Saluja, and R. Jain, "Incorporating testability considerations in high-level synthesis," in *Proc. Int. Symp. Fault-Tolerant Computing,* 1992, pp. 272–279.

[27] M. C. McFarland, A. C. Parker, and R. Camposano, "The high level synthesis of digital systems," *Proc. IEEE,* vol. 78, no. 2, pp. 301–317, 1990.

[28] S. K. Mitra and J. F. Kaiser, *Handbook for Digital Signal Processing.* New York: Wiley, 1993.

[29] T. M. Niermann and J. H. Patel, "HITEC: A test generation package for sequential circuits," in *Euro. Design Automation Conf.* 1991, pp. 214–218.
[30] C. Papachristou, *et al.*, "SYNTEST: a method for high-level synthesis with self testability," in *Proc. Int. Conf. Computer-Aided Design,* 1991, pp. 458–462.
[31] D. A. Patterson and J. L. Hennessy, *Computer Architecture: A Quantitative Approach.* San Mateo, CA: Morgan Kaufman, 1989.
[32] M. Potkonjak and J. Rabaey, "Maximally fast and arbitrarily fast implementation of linear computations," in *IEEE Int. Conf. Computer-Aided Design,* 1992, pp. 304–308.
[33] ——, "Optimizing resource utilization using transformations" *IEEE Trans. Computer-Aided Design,* vol. 13, no. 3, pp. 277–292, Mar. 1994.
[34] M. Potkonjak and S. Dey, "Optimizing resource utilization and testabilit using hot potato techniques," *31th ACM/IEEE DAC Design Automation Conf.*, pp. 201-205, June 1994.
[35] J. Rabaey, C. Chu, P. Hoang, and M. Potkonjak, "Fast prototyping of datapath-intensive architectures," *IEEE Design and Test of Computers,* vol. 8, no. 2, pp. 40–51, June 1991.
[36] M. B. Srivastava and M. Potkonjak, "Transforming linear systems for joint latency and throughput optimization," in *Euro. Design Automation Conf.,* 1994, pp. 267–271.
[37] B. L. Van der Warden, *Modern Algebra.* New York: Frederick Ungar, 1950.
[38] P. Vishakantaiah, J. A. Abraham, and M. Abadir, "Automatic test knowledge extraction from VHDL (ATKET)," in *Proc. 29th ACM/IEEE Design Automation Conf.,* 1992, pp. 273–278.
[39] R. Walker and R. Camposano, *Survey of High-Level Synthesis Systems.* Boston, MA: Kluwer, 1991.

Miodrag Potkonjak (S'90–M'91) for a photograph and biography, see p. 30 of the January 1995 issue of this TRANSACTIONS.

Sujit Dey (S'90–M'91) received the B.Tech. degree in computer science and engineering from the Indian Institute of Technology, Kharagpur, in 1985, the M.S. degree in computer science from Southern Illinois University, Carbondale, in 1987, and the Ph.D. degree in computer science from Duke University, Durham, NC, in 1991.

Since 1991, he has been a Research Staff Member at the C&C Research Laboratories, NEC USA, Princeton, NJ. His research interests include several aspects of computer-aided design and analysis of high-speed, testable IC's and systems from behavioral, register-transfer, and logic level specifications.

Dr. Dey received a Best Paper award at the 31st Design Automation Conference in 1994. He has served on the technical program committee of the International Conference on Computer-Aided Design.

Rabindra K. Roy (S'84–M'92) holds the B.Tech. degree from Indian Institute of Technology, Kharagpur, and the M.S. and Ph.D. degrees from the University of Illinois at Urbana-Champaign, all in electrical engineering..

He is currently a Research Staff Member at C&C Research Labs, NEC USA, Inc., Princeton, NJ, where he is pursuing research in various aspects of testing, synthesis for testability, and low-power design. In the past, he has worked at AT&T Bell Labs and General Electric Research & Development Center investigating problems related to automatic test generation and design for testability.

Dr. Roy has been a co-recipient of two Best Paper Awards at International Conference on VLSI Design, 1994, for his work on low power design of DSP circuits and synthesis of initializable asynchronous circuits. He has more than twenty five publications in refereed journals and conference proceedings. He has served on the technical program committees of the IEEE VLSI Test Symposium and the International Test Synthesis Workshop.