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

Yiorgos Makris, Jamison Collins, Alex Orailoğlu
CSE Department – UC San Diego
{makris, jdcollin, alex}@cs.ucsd.edu

Abstract

We discuss a hierarchical test generation method for DFT-free controller-datapath pairs. A transparency-based scheme is devised for the datapath, wherein locally generated vectors are translated into global design test. The controller is examined through influence tables, used to generate valid control state sequences for testing each module through hierarchical test paths. Fault coverage levels and vector counts thus attained match closely those of traditional test generation methodologies, while sharply reducing the corresponding computational cost.

1. Introduction

Hierarchical test methodologies [8-10] have been gaining popularity due to their ability to manage the complexity of large designs in a *divide-&-conquer* fashion, as shown in Figure (1). Yet several challenges need to be addressed in order for such methodologies to attain efficiency and effectiveness.

Despite the ability to generate efficient local test for each module, the success of hierarchical test generation depends on the efficacy of the test translation process. Vector-by-vector test translation is complete but faces a complexity barrier. Bulk test translation using transparency behavior [1, 3, 7] is fast, albeit at the cost of sacrificing part of the translation capabilities of the design, as shown in Figure (2)(a). A wide yet compact transparency definition and a concise yet fast test translation method are therefore required.

In controller-datapath pairs, such as in Figure (2)(b), the impact of the controller on the datapath needs to be taken into account during test translation. Controllers, however, do not exhibit transparency, endangering the applicability of hierarchical test. In addition, exhaustive controller FSM analysis is expensive. Therefore, the controller-datapath interface is traditionally enhanced through DFT hardware [5] or controller redesign [3]. However, the cost of extra area and possible critical timing path complications incurred require that a comprehensive, yet non-exhaustive analysis of the controller-datapath interaction be performed, before resorting to expensive DFT.

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 [7]. In section 4, we introduce the concept of *influence tables*, a mechanism for exploring the impact of each controller state on the datapath. In section 5, influence tables are combined in order to derive appropriate control state sequences for testing each module. In section 6, 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 7.

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 problem is motivated in this section, based on the MUL example circuit shown in Figure (3), an 8-bit binary shift-&-add sign-magnitude multiplier described in [6].

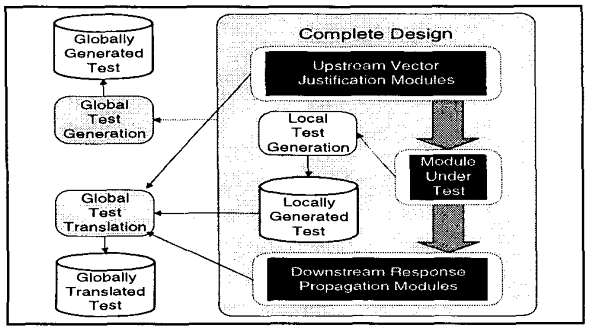


Figure (1): Hierarchical test generation

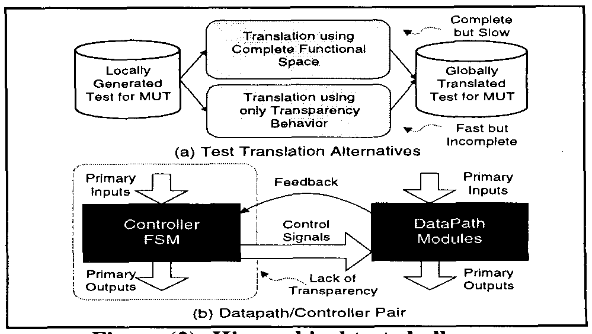
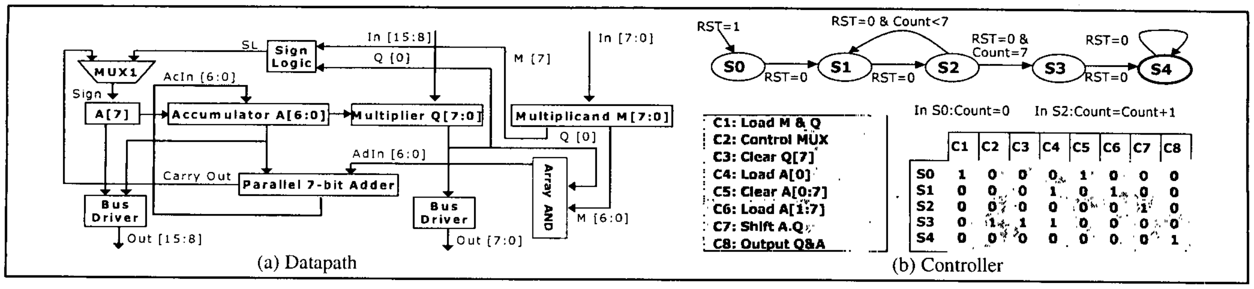

Locally Generated Test for MUT
Translation using Complete Functional Space
Translation using only Transparency Behavior
Globally Translated Test for MUT
Complete but Slow
Fast but Incomplete
(a) Test Translation Alternatives
Primary Inputs
Feedback
Primary Inputs
Controller FSM
Control Signals
DataPath Modules
Primary Outputs
Lack of Transparency
Primary Outputs
(b) Datapath/Controller Pair

Figure (2): Hierarchical test challenges

1081-7735/00 $10.00 © 2000 IEEE



	C1	C2	C3	C4	C5	C6	C7	C8
S0	1	0	0	0	1	0	0	0
S1	0	0	0	1	0	1	0	0
S2	0	0	0	0	0	0	1	0
S3	0	1	1	1	0	0	0	0
S4	0	0	0	0	0	0	0	1

(a) Datapath (b) Controller

Figure (3): Example circuit

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 among 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, will 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[7] and the ACCUMULATOR, will also result in excessive backtracking. Furthermore, word-level reasoning is insufficient to handle signals that split, such as the output M of the MULTIPLICAND.

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 on 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 because of loops such as the one between states *S1* and *S2* and variables such as *Count*. The computational difficulty of the problem has forced full reliance on DFT in previous attempts [3, 5] so as to isolate the datapath from the controller. Yet, we show in this paper that controller analysis, non-exhaustive though it may be, could provide useful results and prevent resorting to costly DFT.

2.3 Controller-datapath seamless test

When design constraints prohibit DFT at the controller-datapath interface, the following search alternatives exist:

(i) **Datapath-First:** The search is performed on the datapath, and solutions are checked against the controller.

(ii) **Controller-First:** The control signals of each valid control sequence are imposed as constraints to the datapath search.

(iii) **Intertwined Search:** Each decision of the datapath search algorithm is checked immediately against the restrictions imposed by the controller.

Even though 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 search space is pruned concurrently from both sides and convergence is faster. Implementation, however, requires a list of hopeful control state sequences to be kept and updated for each datapath decision

A fast yet efficient alternative to the aforementioned three approaches is introduced in the following sections. The proposed hierarchical test scheme avoids exhaustive search based on the concepts of *transparency channels* introduced in [7] and *influence tables* introduced herein.

3. Datapath

In this section, we describe various transparency definitions and we devise a hierarchical datapath test generation approach that addresses the complexity issues described above by employing transparency.

3.1 Transparency definitions

The concepts of *I-Path* and *T-path* for capturing module transparency, in terms of identity and transformation functions respectively were introduced in [1]. The path notion was extended in [4] through *onto* functions *(S-paths)* for providing stimuli to a module and *one-to-one* functions *(F-paths)* for propagating fault responses. A hierarchical test generation scheme and a test result propagation methodology, based on *ambiguity sets*, were introduced in [8]. Complexity issues limit the applicability of these methods to small datapath circuits. Test knowledge extraction for hierarchical designs is captured in terms of *modes* and further combined into test translation paths in [10]. Within the context of RTL testability analysis, the notion of *transparency channels* for capturing transparency was introduced in [7]. Transparency channels comprise a collective superset of the *I-Path, T-Path, S-Path* and *F-Path* [1, 4] concepts, expressing variable bitwidth transparency functions.

3.2 Channel-based hierarchical test generation

Transparency channels facilitate a powerful approach for hierarchical test generation. The method is independent of the actual local test generation method employed for each module. As a result, local tests can be modified and enhanced in order to provide higher fault coverage, without invalidating the translation paths. The overall efficiency of hierarchical test improves when local test generation is guided by global design knowledge [9], maximizing the number of translatable patterns.

A recursive design traversal algorithm is applied for each module in the design, employing transparency channels in order to identify test translation paths [7]. While traversing a module, available transparency channels are probed as to their suitability for constructing test justification or response propagation paths. Variable bitwidth signal entities, loops and reconvergence are considered in order to prioritize the probing of channels, accelerating algorithm convergence. Channels on the identified paths are combined into templates, through which translation is rapidly performed.

4. Controller

The objective of the proposed scheme is to examine the controller and identify control state sequences that are appropriate for testing each module in the design. In order to model the datapath behavior under the impact of the controller, we introduce the concept of *influence tables* that capture the structural interaction between datapath state variables, for each state of the controller. Influence tables are subsequently combined in order to identify control state sequences, thus establishing reachability paths from primary inputs to module inputs and from module outputs to primary outputs. To avoid expensive exhaustive controller reasoning, influence tables capture only the topological but not the functional interaction between primary inputs, primary outputs and state registers of the design. The control state sequences attained thus guarantee the existence of a reachability path but cannot reason on its test translation capabilities. Translation needs to be attained through a datapath hierarchical test path identification algorithm, under the guidance of the identified control state sequence; the relevant approach and algorithm are discussed in section 6.

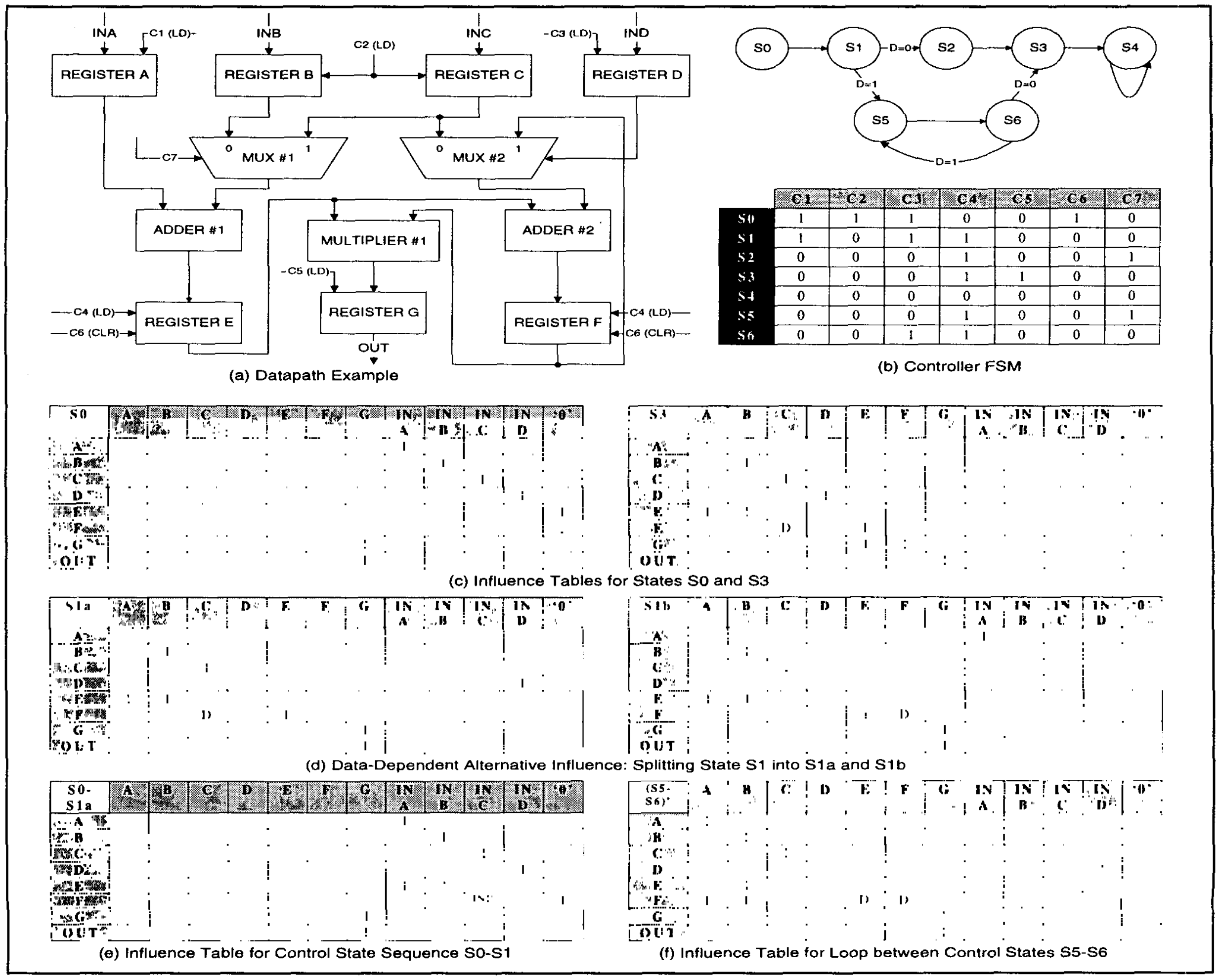
Influence Tables for Control States: The concept of *influence tables* is demonstrated through the controller-datapath pair example of Figure (4). The datapath is given in Figure (4)(a) and the controller is described in Figure (4)(b). The influence tables for states *S0* and *S3* are given in Figure (4)(c). The top row contains the primary inputs, state registers and relevant constant values that during this particular state may influence the primary outputs or state registers noted on the leftmost column. An entry 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'.

Conditional Influence: The influence table for state *S3* demonstrates how conditional influence is captured. 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*. The corresponding table location entry is therefore not a '1' but a variable name, D, indicating the source of the conditional influence.

Data-Dependent Alternative Influence: In Figure (4)(d) we demonstrate how the influence tables handle alternative influence of signals. 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 this mutually exclusive behavior, the influence table for state *S1* is split into *S1a* and *S1b*, each capturing one of the possible influences, as depicted in Figure (4)(d). This is required only when datapath registers such as *D* are used to decide among alternative influences. In the case of MUX #1, which is controlled by a controller signal, the resolution is automatically performed, since distinct values on *C7* indicate distinct control states.

Influence Tables for Sequences of Control States: The effect of a control state sequence on the datapath is obtained by combining the influence tables. An entry in the combined table is filled if the column register influences the row register through the control state sequence. The combined influence table for sequence *S0-S1a* is shown in Figure (4)(e). In the influence table of state *S0* the primary input *INA* influences register *A* which in turn influences register *E* in the influence table of state *S1a*. As a result, in the combined influence table *S0-S1a*, *INA* influences *E*. The rest of the table is filled similarly.

Controller Loops: Influence tables handle loops very efficiently, reducing the complexity of the proposed search method. Since only a structural analysis is performed, a finite number of tables modeling distinct iteration influence are required. Once the maximum possible register interaction through the loop is reached, the influence tables remain constant. For example, Figure (4)(f) shows the final influence table for the loop between states *S5-S6*, denoted $(S5\text{-}S6)^+$, which is achieved after one iteration.



(a) Datapath Example

	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

(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 Table for Control State Sequence S0-S1

(f) Influence Table for Loop between Control States S5-S6

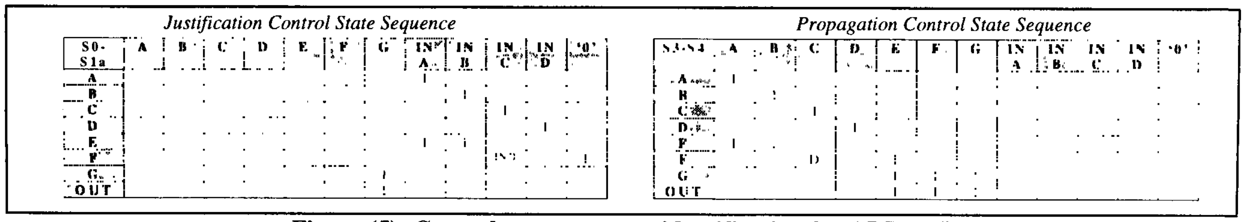
Figure (4): Influence table examples

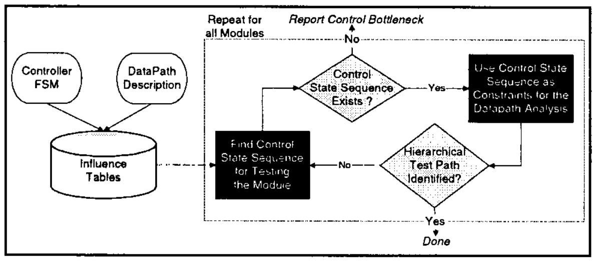
5. Control state sequence identification

Influence tables capture the impact of the controller states and sequences of states on the datapath. This information is subsequently utilized in order to identify control state sequences that are appropriate for testing each datapath module. Initially, the requirements for testing a module need to be defined. These requirements represent 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 Figure (4)(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 *Sk* at which the test may be applied to the module. A state *Sk* is a valid candidate, if the following two conditions hold:

- There exists a sequence of states ending in a predecessor state of *Sk* with an influence table in which the registers that need to be controlled are only influenced by primary inputs. This denotes the *justification control state sequence.*
- There exists a sequence of states starting from a successor state of *Sk* with an influence table in which the registers that need to be observed influence at least one primary output. This denotes the *propagation control state sequence.*

In the previous example, state *S2* qualifies for testing the ADDER #2, with *S0-S1a* being the justification and *S3-S4* being the propagation control state sequence. As shown in Figure (5), during *S0-S1a*, registers *C*, *E* and *D* are only affected 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 an appropriate candidate for testing the ADDER #2 module. Appropriate control state sequences for testing other modules in the design can be similarly obtained. These control state sequences guarantee the existence of a reachability path for the module under test and are further combined with the datapath search scheme that will evaluate the justification and propagation capabilities of the corresponding path, as explained in the next section.


Justification Control State Sequence
Propagation Control State Sequence
S0-S1a
A
B
C
D
E
F
G
OUT
Figure (5): Control state sequence identification for ADDER#2


Repeat for all Modules
Report Control Bottleneck
No
Controller FSM
DataPath Description
Control State Sequence Exists ?
Yes
Use Control State Sequence as Constraints for the Datapath Analysis
Influence Tables
Find Control State Sequence for Testing the Module
No
Hierarchical Test Path Identified?
Yes
Done
Figure (6): Combined controller-datapath search

6. Combined test path identification

The influence table analysis of section 4 provides valid control state sequences that are appropriate for establishing reachability paths from the primary inputs to the module under test and from the module under test to the primary outputs. A datapath hierarchical test path composition methodology is still required in order to identify the exact reachability paths and to evaluate local to global test translation capabilities. However, knowledge of the appropriate control state sequence in advance helps in guiding the datapath reachability analysis in a pruned search space. The control signals associated with these control state sequences are provided as *constraints* to the datapath search algorithm, reducing the number of alternative choices during hierarchical test path identification and thus speeding up the search process.

The combined scheme for controller-datapath hierarchical test path identification is depicted in Figure (6). Initially, the influence tables are derived from the controller-datapath description. Subsequently, a datapath module is selected and an appropriate control state sequence for testing this module is identified using the method of section 4. If no such sequence exists, a control testability bottleneck is reported; in this case no solution exists unless DFT hardware is incorporated in the design. Otherwise, the identified sequence is provided in the form of constraints to the datapath hierarchical test path identification algorithm outlined in [7]. These constraints reduce the backtracking of the search algorithm, effectively speeding up convergence. If the testability requirements of the module remain unmet, a new control state sequence is requested from the controller analysis scheme and the process repeated until either a hierarchical test path is identified, or no more appropriate control state sequences can be found. Examination of control state sequences in order of increasing length helps keep test application time low.

7. Experimental results

A 3-phase experimental setup is employed for validating the adequacy of transparency channels for test translation and the efficiency of the proposed hierarchical test generation scheme.

PHASE #1: The RTL description of the complete design is synthesized and ATPG is applied on the gate-level view, providing the global test vectors, the test generation time, the fault coverage and the vector count.

PHASE #2: Gate-level test generation is performed for each module. Subsequently, the proposed method for test translation path identification is applied and the local vectors are translated via the corresponding templates. The results are accumulated and the total time, fault coverage and vector count is obtained.

PHASE #3: A number of random patterns, equal to the number of global patterns generated in phase #2, are fault-simulated on the complete design and the corresponding coverage is obtained.

The experimental setup has been used on three benchmark designs, comprising simple RTL modules. The first design is a 3-module circuit MTC100 [10], the second design is the shift-&-add multiplier (MUL) shown in Figure (3), and the third circuit is a pipelined multiplier accumulator (MAC) for complex numbers [2]. The results for these circuits are shown in Tables (1)-(3).

Test Generation Time in (sec)	Gate-Level	Hierarchical
MTC100	3.38	0.44
MUL	13.58	4.63
MAC	21.30	8.87

Table (1): Total test generation time comparison

# of Faults Covered	Gate-Level	Hierarchical	Random
MTC100	1034	1038	748
MUL	760	790	569
MAC	27896	27245	22454

Table (2): Fault coverage comparison

# of Vectors Generated	Gate-Level	Hierarchical
MTC100	146	178
MUL	191	198
MAC	627	703

Table (3): Vector count comparison

Path Identification Time in (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 Scheme	0.03	0.05	0.05	2.98	3.88

Table (4): CPU time comparison for hierarchical test path identification approaches

In all three circuits, the channel-based hierarchical test generation scheme outperforms full circuit gate-level ATPG in terms of total test generation time. As shown in Table (1), in the first circuit the reduction is almost of an order of magnitude, while for the other two circuits it is approximately 65%. Table (2) shows that fault coverage slightly increases in the first two circuits, while in the third example a drop of around 2.5% ensues. In all cases, random vectors achieve significantly lower coverage. In terms of vector count, a slight increase of the order of 10-15% is observed, as shown in Table (3), due to the *divide-&-conquer* scheme. As demonstrated by these results, transparency channels facilitate a powerful hierarchical test generation scheme for datapath designs that significantly reduces test generation time while having minimal impact on fault coverage and vector count. Evidently, the proposed scheme constitutes a superior alternative to full circuit gate-level ATPG.

The efficiency of the combined controller-datapath search scheme is demonstrated on two modules of the MUL circuit shown in Figure (3). The three search approaches of section 2 and the proposed methodology of section 5 have been implemented and applied on these modules. The search is expected to be successful on the MULTIPLIER module where reachability paths exist for each of the four sets of control values that are applied to the MULTIPLIER during normal functionality. However, the adder inputs are correlated and the search should report that no path exists.

The CPU time[1] spent by each of the four search approaches on a 266 MHz Pentium^{TM} II with 64 MB of RAM is shown in Table (4). The *Datapath-first* and *Controller-first* search approaches spent a long time before converging. Although they were able to find solutions for the simple MULTIPLIER cases, they did not terminate for the *Control Set D* case which has a complicated solution and for the ADDER also as expected, since no solution exists. The intertwined controller-datapath exhaustive search performed better than the first two approaches due to reduced backtracking. Nevertheless, it spent a significant amount of time in handling the ADDER case and did not terminate on the complicated MULTIPLIER case. Finally, the proposed methodology identified the expected solutions for all the MULTIPLIER cases and the lack of reachability paths for the ADDER. The time spent is almost an order of magnitude less than the best of the alternative approaches, verifying the power of the proposed scheme in identifying hierarchical test paths.

[1]N/T entries signify no termination within 10 CPU minutes.

8. Conclusions

Reachability path identification in controller-datapath circuits requires exhaustive algorithms, resulting 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 channels*. Additionally, an analysis of the controller through the introduced concept of *influence tables* produces valid control sequences that are appropriate for accessing each datapath module. The combination of these two methods results in efficient hierarchical test generation that provides significant speed-up, while preserving fault coverage and vector count levels comparable to full-circuit gate-level ATPG.

9. References

[1] M. S. Abadir, 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*, Morgan-Kaufmann Publishers Inc., 1996.

[3] S. Dey, V. Gangaram, M. Potkonjak, "A Controller Redesign Technique to Enhance Testability of Controller-Data Path Circuits", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 17, no. 2, pp. 157-168, 1998.

[4] 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.

[5] I. Ghosh, A. Raghunathan, N. K. Jha, "A DFT 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.

[6] J.P. Hayes, *Computer Architecture and Organization*, McGraw-Hill, 3rd Edition, 1998.

[7] Y. Makris, A. Orailoğlu, "RTL Test Justification and Propagation Analysis for Modular Designs", *Journal of Electronic Testing: Theory & Applications*, Kluwer Academic Publishers, vol. 13, no. 2, pp. 105-120, 1998.

[8] B. T. Murray, J. P. Hayes, "Hierarchical Test Generation Using Precomputed Tests for Modules", *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 9, no. 6, pp. 594-603, 1990.

[9] R. S. Tupuri, A. Krishnamachary, J. A. Abraham, "Test Generation for Gigahertz Processors using an Automatic Functional Constraint Extractor", *Proceedings of the Design Automation Conference*, pp. 647-652, 1999.

[10] P. Vishakantaiah, J. A. Abraham, D. G. Saab, "CHEETA: Composition of Hierarchical Sequential Tests using ATKET", *Proceedings of the International Test Conference*, pp. 606-615, 1993.