High-Level Synthesis for Testability: A Survey and Perspective

Kenneth D. Wagner
Synopsys, Inc.
700 East Middlefield Road
Mountain View, CA 94043-4033

Sujit Dey
C&C Research Laboratories
NEC USA, Inc.
Princeton, NJ 08540

Abstract

We review behavioral and RTL test synthesis and synthesis for testability approaches that generate easily testable implementations. We also include an overview of high-level synthesis techniques to assist high-level ATPG.

1 Introduction

Synthesis for testability has been the subject of intense research since the late 1980s, concurrent with research into synthesis to satisfy area, timing and, more recently, power constraints. Originally, synthesis for testability identified gate level optimizations that could preserve or enhance circuit testability for a selected fault class without the need for more specific testability insertion techniques. In this domain, only the removal of combinational redundancies has been widely adopted. Occasional predictions of the disappearance of scan and related technologies in favor of synthesis for testability have proven false. Synthesis providers have generally been unwilling to restrict their optimization strategies to satisfy testability requirements (e.g., synthesis for robust path delay fault testability, synthesis to reduce random pattern resistance) when they impact area or timing results.

Instead, more specific testability insertion techniques ranging from ad hoc insertion of control or observe points to insertion of regular structures such as scan chains and built-in self-test (BIST) have emerged. These "invasive" techniques had been used in large computers since the late 1970s, but were generally supported only by custom CAD tools maintained by individual experts in test or manufacturing. In the early 1990s, commercial EDA vendors automated testability analysis and insertion of these explicit testability structures by providing *test synthesis* tools [5,36]. These vendors now offer a diversity of scan-related technologies for use by IC designers.

1.1 Overview of High Level DFT Approaches

The need for fast time-to-market and increased productivity are driving the trend towards high-level design (behavioral and register-transfer level (RTL). The fundamental behavioral synthesis tasks consist of allocation, scheduling, and assignment, Allocation decides the type and number of hardware resources that will be used to implement the behavioral description, scheduling refers to specifying the control step (clock cycle) in which each operation will be executed, and assignment refers to the binding of each variable/operation to one of the allocated registers/functional units (FUs). Fast and accurate estimation is required to traverse the search space for possible solutions to each of these tasks [17,40].

A Control-Data Flow Graph (CDFG) is typically used to represent a behavioral description. The data dependency edges in the CDFG reflect data dependencies of operands on the results of other operations, while control edges represent the flow of control.

In this paper, we give an overview of several behavioral and RTL design and synthesis approaches that have been proposed to generate easily testable implementations, targeting partial scan (sequential ATPG), fullscan and BIST methodologies. We also include an overview of high-level synthesis techniques to assist high-level ATPG.

2 Fundamental Approaches to Synthesis for Testability

In the last decade, research in synthesis for testability has focused on generation and processing of HDL specifications of system function. Most HDL descriptions use Verilog, VHDL or C, and sometimes serve a dual role as simulatable as well as synthesizable descriptions. Many commercial *test synthesis* tools operate post-compilation of an HDL into a gate level netlist, either on technology-independent (generic gates) or technology-dependent (mapped gates) descriptions. EDA vendors and researchers are exploring the coupling of HDL descriptions with testability structures, since it is attractive to apply synthesis compilation and optimization technology directly to a testable HDL description, optimizing functional and test logic concurrently, rather than introducing testability after the HDL has been processed.

Table 1 shows the operational level of testability insertion for selected commercial EDA tools. One important criterion to evaluate test insertion capabilities is their impact on design methodology [6]. Some tools require exporting and importing chip and module-level netlists for testability modification, while others promise fully integrated testability insertion at behavioral, RTL, technology independent or technology-dependent design phases. Another important evaluation criterion is the completeness of solution offered by a test insertion tool, including ease of composition of testability structures into a complete top-level test system, verification of correct protocol and operation at the IC or system level, and test data generation.

Table 1: Operational Level of Testability Insertion

Name	Synthesis Base	Testability Insertion Level
Sunrise	Viewlogic	technology-dependent
Mentor	Autologic II	technology-independent
LogicVision	Synopsys HDL & Design Compiler	HDL
IBM	Booledozer	tech-independent or tech-dependent

33rd Design Automation Conference ®

DAC 96 - 06/96 Las Vegas, NV, USA

Table 1: Operational Level of Testability Insertion

Name	Synthesis Base	Testability Insertion Level
Synopsys	Synopsys HDL & Design Compiler	HDL and technology dependent
Compass	ASIC Synthesizer	technology dependent
AT&T	Synovation	HDL and technology-dependent

3 Behavioral Synthesis for Sequential ATPG

Synthesis for testability at the behavioral level is complicated by the absence of a behavioral fault model that can be strongly correlated to silicon defects. Therefore, researchers have focused on innovative methods to include sequential ATPG or BIST objectives into the behavioral compilation.

3.1 Sequential ATPG Objectives

It has been empirically observed [10,22] that the complexity of generating sequential test patterns grows exponentially with the length of cycles in the S-graph, and linearly with the sequential depth of the FFs in the S-graph. Each node in the S-graph corresponds to a FF, and there is a directed edge from node *u* to node *v* if there is a strictly combinational path from FF *u* to FF *v* in the sequential circuit. Gate-level DFT techniques like partial scan have been developed based on this topological analysis. These attempt to break all loops, except self-loops, and minimize sequential depth. Behavioral synthesis for testability approaches use similar measures, loops and sequential depth, to synthesize testable implementations from behavioral descriptions, while preserving the performance and area constraints of the design.

3.2 Improving Register Controllability and Observability

Traditional register assignment techniques aim to minimize the number of registers needed to store all the variables. One way of improving the controllability and observability of data path registers is to assign the variables of the CDFG to maximize the number of (*I/O) register*s connected to primary I/O [25]. Also, the sequential depth from an *input register* to an *output register* can be minimized during register assignment, thereby improving the controllability and observability of all registers of the data path.

The approach adopted in [25] assigns each primary output to an output register, and then assigns as many intermediate variables as possible to the output registers. Next, it assigns each primary input to an input register, and as many of the remaining intermediate variables as possible to the input registers. Then the input and output registers are merged if possible to minimize the total number of registers. Finally, unassigned intermediate variables are assigned to extra registers. In most cases, the technique assigns a minimum number of registers, while improving testability of the data path. When two variables cannot share a register since their lifetimes overlap, the operations of the CDFG can be re-scheduled such that the lifetime of an intermediate variables does not overlap with the lifetime of an input/output variable, and the intermediate variable can be assigned to an I/O register. A mobility path scheduling technique has been proposed in [26] to minimize the sequential depth between registers and to maximize the number of I/O registers in the data path by sharing between I/O and intermediate variables.

3.3 Creation and Avoidance of Loops in the Data Path

Since loops contribute significantly to the difficulty of sequential ATPG, we discuss how loops are formed in a circuit generated by high level synthesis, and ways of avoiding their formation.

3.3.1 Loops in the behavioral description

Corresponding to each loop consisting of data-dependency edges present in the behavioral description (CDFG), a loop is formed in the data path. The CDFG loops can be broken by selecting a set of *scan variables* from the variables of the CDFG such that each CDFG loop has a scan variable, and assigning each scan variable to a scan register. The problem of selecting a set of scan variables to break the CDFG loops with a minimum number of scan registers is similar to selecting the minimum feedback vertex set (MFVS) to break the loops in a gate-level S-graph, with an important difference. While each selected vertex in an S-graph corresponds to one scan FF, the selected scan variables of a CDFG can share scan registers. Hence the MFVS is not necessarily a good solution to breaking CDFG loops with the minimum number of scan registers. In [33], two measures, the loop cutting effectiveness measure and the hardware sharing effectiveness measure, have been developed. These measures are used to select a set of scan variables such that the selected variables can be maximally shared (requiring a minimal number of scan registers) and the chances of sharing other variables to break loops formed during the subsequent high level synthesis steps are maximized.

A different approach has been adopted in [24]. At first, a set of boundary variables, which determine the boundary of loops, are selected to be assigned to the available scan registers, thereby breaking the loops corresponding to each boundary variable. Though the boundary variables cannot share the same register because they are alive simultaneously, other intermediate variables of the CDFG can share the registers with boundary variables. To facilitate maximal sharing, boundary variables with shorter lifetimes are preferred while selecting the scan variables. Next, the intermediate variables are assigned to both the available scan registers as well as the existing I/O registers, using the register assignment algorithms discussed in the previous section, to further minimize the number of loops.

3.3.2 Loops formed by hardware sharing

Even when the CDFG has no loops, or all the CDFG loops have been effectively broken by scan variables, hardware sharing of registers and functional units can further introduce loops in the data path [33].

When the operations along a CDFG path from operation *u* to operation *v* are assigned *n* separate modules, with *u* and *v* assigned to the same module, a loop of length *n*, termed an *assignment loop*, is created in the data path. Consider an example CDFG consisting of two paths shown in Figure 1. Let the given performance constraint be three control steps, and the resource constraint be two adders. A feasible schedule and assignment of the operations is: {+1:(1,A1), +2:(2,A2), +3:(2,A1), +4:(3,A2), +5:(3,A1)}, where each tuple refers to the control step and resource (adder). Figure 1(b) shows an assignment loop RA1→RA2→RA1 (shown in bold) in the resulting data path. To create a loop-free circuit, the register RA1 needs to be converted into a scan register. Other types of loops may also be formed in the data path during resource sharing [33].

Formation of loops in the data path may be avoided by proper scheduling and assignment. Consider the following schedule and assignment satisfying the performance and resource constraints:

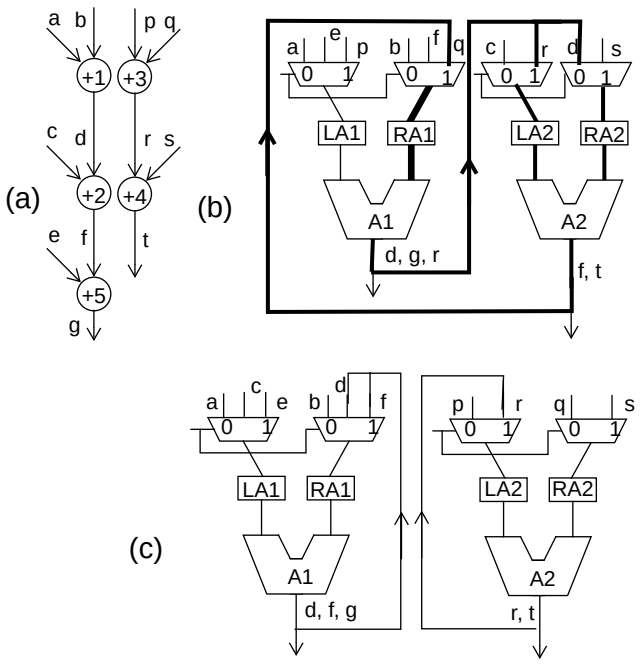

Figure 1. Loops formed during assignment: (a) Example CDFG, (b) Assignment Loop, c) No Loops except Self-Loop

{+1:(1,A1), +2:(2,A1), +3:(1,A2), +4:(2,A2), +5:(3,A1)}. Figure 1(c) shows the resulting data path. It contains only two self-loops. While one register needs to be scanned to break the loops of the data paths in Figure 1(b), no register needs to be scanned for the data path in Figure 1(c), assuming self-loops can be tolerated.

When formation of loops cannot be avoided due to the given performance and resource constraints, registers selected to break the CDFG loops can be reused to avoid creating loops during assignment. In [33], a simultaneous scheduling and assignment technique has been proposed which avoids formation of loops in the implementation, while satisfying the performance and area constraints. At each iteration of the algorithm, from the operations that have not yet been scheduled and assigned, an operation op_i with least slack is selected. The set of (module, control step) pairs, $\{(M_i, C_i)\}$, to which or in which the operation can be assigned or scheduled, are identified. For each pair, the cost in terms of testability, resource utilization and flexibility for scheduling and assignment of subsequent operations, is computed. Subsequently, a pair with the smallest cost is selected. A testability cost function is used to evaluate the costs associated with each type of loop formed and the scan registers necessary to break the loops. In [24], an assignment technique which minimizes the number of loops and maximizes the number of I/O registers in the resulting data path has been proposed.

In conventional gate-level partial scan, the designer synthesizes the module or chip without regard for testability, and then use gate-level partial-scan techniques to break loops enabling efficient sequential ATPG. Results from high level scan selection and loop-breaking indicate that loop-free highly testable designs can be synthesized that require significantly fewer scan FFs than conventional processes.

3.4 Modifying the Behavioral Description to Enhance Testability

A behavioral description can be modified to make the resulting implementation more testable than the implementation generated from the original description. In [9], the behavioral description is analyzed to detect hard-to-test areas, classifying variables as controllable, partially controllable, observable, and partially observable. Based on the testability analysis, test statements, which are executed only in the test mode, are added to improve the controllability and observability of all the variables in the description. The modified behaviors produce circuits with higher fault coverage and efficiency than the original description, at modest area overhead.

In hierarchical designs consisting of several modules, the top level design constrains the controllability and observability of its modules' I/O. A technique has been developed [37] to generate top level test modes and constraints required to realize a module's local test modes. The process of generating global test modes may reveal that some constraints cannot be satisfied, in which case, either the top level description, or the description of an individual module, must be modified to satisfy the constraints [39]. It has been shown that behavioral modification can yield an implementation with higher test efficiency than the original design with a modest increase in area.

A behavioral description can also be modified to make it more amenable to the synthesis for testability techniques discussed in the previous sections. One approach is to transform the CDFG by adding operations which do not change the original computation, but enable more sharing of scan registers so as to minimize the number of scan registers needed. In [16], deflection operations, with the identity element as one of the operands (like add with 0), are inserted between CDFG operations such that the original behavior is preserved. These operations are added to eliminate resource sharing bottlenecks, like overlapping lifetimes, such that more of the selected scan variables can share the same scan registers, thereby reducing the number of scan registers needed to break the CDFG loops.

Also, deflection operations are added so that formation of loops can be avoided during the assignment phase by maximally reusing existing scan registers. Since the deflection operations need to be executed in addition to the original operations, they are added only when the performance and area of the design is not adversely affected. Application of more complex transformations is discussed in [34]. The overall effect is that synthesizing a testable data path from the transformed specification requires fewer scan registers than needed for the original specification.

3.5 The Effect of A Controller on Testability

Most of the behavioral synthesis for test techniques concentrate on improving the testability of the data path, assuming that the controller can be made testable independently, and that its outgoing control signals to the data path are fully controllable in test mode. However, even when both the controller and the data path are individually testable, the composite circuit may not be easily testable by gate-level sequential ATPG. The main problem is control signal implications which may create conflicts during sequential ATPG [14]. The controller may be redesigned such that the identified implications are eliminated. The technique involves adding a few extra control vectors to the existing control vectors which are outputs of the controller. Application of the controller DFT technique has shown the ability to produce highly testable controller-data path circuits, with only marginal area overhead, even when both high-level and gate-level loop-breaking DFT techniques fail.

Another high-level synthesis for testability technique which considers the effect of the control logic on the testability of the design is [18]. Testability is measured not only based on sequential depth and testability characteristics of data path modules, but also the testabil-

ity of registers is determined by analyzing the control logic used to control the loading of the registers.

4 RTL Synthesis for Testability

4.1 RTL Modification & Analysis for Testability

There are several alternatives to enhance RTL descriptions for testability. The description can be augmented to improve testability by rewiring internal signals to more controllable or observable nodes when a test signal is active. With information regarding the connectivity of the modules and the functionality of each module, transformations that restructure the data path and minimize control logic by using don't care conditions extracted from the data path can yield optimized 100% single stuck-at fault testable fullscan designs [8].

An RTL description can also be used to identify the hard-to-test areas of a design, by analyzing testability ranges and the minimum and maximum number of clock cycles needed to control and observe an RTL node [12]. With RTL testability analysis, a partial scan selection method has been proposed which results in significantly better performance when compared to techniques limited to gate-level information only. In [37], an efficient partial scan method is developed to break data path loops. Both register nodes as well as non-register nodes are considered for breaking, with register nodes replaced by scan registers, and transparent scan registers placed on non-register nodes, thereby significantly reducing the number of scan registers needed.

4.2 Introduction of RTL Testability Structures

Testability structures, such as an IEEE 1149.1 boundary scan cell, can be directly synthesized. RTL can be used to describe their functionality. Several problems must be solved with such an approach to avoid sub-optimal results or methodologies: meeting functional and test mode performance constraints; automating safe connection of the structure; recognizing and using custom library cell components when available; and not violating technology rules (such as fanout limitations). The use of specific testability-oriented compilation and optimization directives embedded in the RTL description can also guide synthesis in reducing problems such as those above.

Some testability structures are ill-suited to code directly into an RTL, since they overconstrain synthesis. For instance, the functionality of a scan path can be coded into a Verilog description, but the synthesis system will not recognize the special nature of the structure and hence will not exploit the opportunity to select among Q, Q' or SO outputs to meet design and technology constraints.

Knowledge of structural and functional knowledge embedded in an RTL description has been used for non-scan DFT schemes like test point insertion [15]. Instead of conventional techniques of breaking loops by making FFs scannable, functional units are "broken" by inserting test points, implemented using register files and constants. It is shown that it suffices to make all the loops k-level (k>0) controllable and observable to achieve very high test efficiency. This new testability measure eliminates the need of traditional DFT techniques to make one or more registers in each loop directly (k=0) accessible to scan or primary I/O, significantly reducing the number of test points needed while maintaining high fault coverage.

5 Behavioral Synthesis for BIST

To make a design self-testable using the pseudorandom BIST methodology, it needs to be reconfigured during test mode into a set of acyclic logic blocks (LBs). Each LB has the equivalent of a pseudorandom test pattern generation register (TPGR) at each of its inputs, and a signature register (SR) at each of its outputs. In situ BIST requires reconfiguration of a functional register as a TPGR or SR. Such a register can be implemented as a built-in logic block observer (BILBO) [21]. In each test cycle, the TPGRs at the inputs of a block generate pseudorandom test patterns, and the test response of the block is captured by clocking data ports and analyzed by the SRs at its outputs. Many commercial BIST schemes rely on insertion of partial scan or fullscan into the LB that can be reconfigured to allow initialization and loading/unloading of stimulus and response data during BIST.

5.1 Minimizing Test Registers

A register cannot be configured both as a TPGR and an SR simultaneously, unless it is implemented as a concurrent BILBO (CBILBO), which is very expensive in terms of area and delay penalties. Hence, a self-adjacent register, which serves as both an input and an output of a LB, poses a problem, since it may have to be implemented as a CBILBO. An objective of generating self-testable data paths with low area overhead is to minimize the formation of self-adjacent registers [3,4,19,31].

In [3], it is assumed that every self-adjacent register will have to be implemented as a CBILBO. Given the scheduling and assignment of operations to modules, register assignment is performed to minimize the number of self-adjacent registers, and hence the number of CBILBOs. A conventional method of assigning a set of variables to the minimum number of registers is to color a conflict graph with the minimum number of colors. The nodes of the conflict graph correspond to the variables of the CDFG, and there is an edge between two nodes if the corresponding variables cannot be shared because of their overlapping lifetimes. To minimize the formation of self-adjacent registers, conflict edges are also added between two nodes if the corresponding variables are an input and output of the same module, either due to the variables being the input and output of the same operation, or due to the variable being an input and output of two different operations which are assigned to the same module. Experimental techniques generate data paths with fewer self-adjacent registers and an equal number of total registers, when compared with data paths produced by conventional register assignment techniques.

Formation of self-adjacent registers can be completely avoided by restricting the data path architecture used. In [31], the basic building blocks used to map a variable and the operation which generates the variable is a test function block (TFB), which consists of an ALU, a multiplexer at each of the inputs of the ALU, and a test register (TPGR, SR, or BILBO) at the output of the ALU.

Instead of considering mapping of variables and operations of the CDFG to individual registers and ALUs as done conventionally, each (v, o(v)) pair, termed action, where v is a variable, and o(v) is the operation producing v, is considered for mapping to TFBs. Two actions, (v1, o(v1), (v2, o(v2)) are compatible and can be merged (assigned to the same TFB) if (i) the lifetimes of v1 and v2 do not overlap, and (ii) v1, v2 are not the inputs of o(v1), o(v2) respectively. The second condition is needed to ensure that the output register of a TFB does not become an input of the TFB, thus ensuring that no self-adjacent register is formed. The assignment technique first identifies sequences of compatible actions, each of which can be merged and mapped to a single TFB. A prime sequence does not contain any other sequence. Assignment to a minimal number of TFBs is then achieved by finding a minimal set of prime sequences which cover all the actions of the CDFG.

The restriction of one output register per TFB prevents the sharing of operations whose output variables have overlapping lifetimes. Self-testable datapaths with even fewer TFBs can be formed by using an extended TFB (XTFB), which contains an ALU with mul-

tiple input as well as output registers[19]. During test mode, while the two input registers are configured as TPGRs, only one of the multiple output registers need to be configured as a SR, thus allowing the presence of self-adjacent registers which have to be configured as TPGRs but not SRs. By avoiding the use of CBILBOs while still allowing some self-adjacent registers, use of XTFBs enable generation of self-testable data paths with less test area overhead than either the traditional high level synthesis techniques or the BIST register assignment approach [3]. The test area overhead can be further reduced by relaxing the requirement that the output register of every ALU has to be a SR, instead allowing the test response to propagate through other ALUs before being captured in a SR, forming logic blocks with sequential depth between TPGRs and SRs greater than 1. The above scheme results in fewer SRs but reduces fault coverage, allowing trade-off between test area overhead and fault coverage.

BIST overhead can be reduced by not only minimizing the number of CBILBO registers that need to be used, but also the number of TPGRs and SRs needed to test all the data path modules [32]. After the scheduling and module assignment phases have been completed, register assignment can be done to maximize the number of modules for which a register is an input register and hence can act as a TPGR, and the number of modules for which a register is an output register and hence can act as a SR; in the resulting data path, the TPGRs and SRs can be maximally shared among the data path modules, resulting in a minimal number of registers that need to be converted to TPGRs or SRs. Every self-adjacent register in the data path does not need to be converted into a CBILBO in order to provide a test environment for all the modules. Exact conditions under which a self-adjacent register needs to be a CBILBO are given in [32]. The register assignment phase can check the conditions and try to avoid assignments leading to CBILBOs, whenever possible.

5.2 Minimizing Test Sessions

In the most general BIST scheme, a test path through which test data can go from the TPGRs to the SR at the output of a logic block may pass through several ALUs. This leads to two or more test paths sharing the same hardware (registers, ALUs, multiplexers, buses), thus creating conflicts and forcing need for multiple test sessions. Scheduling and assignment techniques have been presented in [20], which uses test conflict estimates to generate data paths which require minimal number of test sessions and hence have maximal test concurrency. Experimental results show the ability to data paths that require only one test session. Note that assignment techniques like [32], which encourage sharing of TPGRs/SRs between logic blocks may also lead to test path conflicts and hence reduced test concurrency; the techniques in [20] do not address such conflicts.

5.3 Adding Test Behavior

A general BIST scheme is proposed in [31], where only the input and output registers are configured as TPGRs and SRs respectively. Testability metrics are developed to measure the controllability/ observability of signals in the original design behavior, under the application of pseudorandom vectors at the primary inputs. A test behavior, executed only in the test mode, is obtained by inserting test points in the original behavior to enhance the testability of required internal signals. The test points need extra primary I/O, implemented by extra TPGRs/SRs. The combined design and test behavior are synthesized together using any high level synthesis tool. A testing scheme is proposed which uses the test behavior to generate tests for the complete design, controller and data path, using only three test sessions.

5.4 Using Arithmetic Units as Test Generators and Compactors

Instead of using special BIST hardware like TPGRs and SRs, functional units can be used to perform test pattern generation and test response compaction [28]. A high level synthesis methodology has been proposed to synthesize data paths where high fault coverage can be obtained using arithmetic test generators and test compactors. A testability metric termed subspace state coverage is used to guide the synthesis process, both in characterizing the quality of test vectors required to provide complete fault coverage of each functional unit, as well as the quality of test vectors seen at the inputs of each operation in the CDFG after the degradation suffered by the patterns due to propagation through various operations. For each arithmetic unit in the module library, the input subspace state coverage needed to obtain complete structural coverage is characterized. Next an additional generator is applied at the inputs of the CDFG and the state coverage measured at the inputs of the operations. If two operations, with S1 and S2 denoting the states covered at their inputs, are mapped to the same arithmetic unit, the states covered at the input of the unit is the union of S1 and S2. During high level synthesis, assignment of operations to functional units is done to maximize the state coverage obtained at the inputs of each functional unit.

6 High Level Synthesis & Test Generation

Several techniques have been proposed to generate test vectors using the high level description of a design [2,7,19,23,29,39]. We briefly review here two approaches which use high level synthesis to help in test generation. In [38], global constraints that the design imposes on each module are passed to an ATPG tool to generate gate-level tests for each individual module. Subsequently, the module test sets are combined with the global test modes extracted [38] to generate test vectors that can be applied at the primary inputs of the hierarchical design. The high level description can be modified to satisfy the global constraints whenever they cannot be satisfied at the module level.

The ability to re-establish the context of test patterns generated for a module at the top-level of a design allows reuse of test data. Precomputed test sets of the modules can be used to generate tests for the complete design, provided the test environment for each module, giving the set of symbolic justification and propagation paths to and from the module, is known. Automating and developing new DFT and ATPG techniques to facilitate test data reuse are becoming very important as design reuse of cores and other components gains popularity to improve designer productivity and companies seek to leverage their intellectual property investments.

The test environment of an operation assigned to a module can be used as the test environment for the module. In [7], the control and data flow specified in the behavioral description of a design is used to identify the test environment for an operation. The assignment phase in high level synthesis is used to help ensure that each module has at least one operation which has a test environment; if that is not possible, test points are introduced to provide the test environment[ref]. The hierarchical tests, providing high fault coverage, can be generated using the module tests and test environments more quickly than test generation done at the gate-level. Inter-module testing often remains a problem in such a macro test environment.

7 Further Perspectives

This paper has presented an overview of test synthesis and synthesis for testability. It is intended as both a survey and perspective on current practice. While the research results show great promise,

incorporation of the techniques in commercial synthesis and test CAD products may be facilitated if the following issues are addressed: a) currently, the proposed techniques are mostly applicable to data-flow intensive and arithmetic intensive designs like DSP filters and microprocessors. To broaden the scope of their applicability, techniques need to be evolved for control-flow oriented designs, like telecommunication applications; b) all the existing high-level approaches consider only the stuck-at-fault model; other testing methodologies like delay fault testing and IDDQ testing have not yet been addressed.

Incorporation of high-level synthesis for testability techniques in commercial CAD tools will be driven by the need for faster time to market and increased productivity. These market requirements already motivate design capture at higher levels of abstraction. Commercial behavioral synthesis offerings from CAD vendors have increased the acceptability of high level synthesis in the design methodology. As the most important design decisions and activities move to higher levels, it is sensible to migrate the appropriate test-related decisions. Otherwise, re-visiting the design tradeoffs at lower levels to consider testability risks time-to-market and productivity gains.

8 References

[1]M.S. Abadir and M.A. Breuer, "A Knowledge Based System for Designing Testable VLSI Chips," *IEEE Design & Test of Computers*, 2(4):56-68, August 1985.

[2] J.R. Armstrong, "Hierarchical Test Generation: Where We Are, And Where We Should Be Going," *Proc. EURO-DAC*, pp. 434-439, 1993.

[3] L. Avra, "Allocation and Assignment in High-Level Synthesis for Self-Testable Data Paths," *Proc. International Test Conference*, pp. 463-472, 1991.

[4] L.Avra and E.J. McCluskey, "Synthesizing for Scan Dependence in Built-In Self-Testable Designs," *Proc. International Test Conf.*, pp. 734 - 743, 1993.

[5] R.G. Bennetts, Guest editor, "Metamorphosis in Design: Test Synthesis," *IEEE Design & Test of Computers*, Vol. 12, No. 2, Summer 1995.

[6] R.G. Bennetts and K.D. Wagner, "Test Synthesis: Towards Higher Levels of Abstraction, *Proc. Electronic Design Automation & Test Conference, Asia*, 1995.

[7] S. Bhatia and N. K. Jha, "Genesis: A Behavioral Synthesis System for Hierarchical Testability," *Proc. European Design and Test Conference*, 1994.

[8] S. Bhattacharya, F. Brglez, and S. Dey, "Transformations and Resynthesis for Testability of RTL Control-Data Path Specifications," *IEEE Transactions on VLSI Systems*, 1(3):304-318, Sept. 1993.

[9] C.-H. Chen, T. Karnik, and D.G. Saab, "Structural and Behavioral Synthesis for Testability Techniques," *IEEE Transactions on Computer-Aided Design*, 13(6):777-785, June 1994.

[10] K.T. Cheng and V.D. Agrawal, "A Partial Scan Method for Sequential Circuits with Feedback," *IEEE Transactions on Computers*, 39(4):544 - 548, April 1990.

[11] V. Chickermane, J.Lee, and J.H. Patel, "Addressing Design for Testability at the Architectural Level," *IEEE Transactions on Computer-Aided Design*, 13(7):920-934, July 1994.

[12] G. De Micheli, "Synthesis and Optimization of Digital Circuits," *New York, McGraw-Hill, Inc.*, 1994.

[13] S. Devadas, A. Ghosh and K. Keutzer, "Logic Synthesis," *New York: McGraw-Hill, Inc.*, 1994.

[14] S. Dey, V. Gangaram, and M. Potkonjak, "A Controller-Based Design-for-Testability Technique for Controller-Data Path Circuits," *Proc. Int'l Conference on Computer-Aided Design*, pp. 534 - 540, 1995.

[15] S. Dey and M. Potkonjak, "Non-Scan Design-for-Testability of RTL Data Paths,"*Proc. Int'l Conference on Computer-Aided Design*, pp. 640 - 645, 1994.

[16] S. Dey and M. Potkonjak, "Transforming Behavioral Specifications to Facilitate Synthesis of Testable Designs," *Proc. Int' l Test Conf.*, pp. 184-193, 1994.

[17] D.D. Gajski and L. Ramachandran, "Introduction to High-Level Synthesis," *IEEE Design & Test of Computers*, Vol. 11, No.4, Winter 1994.

[18] X. Gu, K. Kuchcinski, and Z. Peng, "Testability Analysis and Improvement from VHDL Behavioral Specifications, *Proc. EURO-DAC*, 1994.

[19] H. Harmanani and C.A. Papachristou, "An Improved Method for RTL Synthesis with Testability Tradeoffs," *Proc. Int'l Conf. on Computer-Aided Design*, pp. 30-35, 1993.

[20] I.G. Harris and A. Orailoglu, "Microarchitectural Synthesis of VLSI Designs with High Test Concurrency," *Proc. Design Automation Conf.*, pp. 206-211, 1994.

[21] B. Konemann, J. Mucha, and G. Zwiehoff, "Built-In Logic Block Observation Techniques," *Proc. Int' l Test Conference*, pp. 37-41, 1979.

[22] D.H. Lee and S.M. Reddy, "On Determining Scan Flip-Flops in Partial-Scan Designs," *Proc. Int'l Conference on Computer-Aided Design*, pp. 322-325, 1990.

[23] J. Lee and J.H. Patel, "Architectural Level Test Generation for Microprocessors," *IEEE Trans. on Computer-Aided Design*, 13(10):1288-1300, Oct. 1994.

[24] T.-C. Lee, N.K. Jha and W.H. Wolf, "Behavioral Synthesis of Highly Testable Data Paths under Non-Scan and Partial Scan Environments," *Proc. Design Automation Conf.*, 1993.

[25] T.-C. Lee, W.H. Wolf, N.K. Jha and J.M. Acken, "Behavioral Synthesis for Easy Testability in Data Path Allocation," *Proc. Int'l Conf. Computer Design*, 1992.

[26] T.-C. Lee, W.H. Wolf, and N.K. Jha, "Behavioral Synthesis for Easy Testability in Data Path Scheduling," *Proc. Int'l Conf. on Computer-Aided Design*, pp. 616-619, 1992.

[27] A. Majumdar, R. Jain, and K. Saluja, "Incorporating Testability Considerations in High-Level Synthesis, *J. Electronic Testing: Theory & Applications*, pp. 43-55, Feb. 1994.

[28] N. Mukherjee, M. Kassab, J. Rajski, and J. Tsyzer, "Arithmetic Built-In Self Test for High-Level Synthesis," *Proc. 13th IEEE VLSI Test Symp.*, 1995.

[29] B.T. Murray and J.P. Hayes, "Hierarchical Test Generation Using Precomputed Tests for Modules," *Proc. Int'l Test Conf.*, pp. 221-229, 1988.

[30] C. Papachristou and J. Carletta, "Test Synthesis in the Behavioral Domain," *Proc. Int'l Test Conf.*, pp. 693-702, 1995.

[31] C.A. Papachristou, S. Chiu, and H. Harmanani, "A Data Path Synthesis Method for Self-Testable Designs," *Proc. Design Automation Conf.*, pp. 378-384, 1991.

[32] I. Parulkar, S. Gupta, and M.A. Breuer, "Data Path Allocation for Synthesizing RTL Designs with Low BIST Area Overhead," *Proc. Design Automation Conf.*, pp. 395-401, 1995.

[33] M. Potkonjak, S. Dey, and R. Roy, "Behavioral Synthesis of Area-Efficient Testable Designs Using Interaction Between Hardware Sharing and Partial Scan," *IEEE Transactions on Computer-Aided Design*, 14(9):1141-1154, Sept. 1995.

[34] M. Potkonjak, S. Dey and R. Roy," Considering Testability at Behavioral Level: Use of Transformations for Partial Scan Cost Minimization Under Timing and Area Constraints," *IEEE Transactions on Computer-Aided Design*, 14(5):531-546,1995.

[35] J. Steensma, F. Catthoor, and H. De Man, "Partial Scan at the Register-Transfer Level," *Proc. Int'l Test Conf.*, 1991.

[36] Test Synthesis Seminar, *Digest of Papers, IEEE Int'l Test Conference*, 1994.

[37] P. Vishakantaiah, J.A. Abraham, and M. Abadir, "Automatic Test Knowledge Extraction From VHDL (ATKET)," *Proc. Design Automation Conf.*, pp. 273-278, 1992.

[38] P. Vishakantaiah, J.A. Abraham, and D.G. Saab, "CHEETA: Composition of Hierarchical Sequential Tests Using ATKET," *Proc. Int'l Test Conf.*, 1993.

[39] P. Vishakantaiah, T. Thomas, J.A. Abraham, and M.S. Abadir, "AMBIANT: Automatic Generation of Behavioral Modifications for Testability," *Proc. ICCD*, pp. 63-66, 1993.

[40] Walker, R.A., and S. Chaudhuri, "Introduction to the Scheduling Problem," *IEEE Design & Test of Computers*, Vol. 12,No.2, Summer 1995.