JOURNAL OF ELECTRONIC TESTING: Theory and Applications 18, 29–42, 2002

Fast Hierarchical Test Path Construction for Circuits with DFT-Free Controller-Datapath Interface

YIORGOS MAKRIS
EE Department, P.O. Box 208285, Yale University, New Haven, CT 06520, USA
yiorgos.makris@yale.edu

JAMISON COLLINS AND ALEX ORAILOĞLU
CSE Department, MC-0114, UCSD, La Jolla, CA 92093, USA
jdcollin@cs.ucsd.edu
alex@cs.ucsd.edu

Received May 16, 2001; Revised July 14, 2001

Editor: Kuen-Jong Lee

Abstract. Hierarchical approaches address the complexity of test generation through symbolic reachability paths that provide access to the I/Os of each module in a design. However, while transparency behavior suitable for symbolic design traversal can be utilized for constructing reachability paths for datapath modules, control modules do not exhibit transparency. Therefore, incorporating such modules in reachability path construction requires exhaustive search algorithms or expensive DFT hardware. In this paper, we discuss a fast hierarchical test path construction method for circuits with DFT-free controller-datapath interface. A transparency-based RT-Level hierarchical test generation scheme is devised for the datapath, wherein locally generated vectors are translated into global design test. Additionally, the controller is examined through the introduced concept of influence tables, which are used to generate valid control state sequences for testing each module through hierarchical test paths. Fault coverage and vector count levels thus attained match closely those of traditional test generation methods, while sharply reducing the corresponding computational cost and test generation time.

Keywords: controller-datapath circuit, hierarchical test path, transparency, influence tables

1. Introduction

Hierarchy exploitation constitutes the dominant direction along which test methodologies attempt to accommodate modern designs. Size and complexity considerations, along with the modular nature of the emerging System-on-Chip design trend, point towards *divide-&-conquer* test solutions. In principle, hierarchical test [11, 12, 15, 20, 21] employs *modular decomposition* in order to reduce the circuit size and thus improve fault coverage and test generation time. However, such approaches rely on *modular transparency*, the attribute wherein each module can be isolated and treated as a stand-alone entity. This capability is provided by reachability paths from the primary inputs to the inputs of the module under test (MUT) and from the outputs of the MUT to the primary outputs of the design.

Reachability paths allow locally generated test vectors for a MUT to be justified from the primary inputs and the corresponding responses to be propagated to the primary outputs. Translating local into global test, however, is computationally very expensive if performed in a vector-by-vector manner. Therefore, hierarchical test methodologies rely on *symbolic* design traversal for

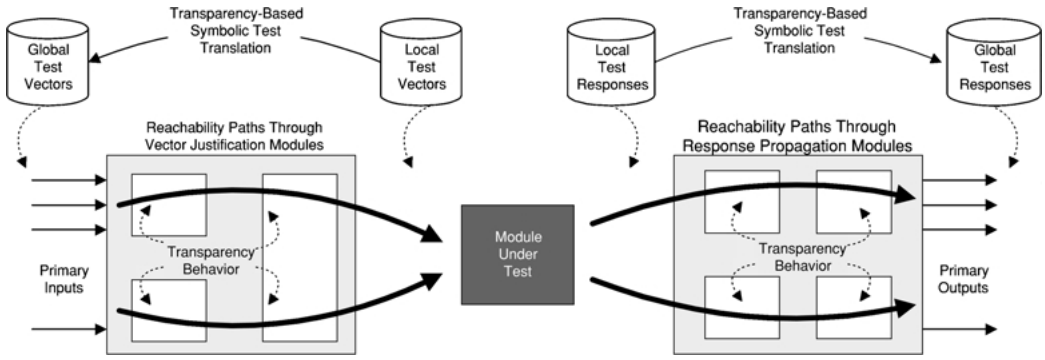


Fig. 1. Hierarchical test generation via transparent reachability paths.

test translation, leveraging on the ability to reason on the logic surrounding the MUT in a bulk fashion, as depicted in Fig. 1. To facilitate symbolic design traversal, reachability paths are constructed using *transparency behavior* [1, 5, 13, 16, 19] of the constituent modules. Transparency behavior enables symbolic vector justification and response propagation. When modules exhibiting transparency behavior are combined into a vector justification (response propagation) path, it appears as if the MUT is directly connected to primary inputs (primary outputs). As a result, any vector may be applied and any response may be unambiguously evaluated.

Reachability paths comprising transparency behavior of the constituent modules are an inseparable part of hierarchical test methods. When transparency is not inherently available in a module or when it cannot be utilized due to path setup condition conflicts, Design-For-Test (DFT) hardware [6, 14] is employed to resolve the problem and thus avoid computationally expensive test translation schemes and significant loss of fault coverage. Nevertheless, the cost and performance impact incurred by DFT calls for thorough reachability path construction. While preserving symbolic design traversal, such paths should exploit as much of the inherent functionality of the design as possible, before resorting to DFT hardware to restore transparency.

In controller-datapath pairs, such as in Fig. 2, the impact of the controller on the datapath needs to be considered. Controllers, however, do not exhibit transparency and therefore require exhaustive FSM analysis, which is computationally very expensive. In most current solutions control logic is not incorporated on reachability paths, which therefore rely on hardware for separating the datapath from the controller. The interface is typically enhanced through DFT [6, 8] or controller redesign [3, 4]. However, the cost of extra area and

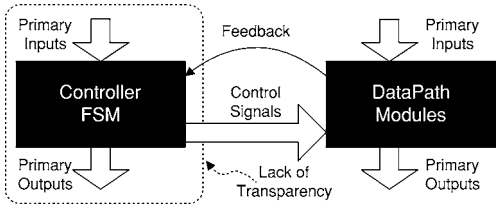


Fig. 2. Controller-datapath pair.

possible critical timing path complications incurred require that a comprehensive, yet non-exhaustive analysis of the controller-datapath interaction be performed, before DFT hardware is employed.

To address these challenges this paper presents a hierarchical test generation solution for controller-datapath pairs that does not presume DFT at the interface. In Section 2, we demonstrate the challenges associated with this task on an example circuit. In Section 3, we present a transparency-based hierarchical test generation approach, using the *transparency channel* definition introduced in [13]. In Section 4, we introduce the concept of *influence tables*, a mechanism for exploring the impact of each controller state on the datapath. Influence tables are combined in order to derive appropriate control state sequences for testing each module and the identified control state sequences are used as constraints that speed up hierarchical test path identification by pruning the search space. Experimental results in support of the proposed combined controller-datapath scheme are provided in Section 5.

2. Motivation

Hierarchical test generation requires that local test be translated to global design test. Test translation on a controller-datapath circuit, however, poses a number of challenges to be addressed. The difficulty of the

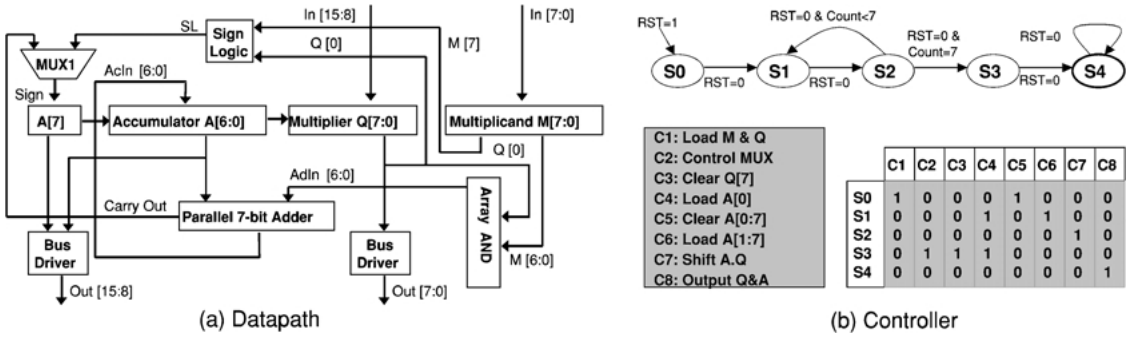


Fig. 3. Controller-datapath implementation of MUL example circuit.

problem is motivated in this section, based on the example circuit of Fig. 3, an 8-bit binary shift-&-add sign-magnitude multiplier described in [7].

2.1. *Datapath Challenges*

Translating the local test in a vector-by-vector manner faces a large search space, making exhaustive search impossible. In the example circuit, reasoning on shifting, adding and clearing is required for testing the ADDER. Automating this process is complicated, and feedback loops, such as the one between the ADDER and the ACCUMULATOR shift-register, only increase complexity. The reconvergent paths from the Q0 signal of the MULTIPLIER through the AND ARRAY, the ADDER and the ACCUMULATOR and through the SIGN, the MUX, the A [5], and the ACCUMULATOR also result in excessive backtracking. Furthermore, word-level reasoning is insufficient to handle signals that split, such as the MULTIPLICAND output M.

Value-based, vector-by-vector test translation defeats the purpose of the hierarchical approach. The benefits of hierarchical test generation arise when test translation is performed through reachability paths. Such paths are constructed using the transparency behavior of the surrounding modules. Transparency provides a trade-off mechanism between the completeness and the complexity of the test translation process. The search is fast but it is performed in a reduced functional space; consequently, some inherent test translation capabilities may be lost. A transparency definition capturing compactly most of the test translation behavior of a module is therefore essential.

2.2. *Controller Challenges*

Many of the reachability path solutions identified on the standalone datapath will be invalid in the combined design, due to the exact sequences of signals imposed by the controller. As shown in the example circuit, the controller generates specific signals and an FSM analysis is required to derive valid control state sequences in support of reachability paths. However, exhaustive FSM analysis is expensive even for simple controllers. Loops in the controller, such as the one between states $S1$ and $S2$, along with feedback signals from the datapath complicate the process of reasoning on exact control state sequences and the corresponding control signal values. The complexity of the problem is equivalent to the exponentially growing complexity of path enumeration on the controller FSM, forcing existing solutions [3, 4, 6, 8] to rely on DFT in order to isolate the datapath from the controller. The area and performance impact incurred, however, necessitates that DFT be used sparingly and only after the inherent design functionality is exploited first. In order to minimize the DFT overhead, a fast, non-exhaustive methodology for examining the control state space and exploring potential solutions needs to be devised.

2.3. *Controller-Datapath Seamless Test*

When DFT is not available at the controller-datapath interface, the following search alternatives exist:

- *Datapath First*: The search is performed on the datapath, and solutions are checked against the controller.
- *Controller First*: The control signals of each valid control sequence are imposed as constraints to the datapath search.
- *Intertwined Search*: Each decision of the datapath search algorithm is checked immediately against the restrictions imposed by the controller.

Although the first two, algorithmically random walk approaches, are simple to implement, they are computationally ineffective. Computational effectiveness can be attained with the third approach, albeit at the expense of implementation complexity. The search is still exhaustive in both the controller and the datapath, but the combined space is pruned concurrently from both sides, thus converging faster. Implementation, however, requires a list of hopeful control state sequences to be kept and updated for each decision.

A fast yet efficient alternative to the aforementioned three approaches is described in this paper. The proposed scheme avoids exhaustive search based on the concepts of *transparency channels* introduced in [13] and *influence tables* introduced herein.

3. Datapath

In this section, we review various datapath transparency definitions and path composition approaches and we devise a RT-Level hierarchical test generation methodology. The datapath challenges described above are addressed by employing fine-grained transparency behavior based on which reachability paths are constructed to provide an accurate and efficient symbolic design traversal mechanism and a concise local to global test translation method.

3.1. Transparency Definitions and Path Composition

Previous work in reachability path construction employs transparency definitions based on the mathematical principles of *surjective*, *injective*, and *bijective* functions. Surjective functions are appropriate for justifying vectors, while injective functions are appropriate for propagating responses; bijective functions are appropriate for both purposes. Abadir and Breuer [1] introduce the concepts of *I-path* and *T-path* for capturing transparency of modules in terms of *Identity* and *Transformation* functions between equal bitwidth signals. The simplicity of *Identity* functions makes the *I-path* the course of choice in hierarchical test generation research. Freeman [5] extends the path notion by introducing the *F-path* and the *S-path* concepts, which are surjective and injective functions, respectively, and are not necessarily defined on equal bitwidth signals. Murray and Hayes [15] propose a hierarchical test generation scheme based on *ambiguity sets* [16]. Complexity considerations limit the applicability of ambiguity sets to small datapath circuits. Vishakantaiah et al. [19] propose an automated test knowledge extraction methodology, wherein transparency is expressed in terms of *modes*. Modes are combined into word level test justification and propagation paths [20], though for simplicity, this work is also limited to *Identity* and *Negation* functions between equal bitwidth signals. Hierarchical test approaches have also been proposed for core-based designs [9, 22].

In our previous work in the area of testability analysis [13], we introduced the concept of *transparency channels* for defining combinational or sequential logic transparency in terms of surjective, injective, or bijective functions. For the purpose of completeness, the definition of transparency channels is provided in Fig. 4, along with examples on RT-Level modules. The innovative feature of transparency channels is the ability to express and combine transparency behavior defined on variable bitwidth signal entities. Unlike previous approaches that construct coarse,

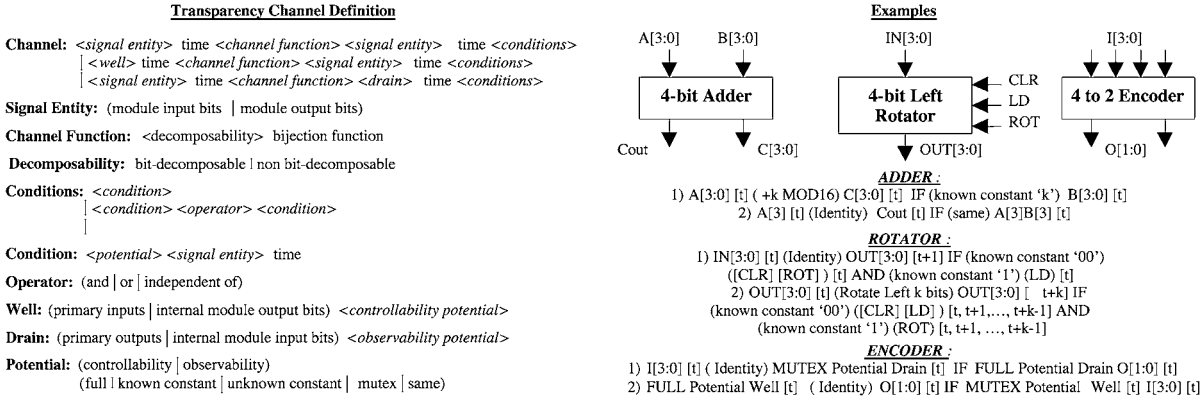


Fig. 4. Transparency channel definition and examples.

word level reachability paths, transparency channels facilitate construction of bit cluster level reachability paths. Given an 8-bit adder, for example, 8-bit transparency functions are typically defined and utilized by previous approaches to traverse through the adder. Transparency channels, however, are also defined on sub-word bit clusters, such as the four least significant bits, thus allowing construction of fine-grained paths.

3.2. RT-Level Hierarchical Test Generation

Transparency channels facilitate a fine-grained reachability path construction method that exploits extensively the traversal capability inherently available in a design. Transparent reachability paths support a powerful *divide-&-conquer* RT-Level hierarchical test generation methodology that results in significant test generation time reduction and highly efficient test for modular designs. A recursive design traversal algorithm is applied for each module in the design, effectively combining transparency channels in order to construct reachability paths. Variable bitwidth signal entities, loops and reconvergence are taken into account in order to prioritize the probing of transparency channels, thus accelerating algorithm convergence. Channels on the identified paths are further combined into templates, through which translation is rapidly performed. An overview of the corresponding hierarchical RT-Level test generation method is shown in Fig. 5.

The proposed method relies on the assumption that reachability paths exist inherently in the design. If this is not the case, a testability analysis methodology such as [14] may be employed in order to pinpoint necessary DFT modifications that will render a modularly transparent design. The identified reachability paths are subsequently used to translate locally generated vectors and responses for each module into globally applicable test. The proposed scheme is independent of the local test generation method and the fault model employed for each module. In addition, transparent reachability paths are identified regardless of the locally generated test vectors. As a result, local tests can be modified and enhanced in order to provide higher fault coverage, without invalidating the hierarchical test translation paths. The fault coverage attained by the local vectors is an upper bound to the fault coverage of the globally translated vectors, for each particular module. Therefore, local test generation should maximize the number of translatable patterns, possibly guided by global design knowledge, such as in [11, 21].

As depicted in Fig. 6, the produced information about the constructed reachability paths includes the module to be reached, the depth of the path in clock cycles, and the primary inputs and outputs participating on the main reachability path at each clock cycle. It also includes the condition inputs and corresponding values necessary for establishing the main path at each clock cycle, the modules that are traversed by the main path at each clock cycle and the corresponding transparency behavior expressed in terms of transparency channels. Such information is sufficient to support automated construction of templates for performing rapid local to global test translation. These templates apply the inverse of the bijective transparency functions on the translation path, providing a global vector that produces the desired local vector. Unused inputs in the global patterns are randomly filled for achieving high collateral fault coverage. Using these templates, test translation is symbolic and can be rapidly performed, supporting fast RT-Level hierarchical test generation.

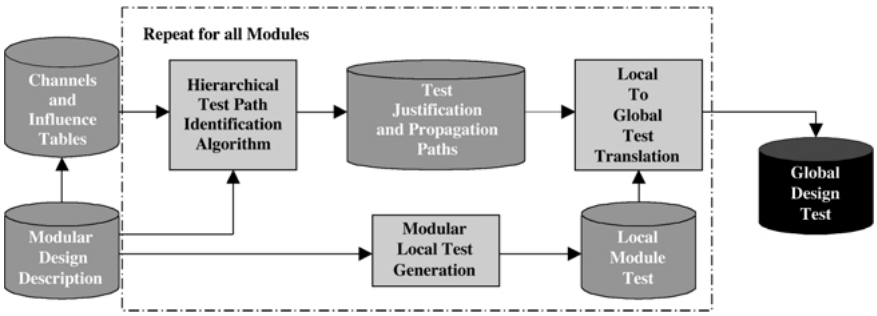


Fig. 5. RT-level hierarchical test generation.

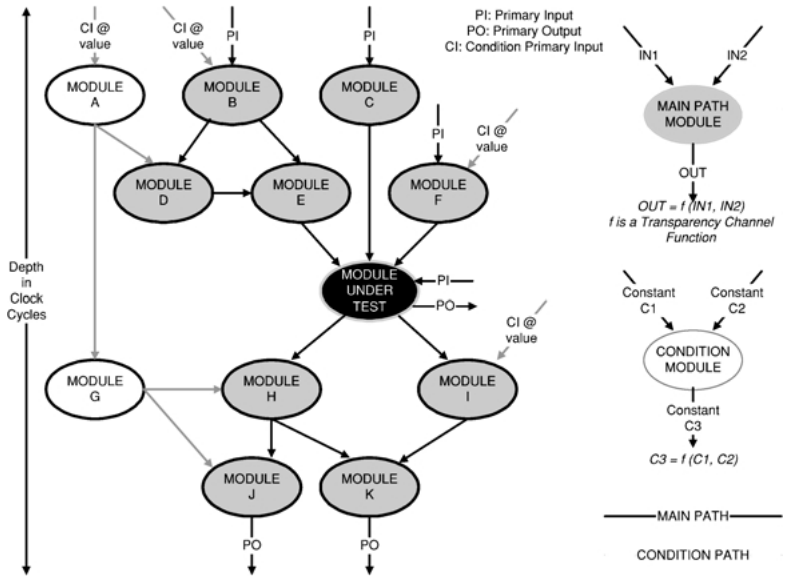


Fig. 6. Constructed reachability path information.

4. Controller

As explained in the motivation section, appropriate control state sequences are required for constructing reachability paths in controller-datapath circuits. The complexity of control state sequence identification, however, typically results in DFT solutions that isolate the datapath from the controller. Yet to reduce the overhead incurred by hierarchical test, inherent control behavior needs to be exploited before resorting to DFT to restore modular transparency.

A possible search strategy is to identify reachability paths on the datapath considering control signals as primary inputs and subsequently search in the controller for an appropriate state sequence producing the desired control signal values. Alternatively, the control signal values of each possible control state sequence may be considered as constraints during datapath reachability path construction. Such *trial-&-error* approaches are easy to implement; however, they are rather inefficient and result in long search times, due to the strict constraints imposed by the controller and the exponential number of control state sequences. A more informed approach may be an intertwined search, wherein each decision of the search algorithm on the datapath is checked immediately for compliance against the restrictions imposed by the controller. This scheme is expected to converge faster since backtracking is significantly reduced. Yet it is still an expensive exhaustive search and thus, hardly a viable alternative to DFT.

4.1. Influence Tables

In contrast to the above strategies, we propose a non-exhaustive method for searching in the combined controller-datapath space and thus rapidly identifying *potentially* appropriate control state sequences for module access. Such control state sequences are subsequently provided as constraints to the reachability path construction algorithm that verifies their suitability. The method is based on the concept of *influence tables* introduced herein.

Influence tables capture the interaction between datapath variables for each state of the controller. This information is subsequently combined across control states, to identify control state sequences that establish influence from primary inputs to the inputs of the MUT and from the outputs of the MUT to primary outputs. However, influence tables capture only topological but not functional interaction. Furthermore, while feedback variables from the datapath are considered when influence tables for sequences of control states are derived from influence tables of control states, the corresponding value is not examined. In addition, in order to avoid the possibly infinite number of control state sequences due to loops in the controller, influence

tables capture only distinct structural interaction across loop iterations, but do not capture distinct functional interaction. As a result, the identified control state sequences are only potential solutions and do not guarantee transparent module access. The latter is examined on the datapath, under the constraints imposed by the identified control state sequence.

The proposed method provides, in essence, a tradeoff between control state sequence identification time and effectiveness in reachability path construction. Although not all identified sequences will produce successful reachability paths, the method probes systematically only the most appropriate alternatives, pruning rapidly at the same time the ones that have no hope of success. Influence tables for each control state may be easily derived from an S-graph and a controller description of the design. Under the assumption that interaction between primary inputs, primary outputs, and registers is typically based on pairs and not on broadcasting, influence tables are generally sparse. Furthermore, the primary role of influence tables is to answer whether there is interaction between pairs of registers, inputs, and outputs. Therefore, influence tables can be viewed as directed graphs and are stored as *Hash Tables* of *Linked Lists*, where the head node of the list stores the hash key, which is the register that is influenced, and the rest of the nodes store the registers that are influencing the key register. Influence queries can thus be efficiently answered without explosion of storage requirements. To avoid excessive storage, influence tables for control state sequences may also be obtained on demand and need not be stored explicitly.

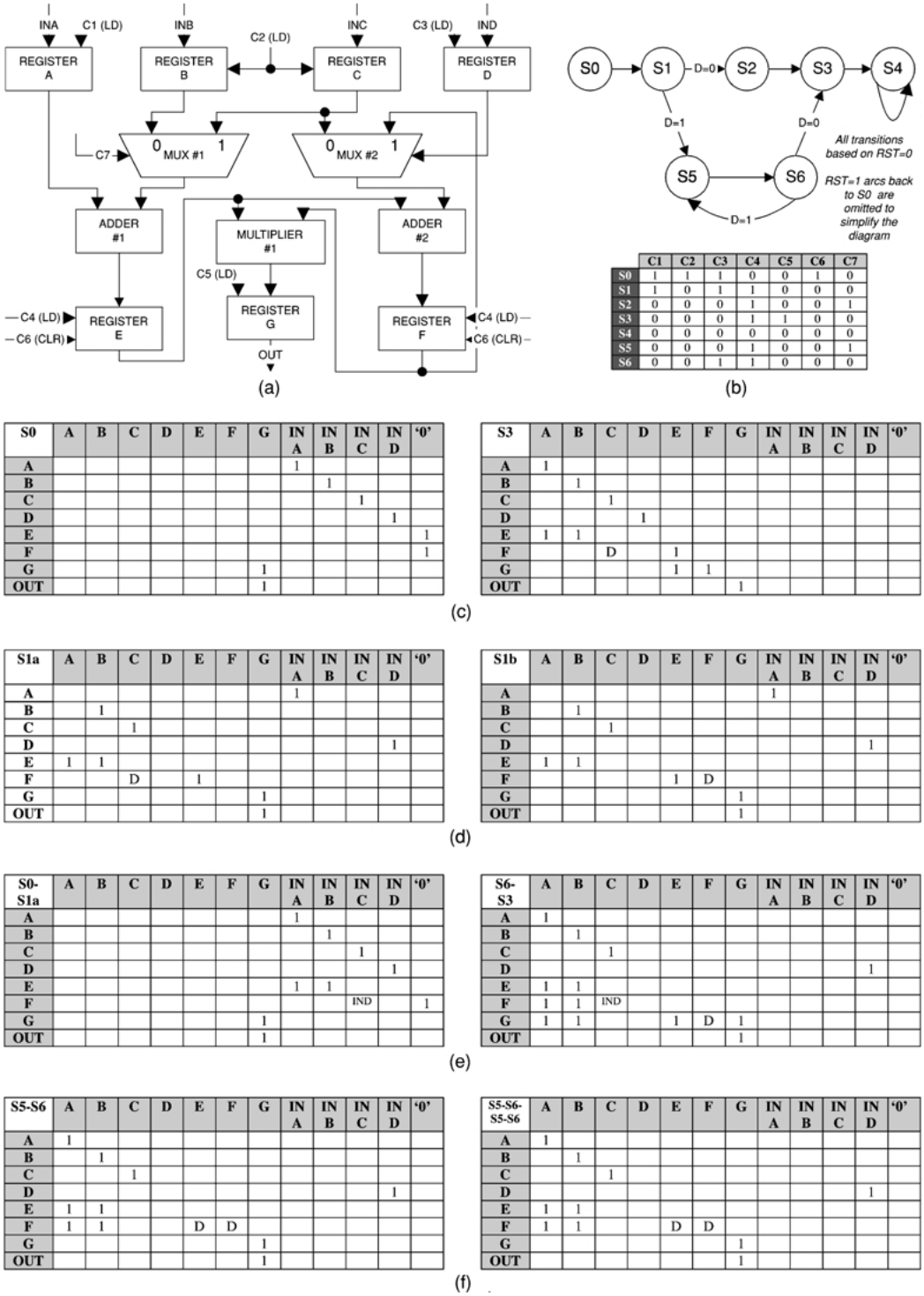
As the number of states in the controller FSM increases, the number of possible influence tables to search through grows significantly and data-dependent alternative control paths amplify this number. Yet the proposed scheme is much more efficient than exhaustive FSM analysis, since loops only result in a finite number of distinct influence tables and datapath feedback is treated in a symbolic fashion during control state sequence identification. Furthermore, since the length of the sequence corresponds to the clock cycles for applying each test pattern, an upper limit is imposed in practice, further reducing the number of alternatives. While influence tables do not perform an exhaustive controller-datapath interaction analysis, they provide a fast capability to exploit inherent control behavior appropriate for reachability paths, before resorting to expensive DFT. The key characteristics of influence tables are outlined below:

4.1.1. Influence Tables for Control States. The concept of influence tables is demonstrated through the controller-datapath pair example shown in Fig. 7. The datapath portion of the circuit is given in Fig. 7(a) and the controller is described in Fig. 7(b). The influence tables for states $S0$ and $S3$ are given in Fig. 7(c). The top row of the influence table contains the primary inputs, state registers and constant values that during this particular state may influence the primary outputs or state registers noted on the leftmost column. A $'1'$ in a table location indicates that the signal entity of the corresponding row is influenced during this particular state by the signal entity of the corresponding column. For example, in the influence table of state $S0$, register A is influenced by the primary input *INA*, since $LD = '1'$. Similarly, registers E and F are influenced by the constant value $'0'$, since $CLR = '1'$ and both register G and the output *OUT* are influenced by register G, since $LD = '0'$.

4.1.2. Conditional Influence. The influence table for state $S3$ demonstrates how conditional influence is captured by the proposed scheme. In state $S3$, register F of the example circuit is influenced through the *ADDER* ♯2 by register C, under a condition on register D. In this case, register D does not directly influence register F, but it does control the potential impact of register C on register F. Therefore, the entry in the corresponding table location is not simply a $'1'$ but rather a $'D'$, indicating the influence of register C on register F, based on a condition on register D.

4.1.3. Data-Dependent Alternative Influence. In Fig. 7(d), we demonstrate how influence tables handle situations where alternative sets of signal entities may influence the signal entity under examination, based upon the value of adjacent signal entities. In state $S1$, for example, register E and—depending on register D—either register C or register F will influence register F through *MUX* ♯2. In order to model these two alternatives, the influence table for state $S1$ is split into $S1a$ and $S1b$, each capturing a possible influence.

4.1.4. Influence Tables for Sequences of Control States. The effect of a control state sequence on the datapath is obtained by combining influence tables. The entry (M, N) is filled in the combined table if M at the beginning of the control sequence influences N at the end of the control state sequence. The combined influence tables for sequences $S0 - S1a$ and



	C1	C2	C3	C4	C5	C6	C7
S0	1	1	1	0	0	1	0
S1	1	0	1	1	0	0	0
S2	0	0	0	1	0	0	1
S3	0	0	0	1	1	0	0
S4	0	0	0	0	0	0	0
S5	0	0	0	1	0	0	1
S6	0	0	1	1	0	0	0

S0	A	B	C	D	E	F	G	IN A	IN B	IN C	IN D	'0'
A								1				
B									1			
C										1		
D											1	
E												1
F												1
G							1					
OUT							1					

S3	A	B	C	D	E	F	G	IN A	IN B	IN C	IN D	'0'
A	1											
B		1										
C			1									
D				1								
E	1	1										
F			D		1							
G						1	1					
OUT							1					

S1a	A	B	C	D	E	F	G	IN A	IN B	IN C	IN D	'0'
A								1				
B		1										
C			1									
D											1	
E	1	1										
F			D		1							
G							1					
OUT							1					

S1b	A	B	C	D	E	F	G	IN A	IN B	IN C	IN D	'0'
A								1				
B		1										
C			1									
D											1	
E	1	1										
F					1	D						
G							1					
OUT							1					

S0-S1a	A	B	C	D	E	F	G	IN A	IN B	IN C	IN D	'0'
A								1				
B									1			
C										1		
D											1	
E								1	1			
F										IND		1
G							1					
OUT							1					

S6-S3	A	B	C	D	E	F	G	IN A	IN B	IN C	IN D	'0'
A	1											
B		1										
C			1									
D											1	
E	1	1										
F	1	1	IND									
G	1	1			1	D	1					
OUT							1					

S5-S6	A	B	C	D	E	F	G	IN A	IN B	IN C	IN D	'0'
A	1											
B		1										
C			1									
D											1	
E	1	1										
F	1	1			D	D						
G							1					
OUT							1					

S5-S6-S5-S6	A	B	C	D	E	F	G	IN A	IN B	IN C	IN D	'0'
A	1											
B		1										
C			1									
D											1	
E	1	1										
F	1	1			D	D						
G							1					
OUT							1					

Fig. 7. Influence tables example: (a) Datapath example (b) Controller FSM (c) Influence tables for states $S0$ and $S3$ (d) Data-dependent alternative influence: Splitting state $S1$ into $S1a$ and $S1b$ (e) Influence tables for sequences of control states $S0 - S1a$ and $S6 - S3$ (f) Influence tables for controller loop between states $S5$ and $S6$.

$S6 - S3$ are demonstrated in Fig. 7(e). In the influence table of state $S0$, primary input *INA* influences register A which in turn influences register E in the influence table of state $S1a$. Therefore, in the combined influence table $S0 - S1a$, the primary input *INA* influences register E. Similarly, in the influence table of state $S0$ the primary input *INC* influences register C, which in turn influences itself and, depending on the value of register D, also influences register F in state $S1a$. Since register D is influenced by *IND* in state $S1a$, in the combined influence table for the state sequence $S0 - S1a$ the primary input *INC* influences register C and upon a condition on the primary input *IND*, it also influences register F. The influence table for the control state sequence $S6 - S3$ is derived similarly.

4.1.5. Controller Loops. In this scheme, loops do not impose the same complexity burden as in the combined controller-datapath search approaches. Since influence tables perform a structural and not a functional progress analysis of the search algorithm, loops are only allowed when the corresponding influence table provides a distinct impact pattern. For example, as shown in Fig. 7(f), the influence table for state sequence $S5 - S6$ is identical to the influence table of sequence $S5 - S6 - S5 - S6$. Since no additional structural influence exists when loop iterations are allowed, no additional influence tables are devised.

4.2. Control State Sequence Identification

Influence tables capture the impact of sequences of control states on datapath modules. This information is subsequently utilized in order to identify control state sequences that are potentially appropriate for establishing reachability paths for each datapath module. Initially, the structural requirements for testing each module are defined representing the primary inputs, registers and primary outputs that need to be controlled or observed in order to test a particular module. For example, testing *ADDER* ♯2 of Fig. 7(a) requires observing register F and controlling register E, register D, and one of registers F and C. The next step is to identify a state S_k in which test can be applied to the module. A state S_k is a valid candidate for this purpose, if the following two conditions hold:

1. There is a sequence of states with a corresponding influence table on which the registers that need to be controlled are only influenced by primary inputs and the sequence ends in a predecessor state of S_k. This denotes the *justification control state sequence* for the module.
2. There is a sequence of states with a corresponding influence table on which the registers that need to be observed influence at least one primary output and the sequence starts from a successor state of S_k. This denotes the *propagation control state sequence* for the module.

In the previous example, state $S2$ is a candidate control state for testing the *ADDER* ♯2, with $S0 - S1a$ being the justification and $S3 - S4$ being the propagation control state sequence. As shown in Fig. 8, during $S0 - S1a$, registers C, E and D are only influenced by primary inputs and during $S3 - S4$, register F influences the primary output. Since $S1a$ is a predecessor state of $S2$ and $S3$ is a successor state of $S2$, the control state sequence $S0 - S1a - S2 - S3 - S4$ is appropriate for testing the *ADDER* ♯2 module. The values required on feedback signals from the datapath to the controller also become requirements that need to be satisfied through the datapath reachability path construction algorithm. For example, state transition $S1a - S2$ requires a specific datapath feedback value on register D, for which a path is therefore constructed. Appropriate control state sequences for testing each module are similarly obtained.

Justification Control State Sequence

S0-S1a	A	B	C	D	E	F	G	IN A	IN B	IN C	IN D	'0'
A								1				
B									1			
C										1		
D											1	
E								1	1			
F										IND		1
G							1					
OUT							1					

Propagation Control State Sequence

S3-S4	A	B	C	D	E	F	G	IN A	IN B	IN C	IN D	'0'
A	1											
B		1										
C			1									
D				1								
E	1	1										
F			D		1							
G					1	1	1					
OUT					1	1	1					

Fig. 8. Control state sequence identification for *ADDER* #2.

4.3. Combined Reachability Path Identification

Influence table analysis provides control state sequences that are appropriate for establishing reachability paths for each module. Nevertheless, a design traversal algorithm is still required in order to identify the exact reachability paths on the datapath portion. Advance knowledge of the candidate control state sequences helps in reducing the number of alternative choices during reachability path composition, thus speeding up the search process.

The combined scheme for controller-datapath reachability path identification is depicted in Fig. 9. Initially, influence tables are derived from the controller-datapath description. For each datapath module, an appropriate control state sequence for testing this module is identified using these influence tables, as explained in the previous subsection. If no such sequence can be found, a testability bottleneck is reported in the controller and no solution exists unless DFT hardware is incorporated in the design. Otherwise, the identified control state sequence is provided in the form of *constraints* to the datapath traversal search algorithm. These constraints reduce substantially the backtracking of the search algorithm, effectively speeding up convergence. If the testability requirements of the module are not satisfied, a new control state sequence is requested from the controller analysis scheme and the process is repeated until either a reachability path is identified, or no more appropriate control state sequences can be found. In an effort to keep test application time low, control state sequences are examined in order of increasing length, so that the resulting paths incur the shortest delay possible.

5. Experimental Results

Experimental results supporting the proposed scheme are provided in this section. The applicability of transparent reachability paths in hierarchical test is investigated by comparing the proposed RT-Level hierarchical test generation method to full-circuit Gate-Level ATPG. The effectiveness of influence tables in accelerating reachability path construction is also examined in comparison to alternative non-DFT search approaches.

5.1. RT-Level Hierarchical Test Generation

The three-phase experimental setup of Fig. 10 is employed to demonstrate the efficiency of the proposed RT-Level hierarchical test generation scheme:

PHASE ♯1: The RT-Level description of the complete design is synthesized and full-circuit ATPG is applied on the Gate-Level view. Global test, along with the test generation time T_F, fault coverage C_F, and vector count V_F, is thus obtained.

PHASE ♯2: The proposed hierarchical test generation methodology is applied and the test translation paths for each module in the design are obtained in time T_P. The first module is synthesized and Gate-Level ATPG is applied on it, providing the local test vectors and the test generation time T_1, fault coverage C_1, and vector count V_1. These vectors are translated through the identified translation paths and fault-simulated on the complete circuit Gate-Level view for all design faults. The test translation time TR_1 and the fault simulation time F_1 are also noted. The process is repeated

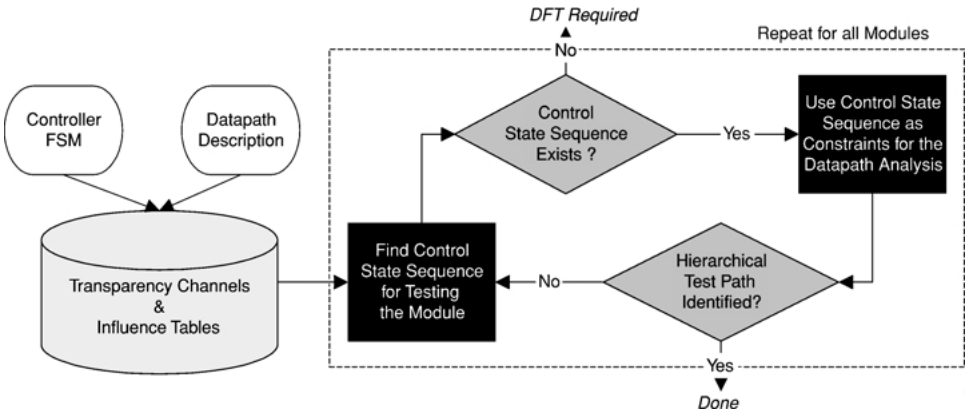


Fig. 9. Combined reachability path construction.

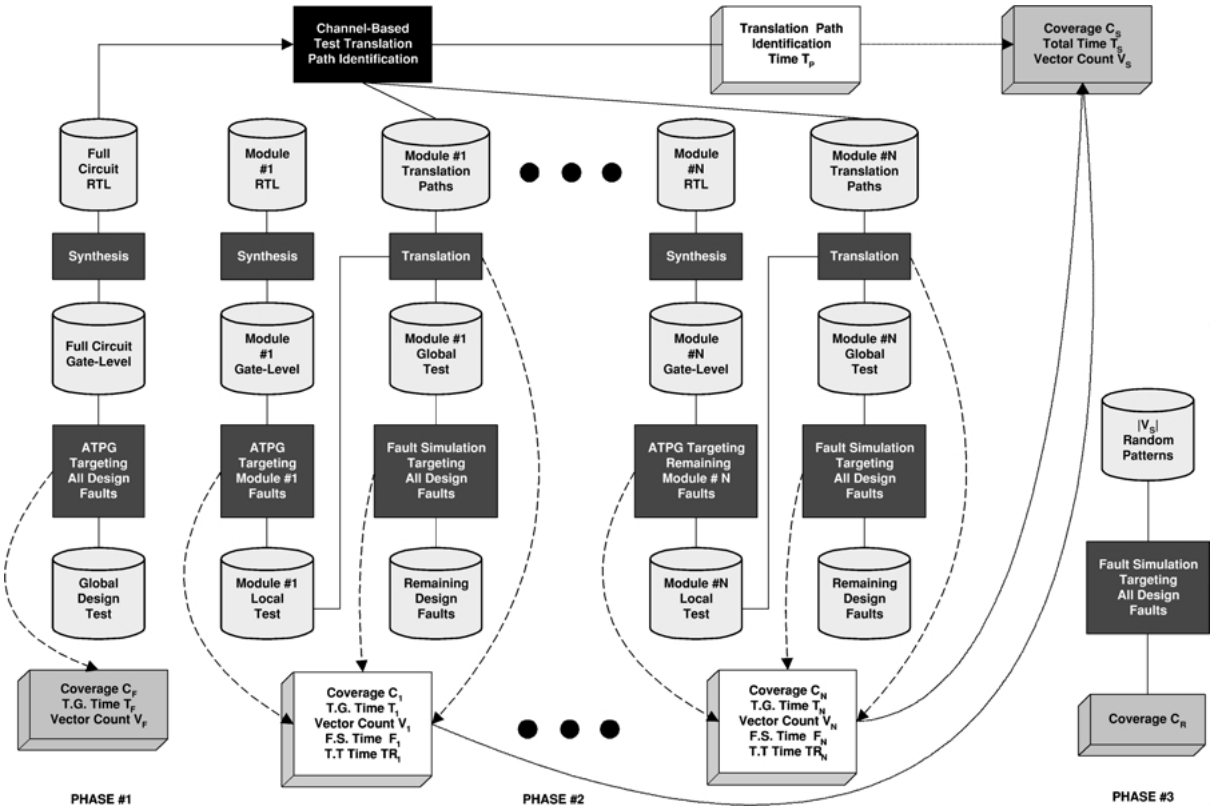


Fig. 10. Experimental setup for RT-level hierarchical test generation.

for each remaining module, targeting only faults that have not been covered by previous global vectors. The results are accumulated and the corresponding time T_S, fault coverage C_S and vector count V_S are obtained. The objective of this phase is to compare this methodology to full-circuit Gate-Level ATPG, based on test generation time, fault coverage and vector count.

PHASE ♯3: A number of V_S random patterns are fault-simulated on the original design and the corresponding coverage C_R is obtained. The objective of this phase is to demonstrate that the global patterns generated by the proposed methodology provide higher fault coverage than randomly generated patterns.

HITEC [18], PROOFS [17] and HOPE [10] are used for test generation, fault simulation, and random pattern test generation. A prototype tool called TRANSPARENT (TRANSlation Path Analysis RENdering Test) [12] was employed for identifying test translation paths and performing the actual translation. The experimental setup was applied on four benchmark designs modified so that they are modularly transparent. The first design is a 3-module circuit (MTC100) introduced in [19], with interesting feedback loop behavior. The second design is an 8-bit shift-&-add binary multiplier (MUL) described in [7]. It comprises 10 modules, including a FSM. The third circuit is an implementation of the commonly used greatest-common-divisor (GCD) benchmark circuit described in [6]. This circuit also comprises 10 modules, including a controller. The fourth circuit is a pipelined multiplier accumulator (MAC) for complex numbers and is described in [2]. It comprises 23 modules, including arithmetic blocks such as multipliers and add/subtract units, registers and a simple overflow detection module.

The test generation time, vector count, and fault coverage, along with the number of aborted, redundant, and total faults is provided for the full circuit ATPG in Table 1. In Table 2, we provide the time spent by TRANSPARENT on identifying the translation paths and performing the actual test translation, the total local test generation time and the total fault simulation time. The results are accumulated and the total test generation time, vector count and fault coverage are shown and highlighted for comparison purposes. In Table 3, the fault coverage obtained by fault simulating V_S

Table 1. Full-circuit gate-level ATPG results.

Full-circuit gate-level ATPG results	Number of modules	Test generation Time (sec) T_F	Vector count (vectors) V_F	Fault coverage (faults) C_F	Aborted number of faults	Redundant number of faults	Total number of faults
MTC100	3	3.383	146	1034	21	0	1055
MUL	10	13.580	191	760	64	10	834
GCD	10	14.560	431	865	37	71	973
MAC	23	21.300	627	27896	238	480	28614

Table 2. RT-level hierarchical test generation results.

RT-level hierarchical test generation results	Time $T_P + \sum TR_i$ (sec)	Time $\sum T_{N_i}$ (sec)	Time $\sum F_{N_i}$ (sec)	Test generation time (sec) T_S	Vector count (vectors) V_S	Fault coverage (faults) C_S
MTC100	0.300	0.117	0.020	0.437	178	1038
MUL	2.800	1.670	0.160	4.630	198	790
GCD	2.960	1.983	0.550	5.493	460	879
MAC	5.320	1.250	2.300	8.870	703	28145

Table 3. Random test generation results.

Random test generation results	Vector count V_S	Fault coverage C_R
MTC100	178	748
MUL	198	569
GCD	460	441
MAC	703	22454

Table 4. Total test generation CPU time comparison.

Test generation CPU time (sec)	Full circuit gate-level ATPG	RT-level hierarchical test generation
MTC100	3.383	0.440
MUL	13.580	4.630
GCD	14.560	5.493
MAC	21.300	8.870

Table 5. Total fault coverage comparison.

Fault coverage (faults)	Full circuit gate-level ATPG	Random test pattern generation	RT-level hierarchical test generation
MTC100	1034	748	1038
MUL	760	569	790
GCD	865	441	879
MAC	27896	22454	28145

Table 6. Total vector count comparison.

Vector count (vectors)	Full circuit gate-level ATPG	RT-level hierarchical test generation
MTC100	146	178
MUL	191	198
GCD	431	460
MAC	627	703

random patterns is provided. The total test generation time of the proposed RT-Level hierarchical test generation scheme and full circuit Gate-Level ATPG is shown in Table 4, demonstrating a speedup of almost an order of a magnitude in all four circuits. Furthermore, the speedup increases with the size of the circuit, implying scalability of the method to larger designs. In addition, Table 5 shows a slight fault coverage improvement in all four circuits, especially on the MAC, the larger benchmark. This improved fault coverage is attributed to the divide-&-conquer nature of hierarchical test. As the size of the circuit increases, the efficiency of ATPG on the complete design reduces and many testable faults are aborted due to the backtracking limit. Performing local test generation for each module allows ATPG to operate on smaller designs and, thus, generating vectors for many of these previously aborted faults and improving fault coverage. On the downside, an increase of vector count in the order of 10–15% is observed, as shown in Table 6, due to the divide-&-conquer approach. The

Table 7. Time comparison of reachability path identification search approaches.

Identification time (sec)	Multiplier control set *A*	Multiplier control set *B*	Multiplier control set *C*	Multiplier control set *D*	Adder
Datapath-first	0.58	4.25	0.47	N-T	N-T
Controller-first	17.97	150.00	17.52	N-T	N-T
Intertwined search	0.28	0.26	0.27	N-T	14.94
Proposed method	0.03	0.05	0.05	2.98	3.88

reason is that during local test generation each vector is only simulated for the faults of the particular module and not for the faults of the complete circuit. Once all local vectors are translated into global circuit test, performing test compaction could alleviate this problem.

5.2. *Path Construction Speed-Up*

The efficiency of the combined controller-datapath search scheme is demonstrated on two modules of the MUL circuit shown in Fig. 3, the MULTIPLIER and the ADDER. The search is expected to be successful on the MULTIPLIER module where reachability paths exist for each of the four sets of control values applied during normal functionality. However, the adder inputs are correlated and the search should report that no path exists. The following search approaches have been implemented and applied on these modules:

- *Datapath-First*: The search is performed on the datapath, and solutions are checked against the controller.
- *Controller-First*: Each valid control state sequence is imposed as constraints to the datapath search.
- *Intertwined Search*: Each decision of the datapath search algorithm is checked immediately against the restrictions imposed by the controller.
- *Proposed Scheme*: Influence Tables are used to derive potential control state sequences that are imposed as constraints to the datapath search.

The CPU time in seconds, spent by each of the four search approaches on a 266 MHz Pentium II with 64 MB of RAM is shown in Table 7. N/T entries in the table signify no termination within the imposed limit of 600 CPU seconds. The *Datapath-First* approach spends a long time backtracking due to solutions that are valid on the datapath but not supported through the controller. The *Controller-First* approach spends a long time examining control state sequences that do not establish datapath reachability paths. Although they are able to find solutions for the simple MULTIPLIER cases, they terminate neither for the Control Set *D* case, which has a solution, albeit a complicated one, nor for the ADDER, which actually lacks a solution. The *Intertwined Search* approach is similar to the *Datapath-First* approach only control compliance is checked at every clock cycle, thus reducing backtracking. Nevertheless, it spends a significant amount of time in handling the ADDER case and does not terminate on the complicated MULTIPLIER case. Finally, the proposed methodology identifies the expected solutions for all the MULTIPLIER cases and the lack of reachability paths for the ADDER in almost an order of magnitude less time than the best of the alternative approaches, verifying the power of the proposed scheme in accelerating reachability path construction.

6. Conclusion

Hierarchical test generation methodologies leverage on the ability to generate test at the boundary of each module but apply it from the primary I/Os of the design. In support of such approaches, reachability paths provide a symbolic local to global test translation capability based on modular transparency. Reachability path identification in controller-datapath circuits, however, requires exhaustive algorithms that result in excessive backtracking. Alternatively, costly DFT is employed for separating the controller from the datapath. To reduce this overhead, the proposed method performs a fast, non-exhaustive search capable of identifying a large number of inherent hierarchical test paths in such designs. Test is devised hierarchically for datapath modules, based on the notion of transparency. Additionally, an analysis of the controller through the introduced concept of influence tables produces appropriate control sequences for accessing each datapath module. The combination of these two schemes results in efficient RT-Level hierarchical test generation that provides significant speed-up, while preserving

comparable fault coverage and vector count to full-circuit Gate-Level ATPG.

References

1. 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.
2. P. Ashenden, *The Designer's Guide to VHDL*, San Matco, CA: Morgan-Kaufmann Publishers, 1996.
3. J.E. Carletta and C.A. Papachristou, "Behavioral Testability Insertion for Datapath/Controller Circuits," *Journal of Electronic Testing: Theory and Applications*, vol. 11, no. 1, pp. 9–28, 1997.
4. S. Dey, V. Gangaram, and M. Potkonjak, "A Controller Redesign Technique to Enhance Testability of Controller-Datapath Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 2, pp. 157–168, 1998.
5. 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.
6. I. Ghosh, A. Ragunathan, and N.K. Jha, "A Design for Testability Technique for RTL Circuits Using Control/Data Flow Extraction," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 8, pp. 706–723, 1998.
7. J.P. Hayes, *Computer Architecture and Organization*, 3rd edn., New York: McGraw-Hill, 1998.
8. F.F. Hsu, E.M. Rudnick, and J.H. Patel, "Enhancing High-Level Control-Flow for Improved Testability," in *International Conference on Computer-Aided Design*, 1996, pp. 322–328.
9. IEEE P1500 standard for embedded core test. Available from http://www.manta.ieee.org/groups/1500.
10. H.K. Lee and D.S. Ha, "HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, no. 9, pp. 1048–1058, 1996.
11. 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.
12. Y. Makris, J. Collins, A. Orailoglu, and P. Vishakantaiah, "TRANSPARENT: A System for RTL Testability Analysis, DFT Guidance and Hierarchical Test Generation," in *Custom Integrated Circuits Conference*, 1999, pp. 159–162.
13. 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.
14. Y. Makris and A. Orailoglu, "DFT Guidance Through RTL Test Justification and Propagation Analysis," in *International Test Conference*, 1998, pp. 668–677.
15. 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.
16. B.T. Murray and J.P. Hayes, "Test Propagation Through Modules and Circuits," in *International Test Conference*, 1991, pp. 748–757.
17. T. Niermann, W.T. Cheng, and J.H. Patel, "PROOFS: A Fast, Memory Efficient Sequential Circuit Fault Simulator," in *Design Automation Conference*, 1990, pp. 535–540.
18. T. Niermann and J.H. Patel, "HITEC: A Test Generation Package for Sequential Circuits," in *European Conference on Design Automation*, 1992, pp. 214–218.
19. P. Vishakantaiah, J.A. Abraham, and M.S. Abadir, "Automatic Test Knowledge Extraction from VHDL (ATKET)," in *Design Automation Conference*, 1992, pp. 273–278.
20. 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.
21. R.S. Tupuri and J.A. Abraham, "A Novel Test Generation Method for Processors Using Commercial ATPG," in *International Test Conference*, 1997, pp. 743–752.
22. Y. Zorian, E.J. Marinissen, and S. Dey, "Testing Embedded-Core Based System Chips," in *International Test Conference*, 1998, pp. 130–143.

Yiorgos Makris received the Diploma of Computer Engineering and Informatics from the University of Patras, Greece, in 1995, the M.S. degree in Computer Engineering from the University of California, San Diego, in 1997, and the Ph.D. degree in Computer Engineering from the University of California, San Diego, in 2001. He is currently an Assistant Professor of Electrical Engineering and Computer Science at Yale University. His research interests include design-for testability, test generation, testability analysis, concurrent test, and design diagnosis.

Jamison Collins received the B.S. degree in Computer Engineering from the University of California, San Diego, in 1999 and the M.S. degree in Computer Engineering from the University of California, San Diego, in 2001. He is currently a Ph.D. student at the University of California, San Diego. His research interests include test generation, simultaneous multithreading processors, aggressive speculation, and branch prediction.

Alex Orailoğlu received the S.B. degree from Harvard College, *cum laude*, in Applied Mathematics in 1977. He received the M.S. degree in Computer Science from the University of Illinois, Urbana, in 1979, and the Ph.D. degree in Computer Science from the University of Illinois, Urbana, in 1983. Prof. Orailoğlu has been a member of the faculty of the Computer Science and Engineering Department at the University of California, San Diego, since 1987. His research interests include the high-level synthesis of fault-tolerant microarchitectures, and the synthesis of testable designs.