Test Requirement Analysis for Low-Cost Hierarchical Test Path Construction [*]

Yiorgos Makris
EE Department - Yale University
New Haven, CT 06520
yiorgos.makris@yale.edu

Alex Orailoglu
CSE Department - U.C. San Diego
La Jolla, CA 92093
alex@cs.ucsd.edu

Abstract

We propose a methodology that examines design modules and identifies appropriate vector justification and response propagation requirements for hierarchical test. Based on a cell-level analysis and transparency composition methodology, test requirements for a module are defined as a set of fine-grained input and output bit clusters and pertinent justification and propagation values. The identified test requirements are independent of the actual test set and are adjusted to the cell-level connectivity and inherent regularity of the module. As a result, they combine the generality required for fast hierarchical test path construction with the accuracy necessary for minimizing the incurred DFT hardware overhead, thus fostering cost-effective hierarchical test. Experimental results on several modules verify the ability of the proposed methodology to moderate the cost of hierarchical test path construction through accurate, compact, and highly parametrizable test requirement definition.

1. Introduction

The ability of hierarchical methodologies to reduce test generation complexity has been counterweighed by the hardware overhead incurred for accessing and testing individually each module in the design. The attainment of hierarchical test is thus moderated and its applicability is reduced. Such overhead is attributed to two closely related tasks, namely, the construction of hierarchical test paths and the definition of test requirements for each module.

Due to complexity considerations, hierarchical test path construction methods [1, 2, 3, 4] operate on high-level design descriptions. Raising the level of abstraction, however, comes with a loss in precision, which in this case limits such methods to the identification of only a few coarse hierarchical test paths. As a result, the module under test is treated as a black box and no information pertaining to its inherent test requirements is utilized, implicitly assuming that all possible vectors and responses need to be justified and propagated, respectively. Yet almost never are all possible vectors and all possible responses required for testing a module. Furthermore, transparent accessibility to the full input and output set of each module is rarely inherent in designs. Consequently, this overkill in test requirement definition results in excessive and controversial hardware overhead in order to establish transparent hierarchical test paths.

Moderating the cost of hierarchical test path construction through an informed test requirement definition calls for an analysis methodology that satisfies the constraints imposed by the overall flow of hierarchical test shown in Figure 1. Since lack of appropriate vector justification and response propagation behavior will result in costly DFT hardware, test requirements need to be precise, almost resembling the actual test. The complexity of justifying and propagating exact test, however, is equivalent to the complexity of full circuit test generation. Therefore, hierarchical test methods rely on the construction of hierarchical test paths capable of justifying and propagating all vectors and responses. Consequently, test requirements also need to be general, almost resembling the symbolic nature of hierarchical test paths.

Tackling this trade-off between general and precise test requirement definition requires an understanding of the severity imposed by test requirements on the construction of hierarchical test paths. An efficient test requirement identification method should take this assessment into account, adjusting accordingly the generality of the identified test requirements and thus the number and granularity of the necessary hierarchical test paths. Furthermore, potential regularity in the module connectivity should be exploited and test requirements should be defined in a compact and parametrized fashion.

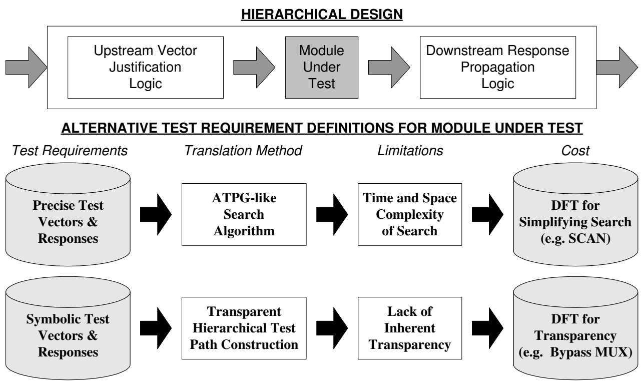

HIERARCHICAL DESIGN
Upstream Vector Justification Logic
Module Under Test
Downstream Response Propagation Logic
ALTERNATIVE TEST REQUIREMENT DEFINITIONS FOR MODULE UNDER TEST
Test Requirements
Translation Method
Limitations
Cost
Precise Test Vectors & Responses
ATPG-like Search Algorithm
Time and Space Complexity of Search
DFT for Simplifying Search (e.g. SCAN)
Symbolic Test Vectors & Responses
Transparent Hierarchical Test Path Construction
Lack of Inherent Transparency
DFT for Transparency (e.g. Bypass MUX)

Figure 1. Precise vs. Symbolic Test Requirements

[*]This work is supported in part through a research grant from Intel Corporation and the University of California MICRO program.

In this paper, we propose a test requirement analysis methodology that exploits recent research efforts in fine-grained transparency extraction and hierarchical test path construction [5] in order to meet the aforementioned objectives. A cell-level analysis and a transparency-based symbolic path composition result in definition of test requirements as a set of sub-word input and output bit clusters. Cell-level analysis supports compactness of the identified test requirements, while sub-word bit clusters enhance their accuracy and symbolic paths guarantee their generality.

Related work is discussed in Section 2, followed by an examination of the severity imposed by test requirements on hierarchical test path construction in Section 3. The proposed test requirement identification methodology is introduced in Section 4 and the appropriate granularity of basic cells is discussed in Section 5. Adjustment of test requirement granularity to the inter-cell connectivity structure is examined in Section 6 and transparency-based hierarchical test path identification is described in Section 7. The proposed methodology is demonstrated through examples in Section 8 and hierarchical test path severity metrics along with experimental results are provided in Section 9.

2. Related Work

While previous methods in hierarchical test path construction have not taken into consideration the inherent test needs of the module under test (MUT), a number of approaches that resemble closely the test requirement identification problem exist in related fields. The objective of test requirement identification for hierarchical test, however, is different than the traditional concept of C-Testability [6] commonly used in Built-In Self-Test and Iterative Logic Array Test. The objective of test requirement identification for BIST [7, 8, 9, 10], for example, is to derive a compact test set that can be easily generated on chip. Analogously, the objective of test requirement identification for ILA test [11, 12, 13] is to exploit regularity and combine test application across cells, minimizing the total test application time. In contrast to these approaches, the objective of the proposed methodology is to identify test requirements that reduce the severity imposed on hierarchical test path construction and thus the corresponding hardware overhead.

3. Hierarchical Test Path Severity

Hierarchical test methods rely on symbolic paths for translating the test set of a module into global design test instead of performing vector-by-vector test translation. Establishing symbolic paths capable of justifying all vectors, however, imposes strenuous requirements on the surrounding logic and has a direct impact on the incurred DFT overhead. As a result, hierarchical test methods have been criticized for the unnecessary generalization of the test requirements. But is it always the case that symbolic test requirements defined on a particular bit cluster impose more demanding requirements on hierarchical test paths than a set of exact test vectors?

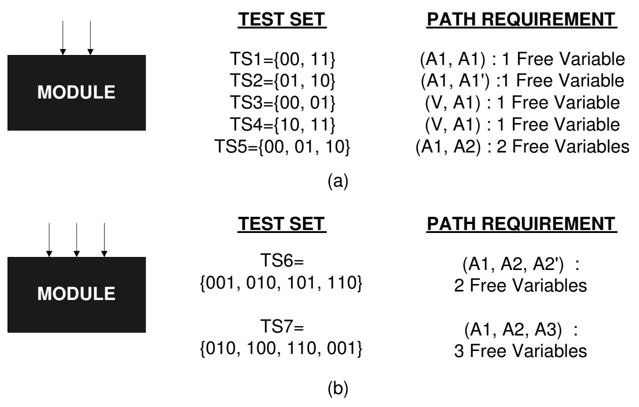


Figure 2. Test Requirement Severity Examples

Answering this question requires an understanding of the severity incurred by an exact test vector set on hierarchical test approaches. Given a set of k-bit test vectors and depending on the number and the distribution of values appearing on each subset of the k-bits, certain restrictions are imposed on the number of primary inputs required and the degrees of freedom necessary between them. Bits that obtain always identical or always inverse values throughout the test set only require one free variable. It is evident that the number of free variables required for a particular set of test vectors is not always equal to the width of the vectors. But at a certain set density point, the full width is required; essentially, we can see that once more than half of the possible values are in the set, no bit can be inferred from the rest, necessitating k free variables for justifying the test set.

Consider for example the 2-input module of Figure 2(a) and the provided alternative test sets. Test requirements for each bit may be either a constant $'0'$ or $'1'$ value symbolically represented by $'V'$, or a free variable represented by $'A'$. Bits that are not required in a test set are represented by $'X'$. For TS_1, a hierarchical test path with one free variable, A_1, at its inputs could be sufficient for satisfying the test requirements. The same observation is valid for test set TS_2. For TS_3 and TS_4, a hierarchical test path with one free variable, A_1, and a constant at its inputs would again be sufficient. Once a test set with 3 vectors is reached, however, such as TS_5, a 2-bit hierarchical test path with no correlation between the bits is necessary. This requirement is almost as severe as requiring two free variables, A_1 and A_2, through the hierarchical test path. In practice, a 2-bit path for providing the two free variables is what hierarchical test methods would establish in this case.

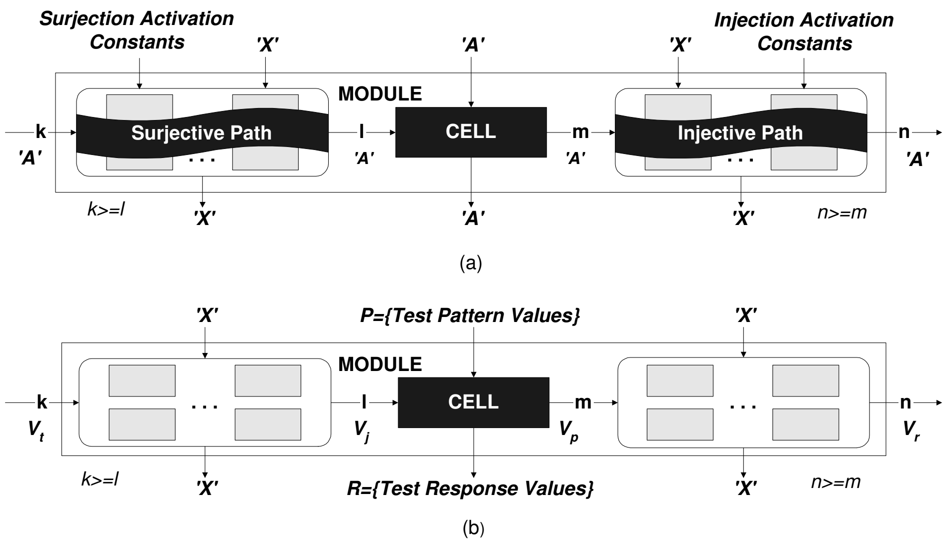


Figure 3. Proposed Methodology vs. Exhaustive Test Requirement Analysis

Similarly, for the 3-input module of Figure 2(b), TS_6 can be satisfied through a hierarchical test path with two free variables, A_1 and A_2, at its inputs. However, TS_7 requires three free variables, A_1, A_2, and A_3. A careful observation reveals that unlike in TS_6, in TS_7 every 2-bit subset of the required bits obtains more than half of the possible values, thus necessitating a 2-bit hierarchical test path with no correlation between the inputs. Since this holds for every subset, from a hierarchical test path construction perspective, the severity of the combined requirement is equivalent to a full 3-bit symbolic path.

Given the symbolic nature of hierarchical test translation paths, there is, evidently, a threshold for the number and distribution of vectors in a test set, over and above which the severity of the corresponding hierarchical test path is equal to that of the full symbolic path. Generalizing the above observations leads to the following condition:

- **Hierarchical Test Path Severity Threshold Condition:** *The hierarchical test path construction severity of a set of k-bit vectors is equivalent to a k-bit symbolic path if every subset of bits obtains more than half of the possible values.*

The above condition signifies when exact test vectors can be safely relaxed into symbolic test requirements, providing a starting point for the test requirement identification methodology discussed in the following section.

4. Proposed Methodology

In order to identify fine-grained test requirements, the proposed methodology targets each basic cell in a module and requires that free variables be justified to all the inputs and propagated from all the outputs of the cell. These symbolic requirements are subsequently translated into a set of sufficient module inputs and outputs to be controlled and observed respectively. As shown in Figure 3(a), inputs or outputs of the cell that are also inputs or outputs of the module are directly assigned a free variable requirement $'A'$. However, there are also l cell inputs and m cell outputs that need to be justified and propagated through the surrounding cells. A transparency composition scheme is employed, identifying a surjective path from k module inputs to the l cell inputs, where $k \geq l$, and an injective path from the m cell outputs to n module outputs, where $n \geq m$. To activate these transparency functions, a number of module inputs have to be set to particular constants. The resulting justification requirements for the module inputs are either a constant $'0'$ or $'1'$, or a free variable $'A'$, while the propagation requirements for the module outputs are free variables $'A'$, since the good machine/bad machine response pair has to be always distinguishable. The remaining module inputs and outputs are assigned to $'X'$.

The transparency-based scheme is a relaxed test requirement analysis for the cell. An exact methodology should be capable of identifying a reduced set of module inputs and outputs through which all test vectors and responses required at the inputs and outputs of the cell can be justified and propagated. In Figure 3(b) for example, the cell inputs that are also module inputs should be assigned to P, the set of required test vectors. Similarly, the cell outputs that are also module outputs should be assigned to R, the set of required test responses. For the l cell inputs and m cell outputs that are justified and propagated through the surrounding cells, exact analysis is more complicated. Assume that V_j, $V_j \subseteq 2^l$, is the set of values that need to be justified to these l inputs of the cell, according to the test vectors.

Similarly, assume that V_p, $V_p \subseteq 2^m$, is the set of values that needs to be distinguishably propagated from these m outputs of the cell, according to the test responses. Essentially, a set V_t, $V_t \subseteq 2^k$, and a set V_r, $V_r \subseteq 2^m$, such that $|V_j| = |V_t|$ and $|V_p| = |V_r|$, are required, with the module implementing a function f from V_j to V_t and a function g from V_p to V_r. Then, V_t and V_r would be the remaining test requirements at the module inputs and outputs.

Such a value-based reasoning for identifying the sets V_j, V_t, V_p, and V_r, providing exact test requirements, is overly time-consuming. Furthermore, as discussed in the previous section, exact vectors do not always impose less severe constraints on hierarchical test path construction. Therefore, the transparency-based scheme described above is employed, resulting in a simpler and faster identification of test requirements, defined as combinations of constant values $'0'$ and $'1'$, symbolic values $'A'$ and don't care values $'X'$ on input and output bit clusters of the module. Yet the success of the proposed methodology depends on the choice of the cell granularity and the identification of surjective and injective paths through the surrounding cells. These issues are discussed in the following sections, where detailed solutions are proposed.

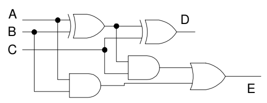
5. Cell Granularity

The granularity of the basic cell on which the proposed analysis is performed is crucial to the severity imposed by the identified test requirements on hierarchical test paths. Appropriate selection of the cell granularity is based on several factors. First of all, the selected cell should satisfy the severity threshold condition, as it guarantees the accuracy of symbolic test requirements at the boundary of the cell. Repetitive cell structures should also be considered, since they result in regular and highly parametrizable test requirements. The size and the number of cells should also be taken into account, due to their direct impact on the complexity of test requirement identification.

An examination of several basic cells commonly found in standard design modules, reveals that they constitute the appropriate granularity level where hierarchical test requirement identification should be performed. More specifically, due to the dense connectivity structure within such basic cells, as compared to the sparser inter-cell connectivity, gate-level test requirements satisfy the severity threshold condition of Section 3. Therefore, they impose the same severity on hierarchical test paths as the full symbolic path, to which they can safely be relaxed. Figure 4 shows examples of four cells and the corresponding gate-level tests[1] which, as demonstrated, satisfy the severity threshold condition. Cells of this granularity level are the basic compo-

[1]Tests were generated using ATALANTA [14] with the random fill option turned off.

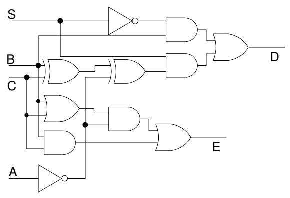
FULL ADDER CELL



Vectors: ABC	Responses: DE
100	10
010	01
001	00
101	
110	
000	

Hierarchical Test Path Severity:
Justify 'AAA' at ABC
Propagate 'AA' from DE

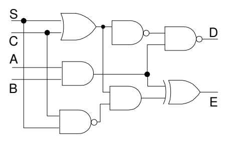
RESTORING DIVIDER CELL



Vectors: ABCS	ABCS	Responses: DE
010X	0001	01
111X	1000	00
110X	0011	10
0110	1011	11
0111	1001	

Hierarchical Test Path Severity:
Justify 'AAAA' at ABCS
Propagate 'AA' from DE

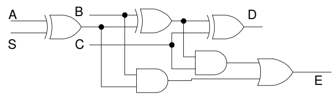
MULTIPLY-ADD CELL



Vectors: ABCS	Responses: DE
0X01	01
0100	00
1000	10
0X10	
1101	
0X11	
1100	

Hierarchical Test Path Severity:
Justify 'AAAA' at ABCS
Propagate 'AA' from DE

NON-RESTORING DIVIDER CELL



Vectors: ABCS	Responses: DE
1000	01
0000	00
0101	10
1100	
1010	
0010	
0100	

Hierarchical Test Path Severity:
Justify 'AAAA' at ABCS
Propagate 'AA' from DE

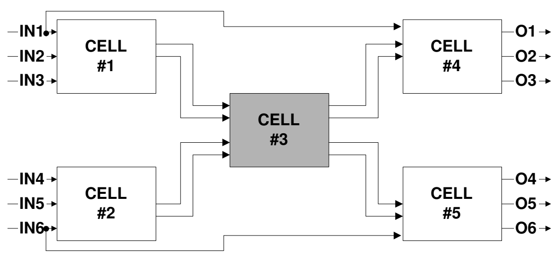
Figure 4. Cells Satisfying the Severity Threshold

nents of several arithmetic circuits, such as adders, array multipliers, restoring and non-restoring array dividers, and square-rooters [15].

Being the minimum repetitive entities in the design, basic cells allow exploitation of possible regularity and reduction of analysis complexity, without necessarily relying on homogeneous designs. With the exception of boundary cells, only prototypical cells need to be analyzed and the corresponding test requirements are defined in a parametrized way, in order to reduce the database storage required. Furthermore, regular requirements incur regular DFT, which can be combined across the requirements of several cells and be highly optimized. These benefits are further demonstrated through examples in Section 8.

6. Test Requirement Granularity Adjustment

The above analysis justifies that test requirement identification at a finer granularity than the basic cell level does not provide hierarchical test path severity reduction. Depending on the inter-cell connectivity, however, the derived test requirements across several cells may also satisfy the severity threshold condition. Therefore, the test requirement gran-



EXAMPLE TEST REQUIREMENTS:

	IN1	IN2	IN3	IN4	IN5	IN6	O1	O2	O3	O4	O5	O6
CELL #1:	A	A	A	V	V	X	A	A	X	X	X	X
CELL #2:	V	V	X	A	A	A	X	X	A	A	A	A
CELL #3:	A	A	A	A	A	A	A	A	A	A	A	A
CELL #4:	A	A	A	V	V	X	A	A	X	X	X	X
CELL #5:	V	V	X	A	A	A	X	X	A	A	A	A
Combined Requirement:	A	A	A	A	A	A	A	A	A	A	A	A

Figure 5. Granularity Adjustment Example

ularity should be adjusted accordingly, resulting in a compact set of test requirements that are as symbolic as possible but do not increase the severity imposed on hierarchical test path construction.

The proposed methodology considers the inter-cell connectivity and adjusts the granularity level accordingly, through a structural analysis of the requirements and the paths for each cell. If the paths required for accessing and testing a cell fully incorporate additional cells, then the cells are combined and test is applied concurrently to them. For example, consider the connectivity structure shown in Figure 5. The surjective and injective paths required for testing cell ♯3 fully incorporate cells ♯1, ♯2, ♯4, and ♯5. It is therefore wise to combine test application for all five cells, since no additional severity is imposed on the corresponding hierarchical test paths.

Another way of explaining this is that the requirements for testing cells ♯1, ♯2, ♯4, and ♯5 are subsets of the requirements for testing cell ♯3, and can therefore be discarded. Consequently, once the test requirements for each cell are identified at the module boundary, an additional granularity adjustment is made. The severity threshold condition is examined across the test requirements. If the condition is satisfied, constant values are relaxed into symbolic paths. Thus, the accuracy necessary for minimizing the hierarchical test path severity is complemented with the generality required for fast hierarchical test path construction.

7. Transparency Path Composition

Within the context of hierarchical test, transparency has been defined as *surjective* functions for justifying test vectors to the inputs of the module under test and *injective* functions for propagating test responses from the outputs of the module under test. Surjective and injective functions are referred to in the literature as *S-Paths* and *F-Paths* respectively [16], while bijective functions satisfying both properties are referred to as *I-Paths* and *T-Paths* [17]. Several variations of surjective, injective, and bijective functions, including *Ambiguity Sets* [1], *Transparency Modes* [2], and *Transparency Properties* [18], have also been used.

The proposed test requirement identification methodology relies on the ability to identify transparency functions through the surrounding cells. Gate-level transparency extraction, however, is a computationally hard problem that limits the applicability of exhaustive algorithms to very small circuits. Consequently, a non-exhaustive method capable of rapidly extracting a wide class of transparency functions is required. Such a transparency extraction method is the outcome of recent research results in transparency composition outlined in [5]. The only difference is that instead of extracting transparency functions from the inputs to the outputs of the module, transparency functions should now be extracted from the inputs to internal signals and from internal signals to the outputs. In all other respects, the methodology proposed in [5] is directly applicable and only the key points are repeated here for the purpose of completeness.

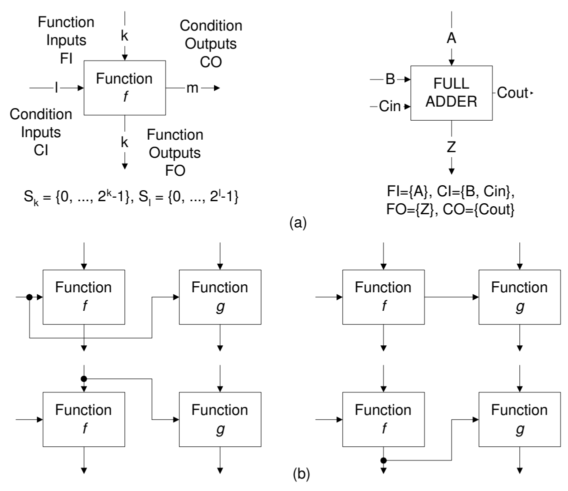


Figure 6. Cell Function Definition and Implications

The proposed transparency composition method is based solely on function classes and not the actual functions. Cell functions are distinguished into four classes according to their inherent transparency behavior and transparency composition is examined through combinations of classes. Consider a function, f, implemented by a cell, as shown in Figure 6(a). The cell has a set of function inputs FI and an additional set of condition inputs CI that activate the cell function. Additionally, the cell has a set of function outputs, FO, and an additional set of collateral outputs, CO, whose value may be either constant or variable for the values of FI and CI. Such a function could be defined for example on a full adder cell, with $FI = (A)$, $CI = (B, C_{in})$,

$FO = (Z)$, and $CO = (C_{out})$. Functions implemented by such cells are not necessarily independent of each other, as the possible interconnection structures of Figure 6(b) reveal. Furthermore, this dependence, which we refer to as *implication*, may be bi-directional. The implication is effected through the condition inputs CI, which may be driven by the CI, CO, or FO of the implicating cell.

In order to study the possible composition of two functions, f and g, into a bijective function, we categorize the function of Figure 6(a) into one of four classes, based on whether it is bijective and on how the implication through the CIs affects the bijection. In the following definitions, S_k and S_l are the sets of all possible values of k-bit and l-bit signals, respectively.

- **Type** $\sharp 1$: *The function is bijective and the bijection is independent of the implication through CIs.*

$$\bigcup_{\forall x \in S_k} f(x) = S_k \tag{1}$$

- **Type** $\sharp 2$: *The function is bijective for each constant value on the CIs, but the bijection depends on the constant.*

$$\forall y \in S_l : \bigcup_{\forall x \in S_k} f(x, y) = S_k \tag{2}$$

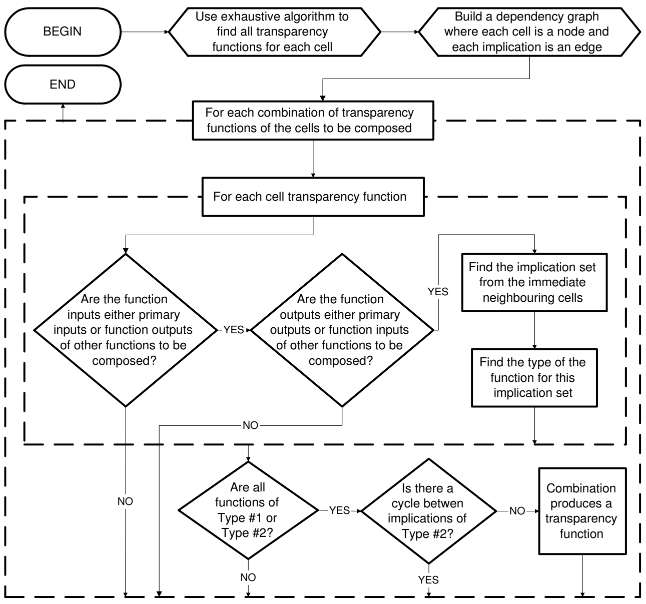
- **Type** $\sharp 3$: *The function is bijective for some but not all constant values on the CIs; the bijection may depend on the constant. There exists at least one constant value for which the function is not bijective.*

$$\begin{gathered} \exists y \in S_l : \bigcup_{\forall x \in S_k} f(x, y) = S_k \\ \bigwedge \\ \exists y \in S_l : \bigcup_{\forall x \in S_k} f(x, y) \subset S_k \end{gathered} \tag{3}$$

- **Type** $\sharp 4$: *No constant value on the CIs makes the function bijective.*

$$\forall y \in S_l : \bigcup_{\forall x \in S_k} f(x, y) \subset S_k \tag{4}$$

While bijections can be potentially composed out of any combination of function types, participation of Type $\sharp 4$ functions in bijection composition consistently results in exponential complexity. As Type $\sharp 4$ functions are inherently not bijective for any constant implication from surrounding functions, they require exhaustive analysis of the composed function. Such bijections are consequently omitted by the proposed method. However, a wide class of transparency functions can be rapidly composed out of the first three types based on the following condition:



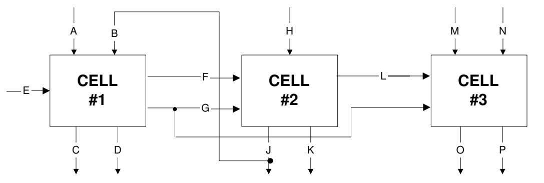
BEGIN
Use exhaustive algorithm to find all transparency functions for each cell
Build a dependency graph where each cell is a node and each implication is an edge
END
For each combination of transparency functions of the cells to be composed
For each cell transparency function
Are the function inputs either primary inputs or function outputs of other functions to be composed?
YES
Are the function outputs either primary outputs or function inputs of other functions to be composed?
YES
Find the implication set from the immediate neighbouring cells
Find the type of the function for this implication set
NO
NO
Are all functions of Type #1 or Type #2?
YES
Is there a cycle betwen implications of Type #2?
NO
Combination produces a transparency function
NO
YES

Figure 7. Transparency Composition Algorithm

- **Transparency Composition Condition:** *The composition of Type $\sharp 1$, Type $\sharp 2$, and Type $\sharp 3$ functions yields a bijection if there exists no cyclic set of implications between functions of Type $\sharp 2$ and Type $\sharp 3$, and every participating Type $\sharp 3$ function g is bijective $\forall z \in V_g$, where V_g is the set of values implicated to function g.*

Under the above condition, Type $\sharp 3$ functions reduce to Type $\sharp 1$ or Type $\sharp 2$ due to the restricted implication set from the surrounding functions. In addition, bijection composition is guaranteed by the acyclicity in the set of implications. The condition is simple to check, facilitating an efficient transparency extraction algorithm described in Figure 7. The condition is only sufficient but not necessary; therefore some transparency functions composed from the above types will be omitted. Nevertheless, the algorithm is capable of rapidly extracting a very wide class of transparency functions, comprising bit cluster level bijections, surjections, and injections.

Using this transparency composition methodology, the surjective and injective paths required for each cell may be identified. Consider for example the circuit shown in Figure 8. In order to access cell $\sharp 3$, a 2-bit transparency path to GL through cells $\sharp 1$ and $\sharp 2$ is required. Cell $\sharp 1$ provides a 1-bit transparency from E to G and cell $\sharp 2$ provides a 1-bit transparency from H to L. An examination of the implication between the two cells reveals that the function of cell $\sharp 1$ reduces to Type $\sharp 2$, while the function of cell $\sharp 2$ reduces to Type $\sharp 1$. Since no cyclic implication between Type $\sharp 2$ and Type $\sharp 3$ cells exists on the graph, the condition holds and therefore the composed function is a bijection through which the signal GL may be controlled.

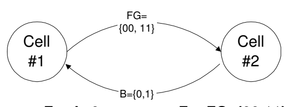


Cell #1: TYPE #3, E bijects to G for AB=00,01 but not for AB=10,11
Cell #2: TYPE #3, H bijects to L for any FG=00,11 but not for FG=01,10

E A B	C D F G
0 0 0	0 0 1 1
0 0 1	0 0 1 0
0 1 0	0 1 1 0
0 1 1	1 0 1 0
1 0 0	0 1 0 0
1 0 1	1 0 0 1
1 1 0	1 1 1 0
1 1 1	0 0 1 0

F G H	J K L
0 0 0	0 0 1
0 0 1	1 1 0
0 1 0	1 0 0
0 1 1	0 0 0
1 0 0	1 1 1
1 0 1	1 0 1
1 1 0	0 1 0
1 1 1	1 1 1

Dependency Graph



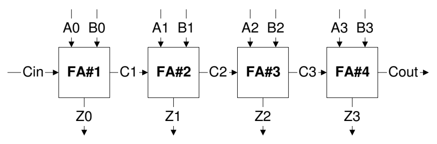
For A=0
Cell #1 is bijective
and becomes a
TYPE #2 function

For FG={00,11}
Cell #2 is bijective
and becomes a
TYPE #1 function

No Cyclic Dependency between TYPE#2 and TYPE #3 so Bijection EL to GH is Composed

Figure 8. Transparency Composition Example

CARRY-RIPPLE ADDER



Test Justification Requirements:

	Cin	A0	B0	A1	B1	A2	B2	A3	B3
FA#1:	A	A	A	V	V	X	X	X	X
FA#2:	X	A	A	A	A	V	V	X	X
FA#3:	X	X	X	A	A	A	A	V	V
FA#4:	X	X	X	X	X	A	A	A	A

Test Propagation Requirements:

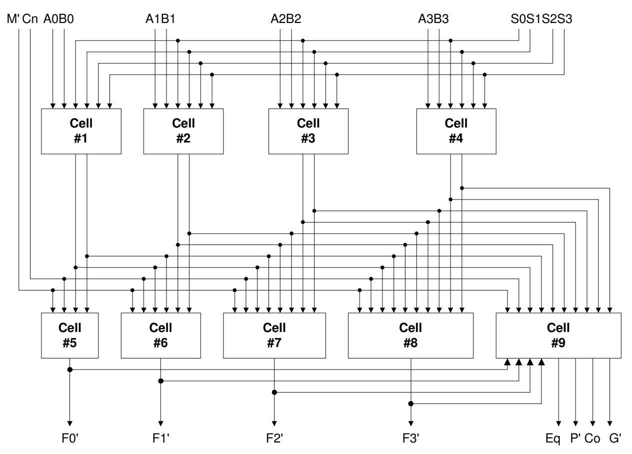
	Z0	Z1	Z2	Z3	Cout
FA#1:	A	A	X	X	X
FA#2:	X	A	A	X	X
FA#3:	X	X	A	A	X
FA#4:	X	X	X	A	A

Figure 9. Test Requirements of Carry-Ripple Adder

8. Examples

The proposed test requirement identification method is demonstrated and evaluated through several example modules in this section. The first module, shown in Figure 9, is a simple 4-bit carry-ripple adder [15], comprising 4 full-adder cells, such as the one shown in Figure 4. Consider for example the test requirements for $FA\sharp3$. According to the proposed methodology of Section 4, $'A'$s are assigned to the inputs and outputs of the cell that are also inputs and outputs of the module, in this case A_2, B_2, and Z_2. The $'A'$ requirement on C_2 is satisfied through a surjective path from A_1, B_1, while the $'A'$ requirement on C_3 is satisfied through an injective path to Z_3, activated by any constant value $'V'$ on A_3, B_3. The remaining inputs and outputs, C_{in}, A_0, B_0, Z_0, Z_1, and C_{out} are assigned $'X'$s. The identified test requirements are symbolic, as necessary for hierarchical test path construction, but also compact and accurate; thus close, in terms of precision, to the minimal hierarchical test requirements of the full adder. Furthermore, the regular structure of the module allows the test requirements to be parametrized; therefore, the analysis is performed only for the prototypical cell and the boundary cases.

74181 ALU



Test Justification Requirements:

	M'	Cn	A0	A1	A2	A3	B0	B1	B2	B3	S0	S1	S2	S3
Cell#1:	V	V	A	X	X	X	A	X	X	X	A	A	A	A
Cell#2:	V	V	V	A	X	X	V	A	X	X	A	A	A	A
Cell#3:	V	V	V	V	A	X	V	V	A	X	A	A	A	A
Cell#4:	V	V	V	V	V	A	V	V	V	A	A	A	A	A
Cell#5:	A	A	A	X	X	X	A	X	X	X	A	V	V	V
Cell#6:	A	A	A	A	X	X	A	A	X	X	A	A	V	V
Cell#7:	A	A	A	A	A	X	A	A	A	X	A	A	A	V
Cell#8:	A	A	A	A	A	A	A	A	A	A	A	A	A	A
Cell#9:	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**	**A**

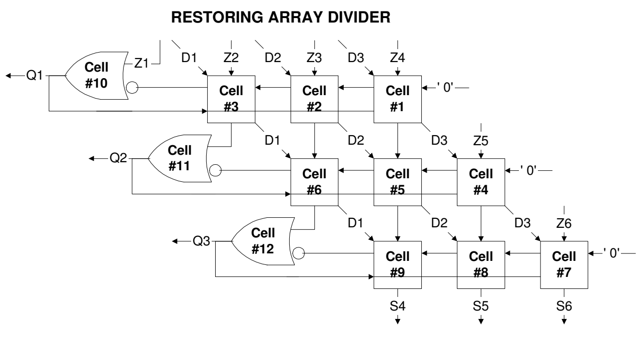
Test Propagation Requirements:

	G'	Co	P'	F3'	F2'	Eq	F1'	F0'
Cell#1:	**X**	**X**	**X**	**X**	**X**	**X**	**A**	**A**
Cell#2:	**X**	**X**	**X**	**X**	**A**	**X**	**A**	**X**
Cell#3:	**X**	**X**	**X**	**A**	**A**	**X**	**X**	**X**
Cell#4:	**X**	**A**	**X**	**A**	**X**	**X**	**X**	**X**
Cell#5:	X	X	X	X	X	X	X	A
Cell#6:	X	X	X	X	X	X	A	X
Cell#7:	X	X	X	X	X	A	X	X
Cell#8:	X	X	X	X	A	X	X	X
Cell#9:	**A**	**A**	**A**	**X**	**X**	**A**	**X**	**X**

Figure 10. Test Requirements of 74181 ALU

The second module is the 74181 ALU [11], which unlike the adder is neither homogeneous, nor regular. This example further demonstrates the ability of the methodology to adjust the identified test requirements to the inter-cell connectivity. The connectivity and the test requirements for each cell are shown in Figure 10. Based on the granularity adjustment methodology of Section 6, the test requirements for the cell pairs $(\sharp1, \sharp5)$, $(\sharp2, \sharp6)$, $(\sharp3, \sharp7)$, and $(\sharp4, \sharp8)$ are merged. Furthermore, establishing a surjective path to the 10 inputs of cell $\sharp9$ requires a hierarchical test path to all 14 inputs of the ALU. Consequently, the test justification requirements for cells $\sharp1$ through $\sharp8$, which are all subsets of the test justification requirements for cell $\sharp9$, are discarded. The final set of test requirements for the ALU is thus adjusted to the module connectivity and is shown in boldface.

The third module is a restoring array divider [15] composed of cells such as the one shown in Figure 4. The circuit and the test requirement analysis are shown in Figure 11. While the module is not homogeneous, its inherent regularity allows parametrization of the test requirements. Consider, for example, the test requirements for cell $\sharp5$. The four inputs of the cells are justified through a surjective path from D_3, Z_4, and Z_5 and a surjective path from D_2 and Z_2. The two outputs of the cell are propagated through injective paths to outputs Q_2, Q_3, and S_4. These inputs and outputs are consequently assigned a test requirement $'A'$. The remaining inputs all require constant values to establish the injective and surjective paths and are, therefore, assigned



Test Justification Requirements:

	Z1Z2Z3Z4Z5Z6D1D2D3
Cell#1:	A V V A V V V V A
Cell#2:	**A V A A V V V A A**
Cell#3:	**A A A V V V A A V**
Cell#4:	V A V V A V V V A
Cell#5:	**V A V A A V V A A**
Cell#6:	**V A A A V V A A V**
Cell#7:	V V A V V A V V A
Cell#8:	**V V A V A A V A A**
Cell#9:	**V V A A A V A A V**
Cell#10:	A A X X X X A X X
Cell#11:	A A A V X X A V X
Cell#12:	X A A A V X A V X

Test Propagation Requirements:

	Q1Q2Q3S4S5S6
Cell#1:	**A A X A X X**
Cell#2:	**A A A X X X**
Cell#3:	A A X X X X
Cell#4:	**X A A X A X**
Cell#5:	**X A A A X X**
Cell#6:	X A A X X X
Cell#7:	**X X A X X A**
Cell#8:	X X A X A X
Cell#9:	X X A A X X
Cell#10:	A X X X X X
Cell#11:	X A X X X X
Cell#12:	X X A X X X

Figure 11. Test Requirements of Array Divider

a test requirement $'V'$, while the remaining outputs are assigned a test requirement $'X'$. The granularity of test requirements is once again adjusted using the methodology of Section 6, and the final set is shown in boldface.

The above examples demonstrate the ability of the proposed methodology to identify symbolic, yet accurate test requirements defined on fine-grained input and output bit clusters. The granularity of test requirements is adjusted to the inter-cell connectivity of the module, in order to further reduce the severity imposed on hierarchical test paths. In addition, while not limited to homogeneous circuits, the methodology exploits inherent cell regularity, in order to parametrize and compact the identified test requirements.

9. Severity Metrics and Experimental Results

The objective of the proposed methodology is to identify test requirements that reduce the severity imposed on hierarchical test and the corresponding testability hardware overhead. To evaluate the burden imposed on hierarchical test paths, the following two metrics are introduced, reflecting the severity of test path existence and test path identification for a module. The underlying assumption for defining the metrics is that the likelihood of path existence and the complexity of path identification decrease, as the generality of the path increases. Path generality increases with the width and with the values attainable at each bit position.

Test Path Existence Severity, reflecting the possibility that testability hardware will be needed to establish transparency paths due to the generality of the test requirements, is defined as

$$TPES(Module) = \sum_{\forall\ Paths} TPES(Path),\ where \tag{5}$$

$$TPES(Path) = \prod_{\forall\ Bits} TPES(Bit),\ and \tag{6}$$

$$TPES(Bit) = \left\{ \begin{array}{lll} 1 & if & 'X' \\ 2 & if & 'V' \\ 4 & if & 'A' \end{array} \right\} \tag{7}$$

Test Path Identification Severity, reflecting the possibility that testability hardware will be needed due to the translation complexity of exact test requirements, is defined as

$$TPIS(Module) = \sum_{\forall\ Paths} TPIS(Path),\ where \tag{8}$$

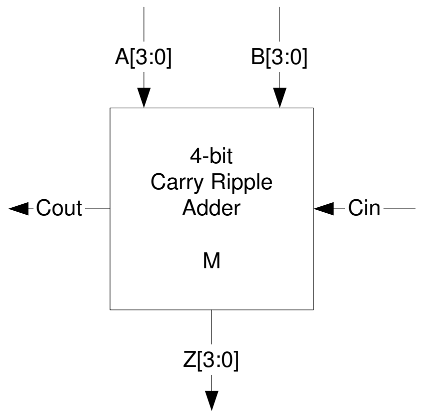
$$TPIS(Path) = \prod_{\forall\ Bits} TPIS(Bit),\ and \tag{9}$$

$$TPIS(Bit) = \left\{ \begin{array}{lll} 1 & if & 'X' \\ 2 & if & 'A' \\ 4 & if & 'V' \end{array} \right\} \tag{10}$$

As an example, Figure 12 calculates the controllability and observability TPES and TPIS of a 4-bit carry-ripple adder for the test requirements imposed by symbolic paths and by compacted gate-level test. Figure 13, further calculates the metrics for the requirements imposed by non-compacted gate-level test and by the proposed methodology. The controllability metrics C-TPES and C-TPIS for the four approaches are summarized in Tables 1 and 2, while the observability metrics O-TPES and O-TPIS are summarized in Tables 3 and 4. Results are also reported in these tables for the restoring divider and the ALU example circuits of the previous section. As demonstrated, the coarseness of the symbolic paths results in very high TPES values, although their generality ensures low TPIS values. On the other hand, the accuracy of exact test ensures low TPES values, yet results in high TPIS values due to the complexity of exact translation. If the test is not compacted the problem is slightly alleviated but the TPIS values are still orders of magnitude higher than the TPES values.

The proposed methodology resolves the problem by combining the generality required for fast hierarchical test path construction with the accuracy necessary for ensuring translatability. As a result, the TPES and TPIS values are of the same order of magnitude[2] and close to the minimal values. Thus, the identified test requirements significantly reduce the overall burden imposed on hierarchical test.

[2]Except for controlling the ALU, where the full symbolic path is required.



TEST PATTERNS - TEST RESPONSES

A3A2A1A0B3B2B1B0Cin	*CoutZ3Z2Z1Z0*
0 0 0 0 1 0 1 1 0	0 1 0 1 1
0 1 0 1 0 1 1 0 1	0 1 1 0 0
1 0 0 1 0 0 1 1 0	0 1 1 0 0
1 1 1 0 0 1 1 0 1	1 0 1 0 1
0 1 0 1 0 0 0 1 0	0 0 1 1 0
1 0 1 0 1 1 1 0 0	1 1 0 0 0
0 0 0 1 1 1 0 0 0	0 1 1 0 1
0 1 1 0 1 0 0 1 0	0 1 1 1 1

8 Distinct Vectors - 7 Distinct Responses

Symbolic Paths

1 Justification Path: "AAAAAAAAA"

C-TPES(M)=4^9=262144
C-TPIS(M)=2^9=512

1 Propagation Path: "AAAAA"

O-TPES=4^5=1024
O-TPIS=2^5=32

Compacted Test

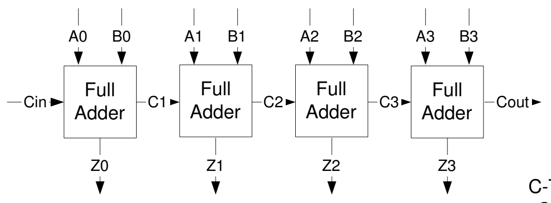
8 Justification Paths: "VVVVVVVVV"

C-TPES(M)=8*2^9=4096
C-TPIS(M)=8*4^9=20197152

7 Propagation Paths: "VVVVV"

O-TPES=7*2^5=224
O-TPIS=7*4^5=7168

Figure 12. Metric Calculation for Symbolic Paths and Compacted Test



TEST PATTERNS - TEST RESPONSES
(RANDOM FILL TURNED OFF)

A3A2A1A0B3B2B1B0Cin	*CoutZ3Z2Z1Z0*
0 1 1 X 0 0 1 X X	0 1 0 X X
X 0 1 1 X 0 0 1 X	X X 1 0 X
0 1 0 X 0 0 0 X X	0 0 1 X X
0 0 0 X 0 1 0 X X	0 0 1 X X
0 0 1 X 0 0 1 X X	0 0 1 X X
X 0 1 0 X 0 0 X 0	X X 0 1 X
X 0 0 0 X 0 1 X 0	X X 0 1 X
X 0 0 1 X 0 0 1 X	X X 0 1 X
X X 0 1 X X 0 0 1	X X X 1 0
X X 0 1 X X 0 0 0	X X X 0 1
X X 0 0 X X 0 1 0	X X X 0 1
X X 0 0 X X 0 0 1	X X X 0 1
0 1 X X 0 1 X X X	0 1 X X X
1 0 X X 1 0 X X X	1 0 X X X
1 0 X X 0 0 X X X	0 1 X X X
0 0 X X 1 0 X X X	0 1 X X X
1 1 X X 0 1 X X X	1 0 X X X

Proposed Methodology

4 Justification Paths:
"XXVAXXVAA"
"XVAAXVAAX"
"VAAXVAAXX"
"AAXXAAXXX"

C-TPES(M)=2^2*4^3+2^2*4^4+2^2*4^4+4^4=2560
C-TPIS(M)=4^2*2^3+4^2*2^4+4^2*2^4+4^2=656

4 Propagation Paths:
"XXXAA"
"XXAAX"
"XAAXX"
"AAXXX"

O-TPES=4^2+4^2+4^2+4^2=64
O-TPIS=2^2+2^2+2^2+2^2=16

Non-Compacted Test
17 Distinct Justification Paths:
(8 have 3 Xs, 4 have 4 Xs, 5 have 5 Xs)

C-TPES(M)=8*2^6+4*2^5+5*2^4=848
C-TPIS(M)=8*4^6+4*4^5+5*4^4*2^4+4^2=38144

7 Distinct Propagation Paths:
(3 have 2 Xs, 4 have 3 Xs)

O-TPES=3*2^3+4*2^2=40
O-TPIS=3*4^3+4*4^2=256

Figure 13. Metric Calculation for Non-Compacted Test and Proposed Method

Table 1. Comparison of C-TPES Metrics

C-TPES Metric	Symbolic Paths	Compacted Test	Non-Compacted Test	Proposed Method
Adder	262144	4096	848	2560
Divider	262144	9216	7360	49152
ALU	238435456	425984	65280	238435456

Table 2. Comparison of C-TPIS Metrics

C-TPIS Metric	Symbolic Paths	Compacted Test	Non-Compacted Test	Proposed Method
Adder	512	20197152	38144	656
Divider	512	4718592	3662468	98304
ALU	16384	6979321856	230430714	16384

Table 3. Comparison of O-TPES Metrics

O-TPES Metric	Symbolic Paths	Compacted Test	Non-Compacted Test	Proposed Method
Adder	1024	224	40	64
Divider	4096	1088	832	36
ALU	65536	5632	4064	32

Table 4. Comparison of O-TPIS Metrics

O-TPIS Metric	Symbolic Paths	Compacted Test	Non-Compacted Test	Proposed Method
Adder	32	7168	256	16
Divider	64	69632	47888	272
ALU	256	1441792	1013856	320

10. Conclusions

Accurate modular test requirement identification is critical to the cost-effectiveness of hierarchical test, since the severity imposed on the corresponding hierarchical test paths is directly related to the anticipated testability hardware overhead. A thorough understanding of the severity imposed by exact test patterns as compared to symbolic test provides the basis for defining appropriate test requirements. The proposed methodology identifies a set of fine-grained, yet adequate input and output bit clusters to be justified and propagated respectively, through which symbolic test can be applied to each basic cell in the module. Through an efficient cell-based transparency extraction approach, the proposed method adjusts the granularity of the identified test requirements to the module connectivity. Furthermore, the identified test requirements are independent of particular test sets and can be parametrized to exploit inherent repetitive structures and regularity in the design, thus reducing the analysis time and the corresponding storage. Most importantly, the identified test requirements combine the generality required for fast hierarchical test path construction with the accuracy necessary for minimizing the corresponding hierarchical test path severity. Thus, the DFT hardware incurred for hierarchical test path construction is reduced, fostering competitive hierarchical test approaches.

References

[1] B. T. Murray and J. P. Hayes, "Hierarchical test generation using precomputed tests for modules," *IEEE Transactions on Computer Aided Design*, vol. 9, no. 6, pp. 594–603, 1990.

[2] P. Vishakantaiah, J. A. Abraham, and D. G. Saab, "CHEETA: Composition of hierarchical sequential tests using ATKET," in *International Test Conference*, 1993, pp. 606–615.

[3] J. Lee and J. H. Patel, "Hierarchical test generation under architectural level functional constraints," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1144–1151, 1997.

[4] R. S. Tupuri, A. Krishnamachary, and J. A. Abraham, "Test generation for gigahertz processors using an automatic functional constraint extractor," in *Design Automation Conference*, 1999, pp. 647–652.

[5] Y. Makris, V. Patel, and A. Orailoglu, "Efficient transparency extraction and utilization in hierarchical test," in *VLSI Test Symposium*, 2001, pp. 246–251.

[6] M. Abramovici, M. A. Breuer, and A. D. Friedman, *Digital Systems Testing and Testable Design*, IEEE Press, 1990.

[7] H. Al-Asaad, J. P. Hayes, and B. T. Murray, "Scalable test generators for high-speed datapath circuits," *Journal of Electronic Testing: Theory and Applications*, vol. 12, no. 1/2, pp. 111–125, 1998.

[8] I. Voyiatzis, A. Paschalis, D. Nikolos, and C. Halatsis, "R-CBIST: An effective RAM-based input vector monitoring concurrent BIST technique," in *International Test Conference*, 1998, pp. 918–925.

[9] K. K. Saluja, R. Sharma, and C. R. Kime, "A concurrent testing technique for digital circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, no. 12, pp. 1250–1260, 1988.

[10] D. Gizopoulos, A. Paschalis, and Y. Zorian, "An effective built-in self-test scheme for parallel multipliers," *IEEE Transactions on Computers*, vol. 48, no. 9, pp. 936–950, 1999.

[11] E. J. McCluskey and S. Bozorgui-Nesbat, "Design for autonomous test," *IEEE Transactions on Computers*, vol. c-30, no. 11, pp. 866–874, 1981.

[12] T. Sridhar and J. P. Hayes, "Design of easily testable bit-sliced systems," *IEEE Transactions on Computers*, vol. c-30, no. 11, pp. 842–854, 1981.

[13] H. Elhuni, A. Vergis, and L. Kinney, "C-Testability of two-dimensional iterative logic arrays," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 5, no. 4, pp. 573–581, 1986.

[14] "ATALANTA combinational test generation tool," Available from `http://www.ee.vt.edu/ha/cadtools`.

[15] B. Parhami, *Computer Arithmetic: Algorithms and Hardware Designs*, Oxford University Press, 1999.

[16] S. Freeman, "Test generation for data-path logic: The F-path method," *IEEE Journal of Solid-State Circuits*, vol. 23, no. 2, pp. 421–427, 1988.

[17] M. S. Abadir and M. A. Breuer, "A knowledge-based system for designing testable VLSI chips," *IEEE Design and Test of Computers*, vol. 2, no. 4, pp. 56–68, 1985.

[18] Y. Makris and A. Orailoglu, "RTL test justification and propagation analysis for modular designs," *Journal of Electronic Testing: Theory and Applications*, vol. 13, no. 2, pp. 105–120, 1998.