ABSTRACT: Considering testability during the early stages of the design flow can have several benefits, including significantly improved fault coverage, reduced test hardware overheads, and reduced design iteration times. This paper presents an overview of high-level design methodologies that consider testability during the early (behavior and architecture) stages of the design flow, and their testability benefits. The topics reviewed include behavioral and RTL test synthesis approaches that generate easily testable implementations targetting ATPG (full and partial scan) and BIST methodologies, and techniques to use high-level information for ATPG.

I. Introduction

The growing complexities of integrated circuits and aggressive time-to-market requirements, coupled with the improvements in design quality that result from exploring high-level tradeoffs, are driving the trend towards designing from behavioral and register-transfer level (RTL) descriptions.

A behavioral description contains an algorithmic specification of the design’s functionality, and may contain little or no information about the design’s cycle-by-cycle behavior or structural implementation. Behavioral synthesis tools typically compile a behavioral description into a suitable intermediate format, such as Control-Data Flow Graph (CDFG). Vertices in the CDFG represent the various operations of the behavioral description, and data and control edges are used to represent the data dependencies between operations and the flow of control. Behavioral synthesis converts a behavioral description into a structural, RTL implementation that is described as an interconnection of macro blocks (e.g. functional units, registers, multiplexers, buses, memory blocks, etc.), and random logic.

High-level synthesis tools typically perform one or more of the following tasks: transformations, module selection, clock selection, scheduling, resource allocation and assignment (also called resource sharing or hardware sharing). The process of scheduling determines the cycle-by-cycle behavior of the design by assigning each operation to one or more clock cycles or control steps. Allocation decides the number of hardware resources of each type that will be used to implement the behavioral description, and assignment refers to the binding of each variable (operation) to one of the allocated registers (functional units). A comprehensive survey on high level synthesis techniques can be found in [7, 14].

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

II. 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 synthesis process.

A. Sequential ATPG Objectives

It has been empirically observed [9, 23] 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 flip-flops (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 satisfying the performance, power, and area constraints of the design.

B. 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 registers connected to primary I/O [26]. 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 [26] 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 [27] 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.

C. 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.

**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 [37], 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 [25]. 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.

**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 [37].

When the operations along a CDFG path from operation \( u \) to operation \( v \) are assigned to \( n \) 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 [37].

Formation of loops in the data path may be avoided by proper scheduling and assignment. Consider the following schedule and assignment which satisfies the given performance and resource constraints: \( \{ +1:(1,A1), +2:(2,A1), +3:(1,A2), +4:(2,A2), +5:(3,A1) \} \). 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.

![Figure 1: Loops formed during assignment: (a) Example CDFG, (b) Assignment Loop, (c) No Loops except Self-Loops](image-url)
D. 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 [8], 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 [40] 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 [42]. 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 [13], 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 [38]. The overall effect is that synthesizing a testable data path from the transformed specification requires fewer scan registers than needed for the original specification.

A behavior level controllability enhancement technique that focuses on control-flow is presented in [21]. A decision node (conditional operation) in the CDFG is said to be $K$-controllable if its outcome can be controlled directly or indirectly from the primary inputs within $K$ clock cycles. Decision nodes with high values of $K$ are identified as candidates for controllability insertion. Controllability insertion is performed by adding a test point that forces the outcome of a decision node to True or False, or complement its value, in the test mode. For designs with complex control flow constructs such as nested conditionals and data-dependent loops, the addition of a few such control points can significantly improve the fault coverage and test generation time of a gate-level sequential ATPG tool on the synthesized netlist [21].

E. 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 [11]. The controller may be redesigned so 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 testability of registers is determined by analyzing the control logic used to control the loading of the registers.

III. RTL Synthesis for Testability

A. RTL Modification and Analysis for Testability

There are several alternatives to enhance RTL descriptions for testability. The description can be augmented to improve testability by re-wiring 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 full scan designs [5].

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 [10]. 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 [39], 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.

B. 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 at the RT level, 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.

Structural and functional knowledge embedded in an RTL description has been used for non-scan DFT schemes like test point insertion [12]. 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 if 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.

Knowledge of the circuit functionality can also be used to reduce significantly the overheads associated with scan design [6, 31]. For example, functional paths between registers, primary inputs, and primary outputs in the circuit can be configured to operate as scan paths in the test mode. A technique to maximally utilize such functional paths in forming parallel scan chains is presented in [6], and is shown to lead to large reductions in area and test application time for full scan designs. When a complete set of scan paths cannot be formed by using functional paths between registers, a minimal number of registers are scanned to complete the scan paths. Alternatively multiple scan configurations can be used to shift values into and out of registers in phases, requiring fewer registers to be scanned [31]. The hardware sharing process also affects the number of scan registers and number of scan configurations required [31].

### IV. 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 pseudo-random 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) [22]. 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 full scan into the LB that can be reconfigured to allow initialization and loading/unloading of stimulus and response data during BIST.

#### A. 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 [2, 3, 19, 34].

In [2], 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 [34], 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, and 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. Testable data paths with even fewer TFBs can be formed by using an extended TFB (XTFB), which contains an ALU with multiple 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 testable data paths with less test area overhead than either the traditional high level synthesis techniques or the BIST register assignment approach [2]. 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 [35]. 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 [35]. The register assignment phase can check the conditions and try to avoid assignments leading to CBILBOs, whenever possible.

### B. 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 use 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 [35], 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.

### C. Adding Test Behavior

A general BIST scheme is proposed in [33], 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 behaviors 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.

A technique to generate the test behavior using redundant register transfers in presented in [32]. The data path obtained as a result of behavioral synthesis may be able to execute register transfers that are not part of the behavioral description, by simply assigning appropriate values at the control signals. Such register transfers are referred to as redundant register transfers. Redundant register transfers may be used to improve the randomness of signals in the data path, and require a modification only to the controller [32]. In [36], a technique to resolve test path conflicts using redundant paths is described. Identity operations are added to the CDFG and assigned to execute on functional units in clock cycles where they are not required to perform any operations from the original CDFG. Such operations may be used to eliminate test conflicts, and enable sharing of test registers, leading to a reduction in BIST overheads.

### D. 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. A high level synthesis methodology has been proposed in [29] 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.

### V. High Level Synthesis and Test Generation

Several techniques have been proposed to generate test vectors using the high level description of a design [1, 4, 8, 24, 30, 41]. We briefly review here two approaches which use high level design information to help in test generation. In [41], 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 to generate test vectors that can be applied at the primary inputs of the hierarchical design [40]. 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. Pre-computed 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 environment of an operation assigned to a module can be used as the test environment for the module. In [4], 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 [17]. 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. For RTL designs where a behavioral description is not available, partial behavioral information can be extracted using a symbolic analysis of the design, and used to derive test environments for each module [15]. The test environment based approach can also be extended to programmable data paths, where the aim is to generate microcode for the test programs, as well as data input patterns to test each module in the design [16]. Inter-module testing often remains a problem in such a macro test environment.
VI. Conclusions

This paper has presented an overview of synthesis for testability at the behavior and register-transfer levels. 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.

References