JOURNAL OF ELECTRONIC TESTING: Theory and Applications 13, 105–120 (1999)

RTL Test Justification and Propagation Analysis for Modular Designs*

YIORGOS MAKRIS AND ALEX ORAILOĞLU
Reliable Systems Synthesis Lab, CSE Department MC-0114, UCSD, La Jolla, CA 92093
makris@cs.ucsd.edu
alex@cs.ucsd.edu

Received March 23, 1998; Revised June 25, 1998

Editor: N. Jha

Abstract. Modular decomposition and functional abstraction are commonly employed to accommodate the growing size and complexity of modern designs. In the test domain, a *divide-and-conquer* type of approach is utilized, wherein test is locally generated for each module and consequently translated to global design test. We present an RTL analysis methodology that identifies the test justification and propagation bottlenecks, facilitating a judicious DFT insertion process. We introduce two mechanisms for capturing, without reasoning on the complete functional space, data and control module behavior related to test translation. A traversal algorithm that identifies the test translation bottlenecks in the design is described. The algorithm is capable of handling cyclic behavior, reconvergence and variable bit-widths in an efficient manner. We demonstrate our scheme on representative examples, unveiling its potential of accurately identifying and consequently minimizing the reported controllability and observability bottlenecks of large, modular designs.

Keywords: RTL testability analysis, test justification, test propagation, modular design, DFT

1. Introduction

The latest silicon manufacturing technology improvements have facilitated an explosion in the size and complexity of modern designs. Consequently, extreme challenges are imposed on tools and methodologies employed in the design and test of complex, digital circuits. No panacea solution exists, capable of addressing these challenges in an efficient, universal fashion. Furthermore, test has emerged as the most expensive and threatening burden in a circuit design cycle, revealing the imperative need for test-related innovative solutions.

As a result, a vast number of techniques and approaches have been developed, both for enhancing the testability of a design through DFT modifications and for improving the test generation and application process [1–7]. The nontrivial task of deciding the exact test framework for each design is left to the test engineer. DFT related decisions have to be made as early in the design cycle as possible, yet without compromising their cost-effectiveness. In order to select judiciously among the wide variety of choices, a priori testability information of the design is required [8–10]. Acquiring such test knowledge for a design is the objective of testability analysis, but its applicability scope has been limited so far, by a number of challenges arising in modern designs.

The enormous size of modern circuits rules out test approaches that address the complete design as a monolithic entity. State of the art practices for large designs employ both modular decomposition and functional abstraction in order to alleviate test problems. Gate-level ATPG is applied at each module boundary for efficient local test generation, while higher description levels are utilized for the translation of local into global test. Consequently, any viable test analysis methodology

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

needs to take into consideration the modular nature of IC designs and emerging test approaches, in order to prove applicable and provide meaningful data.

Due to the strict requirements of a typical circuit, testability enhancements need to be introduced as early in the design cycle as possible. Additionally, cost-effective DFT decisions require a more global understanding of the design than a gate-level analysis may provide. The high-level information available at behavioral RTL descriptions addresses both circuit size problems and design cycle constraints. As a result, RTL is the most effective level for performing testability analysis and guiding DFT modifications.

Large datapaths, intricate control and complex sequential logic are common features of modern designs, imposing additional burdens on testability analysis. Any attempt to reason exhaustively upon the complete functional space in a value-by-value manner, using traditional behavior capturing mechanisms such as FSMs and BDDs, is unlikely to succeed. Consequently, a judicious way of pruning to a restricted set of test-related behavior is necessitated. Temporal reasoning and handling complex control logic remain essential parts of such test related behavior.

Section 2 motivates the research being outlined, presenting a test framework for large, modularly designed circuits that the analysis methodology targets. Previous work in the area is discussed and associated challenges pinpointed. Section 3 provides an overview of the proposed scheme. Sections 4 through 7 present details of the four components of the proposed scheme. These four sections discuss the identification of the test translation requirements and the test translation related behavior of each module, the traversal of the design for test translation requirement satisfaction and the minimization of the reported bottlenecks. Section 8 demonstrates the proposed methodology on example circuits, while Section 9 presents the experimental validation flow and outlines the obtained results.

Our research aims at providing an early testability assessment of large, modularly designed circuits, capable of reasoning on both combinational and sequential logic. Within our scheme, data and control path logic is handled in a uniform fashion, exploiting the two mechanisms for capturing test translation related behavior. The goal of the analysis is the identification of RTL testability bottlenecks associated with test justification and propagation. Since the ultimate objective of this work is to guide design testability enhancements in an informed manner, test translation bottlenecks are captured in the signal entity domain that resembles the structural nature of most DFT modifications.

2. Research Motivation

Size and complexity considerations impede the ability of test generation tools to handle large designs as single, monolithic entities. We describe a framework that facilitates an effective test methodology, commonly employed in real designs. Within this framework, we define the objectives of the proposed testability analysis methodology. We refer to previous research efforts in the area and discuss associated challenges.

2.1. Problem Definition

Our work targets large modular designs and addresses one module at a time. A common framework employed for testing such designs utilizes local test generation within each module and subsequent translation of the local test into test patterns and responses meaningful at chip pinouts. During test application, each module is tested individually, while the remaining modules are grouped into upstream test justification logic and downstream test propagation logic. Test is initiated at the primary inputs and terminated at the primary outputs. Feedback loops may result in overlaps between the module under test, the justification logic and the propagation logic. This test framework is depicted in Fig. 1.

While a gate-level test generation process at each module boundary proves efficient, translation of local into global test endangers and limits the applicability of the above framework. Several reasons can be quoted as the primary factors for the limited success of the test translation process. During local test generation, global design capabilities are not taken into account, thus leading to nontranslatable test. Furthermore, the

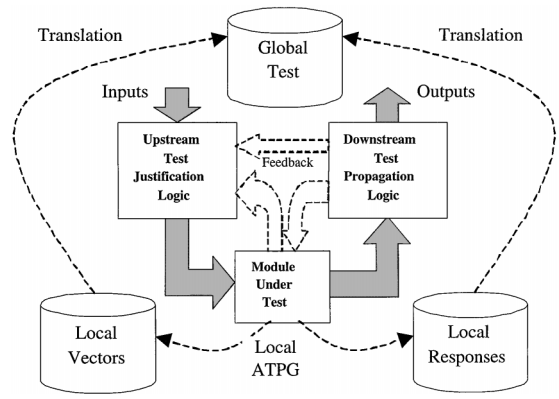


Fig. 1. Test framework.

potential overlap between the module under test, and the justification and propagation logic may degrade the locally generated test efficiency. The worst problem, however, is the complexity of the translation process itself, even in the absence of the above constraints. Exhaustive reasoning on the complete functional space of the design, so as to translate the local test in a vector-by-vector fashion, is of equivalent complexity to global design test generation. Therefore, mechanisms for identifying and utilizing only test translation related behavior for justifying and propagating test symbolically, are required.

Looking at the problem from a temporal point of view, the latest point for useful analysis in support of DFT is the RTL. As shown in Fig. 2, the analysis objective is to identify the bottlenecks that DFT modifications will be required to resolve. Strict time-to-market requirements and excessive complexity prohibit any reasoning at lower levels. Consequently, the analysis methodology needs to employ symbolic, bulk mode reasoning, instead of vector-by-vector translation, in order to pinpoint the test translation bottlenecks. Inevitably, due to the inability to reason exhaustively upon the design functionality, some locally generated tests will not be translated into global tests.

Any locally testable fault that remains uncovered after the translation of the local into global test can be classified as either of the following two types.

1. A fault for which the local test generator, due to lack of global design knowledge, selected a non-translatable test vector. Disabling the *random-fill* phase of a test generator can help moderate this effect. Furthermore, in order to eliminate this problem, a methodology was proposed in [6], wherein the global design capabilities are captured in terms of *constraints* and are examined during the local test generation. However, extracting these constraints requires reasoning on the complete functional space of the global design and is as difficult as to global test generation.
2. A fault that is untestable at the global design boundary. While the typical case for untestable faults is thought to be functional redundancies, we need to include herein faults that cannot be tested due to computational complexity limits encountered during test translation. In this case, DFT hardware is utilized to provide accessibility paths to the boundary of the modules for supporting the translation [3, 5, 7]. However, an indiscriminate provision of such accessibility paths to the boundary of each module is overly expensive. DFT modifications for enhancing the test translation process need to be guided by an analysis methodology that will pinpoint the test translation bottlenecks. Consequently, judicious and cost-effective DFT insertion will be facilitated for eliminating these bottlenecks.

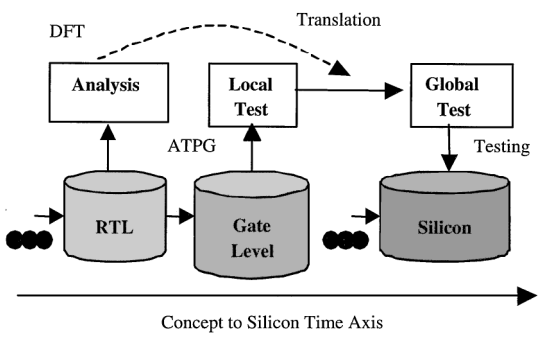


Fig. 2. Temporal view of analysis objective.

2.2. *Related Work*

Several researchers have identified the benefits of combining functional abstraction and modular decomposition, in order to assure improved testability of large designs. Among them, Murray and Hayes [11] proposed a test result propagation scheme through modules based on *ambiguity sets*. Vishakantaiah et al. [10] presented a test knowledge extraction methodology for hierarchical designs, wherein behavioral capabilities of the modules are extracted in terms of *modes*. Ghosh et al. [3] presented a similar DFT and test generation technique for core-based systems, based on the *control/data flow graph*. Hansen and Hayes [12] described a high-level test generation scheme for modular designs, utilizing a functional fault model of *physically induced faults* and *symbolic scheduling* to alleviate the large search space. Recently, Tupuri and Abraham [6] introduced a functional test generation method for embedded modules by incorporating the test justification and propagation capabilities of the surrounding logic into the test generation process in the form of *constraints*. Also, Chen et al. [1], Corno et al. [8] and Lee and Patel [9], suggested various general approaches for addressing behavioral and RTL testability analysis.

2.3. *Challenges*

Several challenges are associated with the task of identifying test translation bottlenecks. Relying on the

justification and propagation of the exact vectors and responses for identifying test translation bottlenecks is overly expensive. The complexity of doing the translation itself would make such an approach unpalatable. Furthermore, any changes in the actual test set, would invalidate the analysis results. Identification of a set of sufficient symbolic test justification and propagation requirements for each module, coupled with an analysis of the design as to its capability to satisfy these requirements, possibly offers a solution. An identification of independent cones of logic, as discussed in Section 4, is suggested as a heuristic approach in this direction.

Further challenges are associated with the local to global test translation. We discuss and demonstrate them in Fig. 3, on a small, complex circuit, originally presented in [10].

- Exhaustive examination of the complete functional space of the modules in the design can be highly expensive. Large datapaths (e.g., 16-bit adder), sequential logic (e.g., 2^{12} states of the 12-bit counter) and intermodule complex behavior (e.g., feedback from OUT to the adder) prohibit exhaustive functional reasoning during the translation. Therefore, the challenge is to identify a subset of the complete functional space that is tolerable in terms of complexity and that captures the most common test translation behavioral features. In the example of Fig. 3, utilizing the LD capability of the counter for justifying values to the rest of the circuit eliminates the need to reason upon the complex counter FSM. In Section 5, we describe two mechanisms for capturing test translation related behavior.
- Identification of the appropriate domain for examining the design is a second challenge. In our example, reasoning on the output of the adder can be performed either through arithmetic values or through bit-wise signal entities. Choosing between the value domain and the signal domain is a critical decision. The value domain allows exploitation of arithmetic properties while the signal domain supports complex control logic and variable bit-widths. Our scheme examines both attributes, in a structural manner that resembles the nature of most DFT techniques.
- Modular traversal does not necessitate the use of the complete word width. For example, while propagating the counter's test responses over the adder in Fig. 3, we only need to reason on the sub-word that comprises the lowest 12 bits. An efficient scheme requires handling of dynamic changes in the signal bit-width.
- Such dynamic variance impedes the a priori extraction of module transparency behavior. The connectivity model of the complete design and the test requirements need to be taken into account. For example, the fact that one of the inputs of the adder splits into a 4-bit and a 12-bit signal hints that we need to be able to extract *on-the-fly* transparency behavior for these signal bit-widths.
- Sequential logic, reconvergent paths and feedback loops generate further challenges. In Fig. 3, sequences of two vectors are necessary in order to test the register, creating a cycle. Moreover, the feedback line from the adder's OVF to the counter's CLR signal, along with the FSM behavior of the counter, compose a complicated sequential logic that is difficult to capture. Handling sequential logic and cyclic behavior is an inseparable part of a viable test translation analysis scheme.

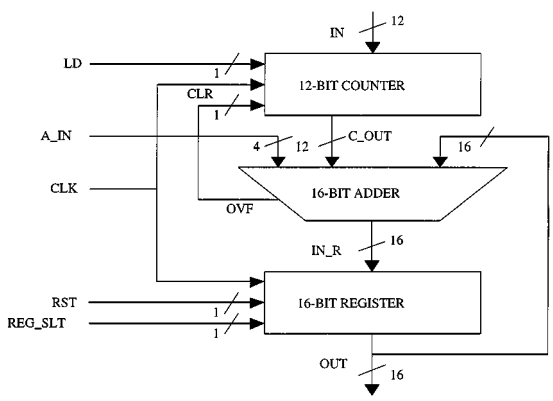


Fig. 3. Illustrative example for test translation challenges.

Any *divide-and-conquer* type of methodology, such as the module-by-module test translation analysis, imposes the challenge of combining and sharing the distributed results. In our case, the challenge is to minimize the reported bottlenecks by identifying the root causes that can be shared among multiple modules. The sequence in which the modules are examined and the local traversal decisions can have a crucial impact on the final results. As this minimization problem is computationally taxing, we propose two heuristics, in Section 7, that attempt to address this challenge.

3. Methodology Overview

As depicted in Fig. 4, our analysis methodology addresses the aforementioned challenges in four steps, discussed in the following sections. Initially, the test

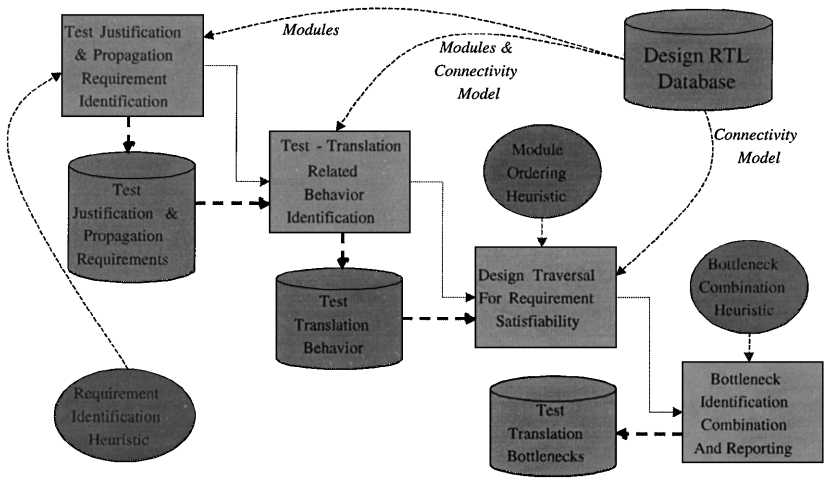


Fig. 4. Analysis methodology overview.

translation requirements are identified for each module in the design. Since actual test and responses are not available during analysis, we rely on a simple heuristic that attempts to ease the overkill of justifying and propagating any possible vector and response. Based on the cone-of-logic of each input and output, a set of justification and propagation requirements on signal entities are extracted, as explained in Section 4. Using these signal entities and the connectivity model of the complete design, we then identify the test translation related behavior of each module. This behavior is captured in one of two ways, described in Section 5.

Consequently, a traversal algorithm is applied to the boundary of each module. The algorithm attempts to satisfy the test justification and propagation requirements, using only the test translation related behavior of the modules in the design. The details of the traversal algorithm and the heuristics employed are further discussed in Section 6. Finally, in order to minimize the reported bottlenecks, a heuristic for combining bottlenecks between modules is applied as described in Section 7.

4. Test Requirement Identification

Since actual test vectors and responses are not known at the time of analysis, the most naive way is to require justification and propagation of every possible value and response to and from the boundary of each module. For sequential modules, reasoning upon sequences of vectors and responses needs to be considered. For a module of sequential depth k, we will require $k+1$ consecutive patterns and responses to fully test the module. Justification and propagation of every possible vector and response can be avoided based on a simple heuristic. We suggest in that context the identification of the cone of logic driven by each input and into each output.

This input/output mapping is an indication of the decomposability of the module, through which the dependent sets of inputs and outputs can be extracted and used for defining more realistic requirements. In either case, the requirements need to be expressed in a collective form and not in a value by value manner. Our heuristic captures these requirements in a stream-wise fashion, employing signal entities of variable bit-width. To demonstrate this, we examine the cones of logic in the 4-bit register of Fig. 5.

The register has sequential depth of 1. Consequently, two consecutive patterns are needed to test it. Since each output OUT[k] is driven only by two signals (CLR and IN[k]), the justification requirements are to get any

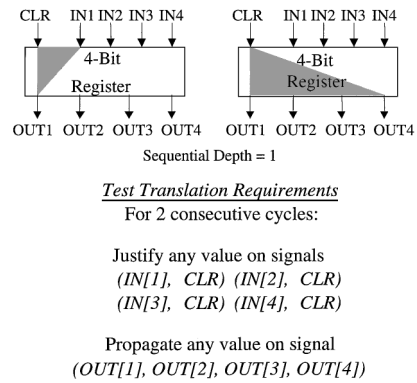


Fig. 5. Requirement identification heuristic.

two values on consecutive clock cycles on any 2-bit signal entity (CLK, IN[k]). This is a looser requirement than justifying any two values in consecutive clock cycles on the complete 5-bit input signal entity (CLR, IN[1], IN[2], IN[3], IN[4]). On the other hand, since the CLR signal drives all outputs, the propagation requirements are the translation in a distinguishable manner of any value at the complete 4-bit signal entity (OUT[1], OUT[2], OUT[3], OUT[4]).

5. Test Translation Related Behavior

Identification of the test translation requirements is followed by a capture of the test translation related behavior of each module. This behavior is utilized for examining the satisfiability of the requirements and identifying the bottlenecks. As discussed in Section 2, exhaustive examination of the complete functional space cannot be tolerated due to complexity issues. Therefore, we rely on capturing only test translation related behavior for each module and utilizing only this behavior for test justification and propagation. Although this scheme sacrifices some of the behavioral space, it provides the ability to reason on the translation capabilities of the design in an automated fashion.

We have developed two schemes for capturing test translation related behavior. The first scheme is based on arithmetic properties of modules and targets mainly datapath modules. The second scheme is an elaboration on the first scheme, based on channels (*bijection functions*) that can address uniformly data and control path modules and that can handle variable bit-width signal entities. These two schemes are introduced in this section.

5.1. Property-Based Scheme

The basic concept underlying our property-based scheme is the utilization of arithmetic properties that can provide a simple transparency mechanism over modules. Test requirements are translated in bulk mode, using the algebraic scheme that these arithmetic properties compose. The arithmetic properties do not capture the complete functional space of the modules but rather the types of behavior that will be most probably used for test translation.

Using this type of behavior, requirements at the input signal entities of a module are translated to equivalent

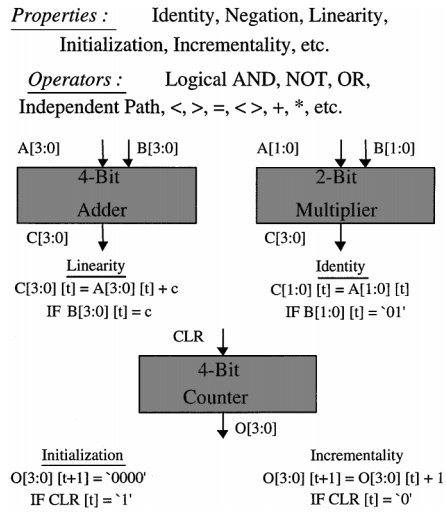


Fig. 6. Property-based scheme examples.

requirements at the output signal entities of the module and vice versa. The new requirements are possibly related through operators and are based on the satisfiability of a number of conditions, as demonstrated in Fig. 6. Only test translation related behavior of modules is considered for requirement translation. In a similar fashion, condition compliance examination is based on the same property combination scheme. The types of properties that our scheme considers include identity, linearity, negation, initialization and incrementality. The operators used for combining these properties are both in the arithmetic domain (e.g., $+$, *, $<$, $=$, $>$) and in the structural path domain (e.g., logical AND, NOT, OR). The arithmetic property scheme is capable of handling both combinational and sequential modules since timing is captured as part of the properties, as demonstrated on the 4-bit counter.

The arithmetic property scheme for capturing test translation related behavior of modules proves efficient for datapath designs, addressing combinational and sequential modules. It facilitates an algebraic scheme that not only can reason on whether a translation is feasible but also may guide the translation itself. However, a number of reasons limit its applicability. Control logic, usually, does not exhibit such arithmetic properties and therefore cannot be handled therein. Furthermore, reasoning on sub-word, variable bit entities is not simple through the arithmetic properties and relations between different width signal entities are not captured. Also, using this algebraic reasoning, bottlenecks are identified and reported in the value domain, while most DFT modifications are structural. Furthermore,

the interpretation between the two domains is not always trivial. The above problems have necessitated a more elaborate scheme, capable of addressing them efficiently.

5.2. Channel-Based Scheme

A behavioral formalism that addresses the above challenges is presented and demonstrated on simple example modules, followed by a discussion of its potential. The transparency aspects of the modules are captured in a structural manner through the notion of *channels*. Controllability and observability are captured through the notions of *well* and *drain*. This scheme handles variable bit-width signals through the *signal entity* notion and facilitates an efficient mechanism capable of handling uniformly, based on *conditions* and *operators*, combinational or sequential, data or control path modules.

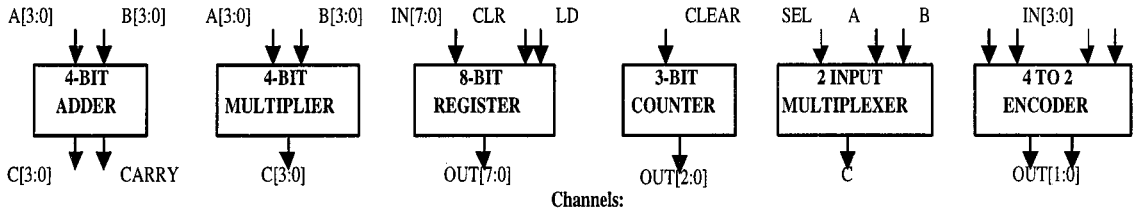
5.2.1. Behavioral Formalism. The components of the behavioral formalism are defined below.

Channel: A *channel* is defined to be a *mapping* between an input *signal entity* and an output *signal entity* or between a *well* and an output signal entity or between an input signal entity and a *drain*, based on the compliance of a set of zero or more *conditions*.
Mapping: A *mapping* is defined as a one-to-one and onto function from the set of possible values of an input *signal entity* or a *well*, to the set of possible values of an output *signal entity* or a *drain*.
Signal Entity: A *signal entity* represents a bundle of one or more signals at a certain point in time. Both the signals and the time point can be defined either statically or dynamically. Dynamic definition is a collective way of capturing many static definitions, allowing more flexibility and generality. As an example, a static definition of a signal entity might be *IN[2] at [t]* while a dynamic definition would look like *ONE OF IN[3], IN[2], IN[1] at any t where* $t > t_0$. The dynamic definition can be a choice from either a list of alternatives or from a closed type set definition.
Well: A *well* is the equivalent of a controllability point and captures the ability of a module to generate vectors on signal entities. It also captures the controllability of the primary inputs. Each well has associated with it a *potential* on signal entities.
Drain: A *drain* is the equivalent of an observability point and captures the ability of a module to evaluate vectors on signal entities. It also captures the observability of the primary outputs. Each drain has associated with it a *potential* on signal entities.
Potential: The *potential* of a well or a drain captures the type of vectors that can be generated or evaluated on a signal entity. For example, a full potential well of width k can generate the complete set of all possible 2^k vectors and a full potential drain of width k can evaluate likewise the complete set. The potential of a well or a drain is not defined in the value domain but rather in the signal domain. This implies that a well cannot generate a set of specific values unless a structural property defining the elements of the set in the signal domain (e.g., mutual exclusion, same, inverse) exists. These sets are captured in a stream-wise manner, avoiding the complete functional space complexity problem.
Conditions: Conditions are defined on one or more input signal entities and are combined through *operators*. In order for a channel to be activated, the conditions associated with it need to be satisfied.
Operators: Operators between signal entities are employed to create the above conditions. These operators can be either logical (e.g., AND, OR) stream-related (e.g., INDEPENDENT OF) or arithmetic (e.g., $=, \neq, \leq, \geq$). The only requirement is that they have to be expressed in a stream-wise fashion, since this is the rule for our behavior capturing mechanism. The latter point amplifies the fact that we cannot afford to perform directly a value-based analysis but instead we implicitly imitate it through a more generic, stream-wise reasoning.
Time-Extended Channels: A *time-extended channel of depth k* is a sequence of k channels instantiated in k consecutive clock cycles, with respect either to the input or the output signal entity.

5.2.2. Examples. A number of examples of basic RTL modules together with some of their channels are shown in Fig. 7, displaying how the behavioral formalism is used for capturing module traversal capabilities. These capabilities will be utilized by the design traversal methodology described in Section 6, in order to examine the satisfiability of the test justification and propagation requirements of each module.

5.2.3. Scheme Evaluation. The following points address the strengths of the behavioral formalism, summarizing its potential.

- The behavioral formalism that we introduced constitutes an abstraction scheme that decouples the



Channels:

ADDER

1) ONE OF {A[3:0], B[3:0]} at [t] MAPS TO C[3:0] at [t] IF CONSTANT POTENTIAL WELL at [t] MAPS TO ONE OF {B[3:0], A[3:0]} at [t]
2) FULL POTENTIAL WELL at [t] MAPS TO CARRY at [t] IF ALL SAME (A[3], B[3]) at [t]

MULTIPLIER

1) ONEOF {A[3:0], B[3:0]} at [t] MAPS TO ANY 4 CONSECUTIVE BITS OF C[6:0] at [t] IF ALL MUTEX (ONE OF {B[3:0], A[3:0]})

REGISTER

1) IN[n] at [t, t+k] MAPS TO OUT[n] at [t+1, t+1+k] IF CONSTANT '1' POTENTIAL WELL at [t, t+k] MAPS TO LD at [t, t+k]
AND CONSTANT `0' POTENTIAL WELL at [t, t+k] MAPS TO CLR at [t, t+k], where n=[0,7] (Time-extended channel of depth k)

COUNTER

1) FULL POTENTIAL WELL at [t] MAPS TO OUT [3:0] at [t, t+1] IF FULL POTENTIAL WELL at [t] MAPS TO OUT[3:0] at [t] AND CONSTANT `0' POTENTIAL WELL MAPS TO CLEAR at [t]

2) FULL POTENTIAL WELL at [t] MAPS TO OUT[1] at [t+1, t+2] IF FULL POTENTIAL WELL at [t] MAPS TO OUT[0] at [t] AND CONSTANT POTENTIAL WELL at [t] MAPS TO OUT[2] at [t] AND CONSTANT `0' POTENTIAL WELL MAPS TO CLEAR at [t]

(Time-extended channel of depth 2)

MULTIPLEXER

1) ONE OF {A,B} at [t] MAPS TO C at [t] IF FULL POTENTIAL WELL at [t] MAPS TO SEL at [t]

ENCODER

1) FULL POTENTIAL WELL at [t] MAPS TO C at [t] IF ALL SAME (A,B) at [t]
2) FULL POTENTIAL WELL at [t] MAPS TO OUT [1:0] [t] IF ALL MUTEX (IN[3:0]) at [t]

Fig. 7. Channel-based scheme examples.

module traversal capability extraction and the actual DFT modification process from testability analysis, providing a clean interfacing mechanism between them. Consequently, each of these tasks can be individually addressed.

- Although it does not cover the complete behavioral space of the modules, it handles variable bit-widths of input and output signal entities through the well and drain notions and applies not only at the full word size but also on sub-word signal entities. The ability of the channels to express any arbitrary bijection function brings our scheme as close to the concept of transparency as possible.
- There is more than one way to describe a traversal through channels. Extracting all the channels in advance is not practical. A similar problem limits the approaches described in [9, 10]. In order to address this issue, we rely on an *on-the-fly* channel selection for our test justification and propagation analysis, as explained further in the traversal algorithm.
- The time-extended channels provide a capability to reason on sequences of vectors and responses, thus facilitating sequential logic testability analysis. This proves valuable in case a clock disable signal is not provided for sequential modules.

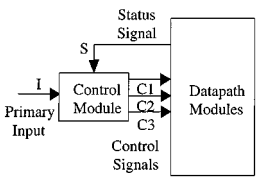
In short, the introduced formalism provides a concise and efficient way to capture behavior related to test justification and propagation, in a structural manner. In conjunction with the structural nature of DFT techniques, it facilitates a realistic testability analysis methodology that identifies the test translation bottlenecks as described in Section 6.

5.2.4. Handling Control Modules. Control logic, in general, does not exhibit the same type of transparency

behavior as datapath logic. Control modules interact with a number of datapath modules simultaneously and have a crucial impact on any type of path-based reasoning, such as the proposed testability analysis scheme. The FSM-like nature of control/data interaction is such that symbolic reasoning is not adequate and exact value analysis is required for capturing control capabilities and verifying path instantiation.

Fortunately, compared to the state space of datapath modules, the state space of control modules is much smaller, enabling precise reasoning. As demonstrated through research in the formal verification area [13, 14], abstracting away the datapath space of a design leads to the extraction of compact FSM representations of the control logic. These compact FSMs allow state reachability and edge traversability reasoning that captures the exact behavior of control logic.

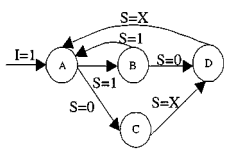
The behavioral formalism introduced in Section 5.2.1 is capable of capturing control logic capabilities either through the *channel* or through the *well/drain* notions. Referring to Fig. 8, a control module interacts with datapath modules through a set of control signals. These signals are generated based on previous state, primary inputs and possibly status signals going back from the datapath modules to the control module. If the control module behavior encompasses direct relations between primary inputs, status signals and control signals then the channel notion captures this transparency behavior, just as in datapath modules. Conditions generated on either status signals or primary inputs are again traced backwards to verify their compliance. In addition, in order to capture exact control signal sequences, a compact FSM is extracted, where the states are defined by the control signal combinations. The control module is subsequently described as a well with a constant potential for each state. For each path of transitions in the FSM, the control module is described as a well with a sequence of potentials defined across consecutive clock cycles. The *condition* mechanism allows for prior state verification. In the event that test responses need to be propagated over status signals, the control module behaves as a drain. The potential of the drain is the set of values or sequences of values on the status signal that take the FSM to distinguishable states. Examples of such control module behavior are given in Fig. 8.



Control/Data Interaction

I	S	State	Next
1	X	X	A
0	1	A	B
0	0	A	C
0	1	B	A
0	0	B	D
0	X	C	D
0	X	D	A

Control Logic State Table



Control Logic FSM

	C1	C2	C3
A	0	0	0
B	0	0	1
C	0	1	0
D	1	0	0

Control Signal Values

a) S at [t] MAPS TO C3 at [t+1] IF State at [t] = A
b) S at [t] MAPS TO FULL POTENTIAL DRAIN at [t] IF State is ONE OF {A,B}
c) CONSTANT `000' POTENTIAL WELL MAPS TO S[t] IF I at [t-1] MAPS TO CONSTANT `1' POTENTIAL WELL

Control Logic Transparency Behavior Examples

Fig. 8. Handling control modules.

This exact value FSM analysis, akin to traditional test generation, inherits all the traversal challenges such as reconvergence and cyclic behavior. Heuristic solutions for these problems are described in Section 6.3. The ability of the channel-based behavioral formalism to capture transparency behavior of both control and datapath modules in a uniform fashion is a major strength when addressing large designs. Furthermore, it facilitates efficient decisions during DFT insertion, as demonstrated in Section 8.

6. Test Translation Design Traversal

The channel-based formalism is a superset of the arithmetic property-based scheme, since each arithmetic property can be expressed as a set of channel bijection functions. Independent of the scheme utilized for capturing the test translation behavior of each module, a traversal algorithm that examines the satisfiability of the requirements is necessary. We introduce our algorithm based on the notion of channels and we clarify how our scheme handles design traversal challenges.

6.1. Algorithm

Our algorithmic scheme is based on requirement transformation through the notion of channels. Starting from the signal entities on which a requirement is defined, we utilize the global circuit connectivity model to identify the predecessor or successor modules. We then select a channel for this module that will transform the requirement into a number of requirements on signal entities of the module, related through operators and

```
satisfy (requirement) {
for each signal entity in requirement do{
   find relevant module in the design;
   repeat until no more available channels for module{
      select channel;
      if conditions comply satisfy (    new_requirement);
     }
   if not satisfied store best-match bottlenecks with module;
  }
}

main { repeat until no more modules {
        select _module;
        for each module {
           identify _requirements   (module_under_test);
           for each requirement {
              satisfy (requirement);
              report bottlenecks of modules on final path;
            }
          }
        }
         combine _bottlenecks;
     }
```

Fig. 9. Design traversal algorithm.

based on a number of conditions. If these conditions are satisfied then the process is repeated for the new requirements. Otherwise, a new channel is chosen. Each search path terminates when the requirements are satisfied through well or drain capabilities or when there are no more alternative channels to consider, in which case the nonsatisfiable signal entities are reported as bottlenecks. In Fig. 9 we provide a pseudocode form recursive description of the proposed algorithm.

The conditions that the channels generate when traversing modules are also expressed as requirements and the same algorithm is applied to examine their satisfiability. Since conditions are typically harder to satisfy, for backtracking minimization we examine them first. Further information regarding the algorithm shown can be found for the aspects depicted in boldface in Fig. 9. Thus, *requirement identification* is described in Section 4, *channel selection* between possible alternatives examined in the current section and *module selection* and *bottleneck combination* discussed in Section 7.

6.2. *Channel Selection Decision Factors*

Selecting judiciously among the alternative channels or combination of channels that satisfy a requirement over a module has a significant impact on the amount of backtracking performed. A greedy approach, *best match first*, is currently employed but only after a number of *decision factors* are examined. These factors are either *static* or *dynamic*. Static factors are extracted from the circuit connectivity model and include the number of signal entities and conditions involved in a channel and its latency in number of clock cycles. Dynamic factors capture the maximum potential that has been satisfied on each signal entity and the formation of loops or reconvergent paths in the search space.

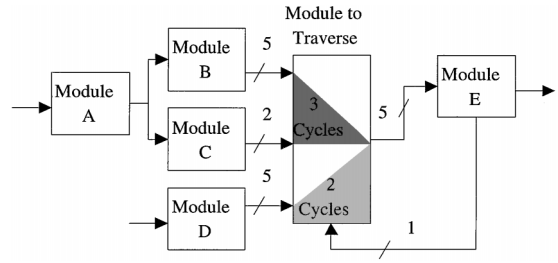


Fig. 10. Channel selection example.

In Fig. 10, an example is given where two alternative channels can justify the output of the module under test. The upper channel has a latency of three clock cycles, involves two signal entities of total width seven and creates one condition on a 2-bit signal entity. It further causes a reconvergent path in the search space when module A is reached. The lower channel has a latency of two clock cycles, involves two signal entities of total width six and creates one condition on a 1-bit signal entity. Also, a feedback loop from module E is created. In this case, both channels are rated as equally difficult. Hence, the decision would be made based on the dynamic factors capturing the maximum requirement previously satisfied on each of the two alternative channels, through modules B and D.

6.3. *Feedback Loops and Reconvergence*

Our scheme utilizes the cyclic behavior caused by feedback loops for requirement transformation over modules. Potential cycles are identified from the connectivity model and local analysis is performed. Channels that create loops are only selected when local analysis identifies a potential loop exit point, within an upper bound of iterations. Channels that create reconvergent paths are also identified from the connectivity model. To address this problem, multiple paths are followed in parallel and channel selection is postponed until the convergence point.

7. Bottleneck Combination

The proposed traversal algorithm identifies the controllability and observability bottlenecks for each module. Due to dynamic factors affecting local decisions during requirement satisfiability examination, the sequence in which the modules are considered has a critical impact on the final number of bottlenecks reported. Furthermore, the same bottleneck might solve translation problems for more than one module. Examining the modules in all possible sequences is computationally highly expensive. In order to address this problem, two heuristics are proposed, as outlined below.

The first heuristic gives the order in which the modules should be examined for justification and propagation requirement satisfiability. We assume that each signal entity in our design is an equiprobable bottleneck and we calculate the number of signal entities involved in examining each module. At each point, we examine the module that shares the maximum number of signal entities with the rest of the modules. Maximization of the number of signal entities with an associated dynamic factor is thus attained, after examining each module in the above sequence. Also, we expect that resolving the identified bottlenecks for a module will resolve possible bottlenecks for other modules, minimizing the total number of reported bottlenecks. The scheme is repeated until all the modules are examined. Individual orderings are derived for propagation and justification analysis.

The problem is easily formulated as a *find all paths* graph problem, where each module is a node and each signal entity a weighted edge according to the signal entity width. Primary inputs and primary outputs are also considered as nodes. In Fig. 11, a simple example is given, along with the consequent ordering. Justification analysis starts from module E, since it requires 12 signals, out of which eight are also required by other

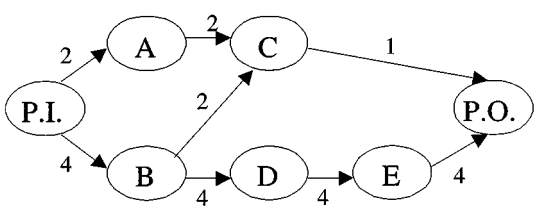


Fig. 11. Module ordering heuristic.

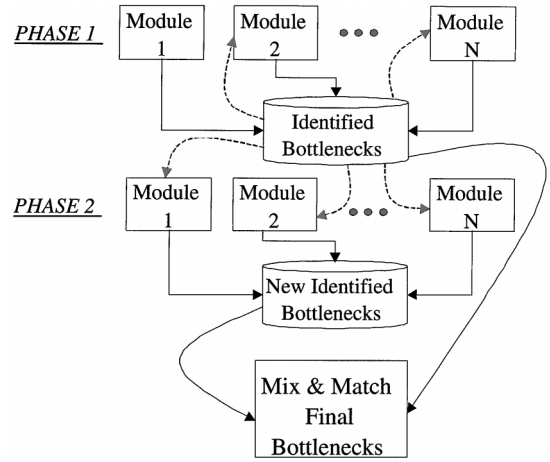


Fig. 12. Bottleneck combination heuristic.

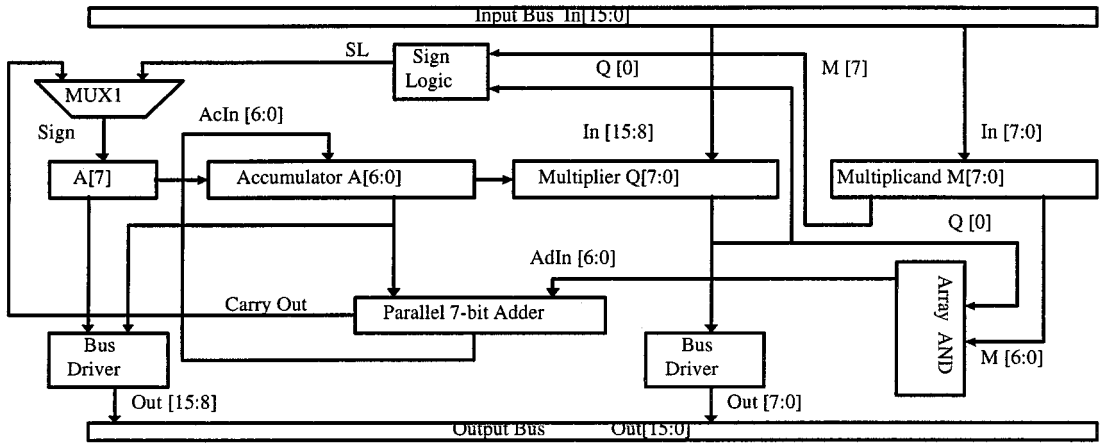
modules (B and D), the maximum number of shared signals. Similarly, propagation analysis starts from module B since it requires 15 signals, out of which nine are also required for other modules (A, C, D and E), again the maximum number of shared signals.

Since our first heuristic relies on a probabilistic assumption on potential bottlenecks, we also employ a second heuristic depicted in Fig. 12, which attempts to minimize the actual bottlenecks reported. During the first analysis phase of the heuristic, modules are examined in the order determined by the primary module ordering heuristic. Bottlenecks identified for a module k are considered resolved when we examine a subsequent module i with $i > k$. With all reported bottlenecks from the first phase considered resolved, we run a second analysis phase in the same module order and thus obtain a new set of bottlenecks. The bottlenecks from both phases are subsequently combined and a *mix & match* type of algorithm eliminates the redundant ones. Although this scheme does not ensure the maximum amount of bottleneck sharing, it has significantly decreased the number of reported bottlenecks in our experiments.

8. Examples

We demonstrate our testability analysis methodology on two example circuits. The first circuit is a sign-magnitude 8-bit binary multiplier described in [15]. The arithmetic property scheme has been applied on the circuit datapath portion. In Fig. 13, a circuit block diagram is depicted. The justification analysis at the

RTL BLOCK DIAGRAM OF BINARY SIGN-MAGNITUDE MULTIPLIER



JUSTIFICATION REQUIREMENT EXAMINATION EXAMPLE

(#1) (#2)

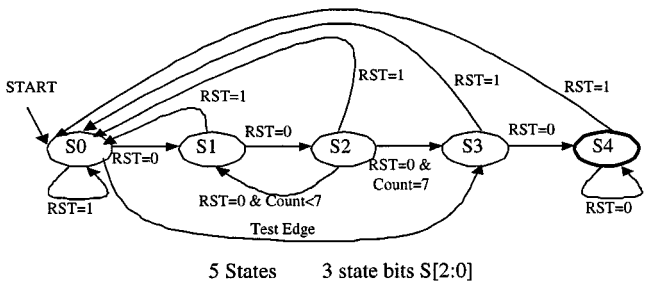
Objective: Justify All OUT[15:0] [t] => *(Threading)* => Justify All OUT[7:0] [t] **IND** Justify All OUT [15:8] [t]
(#1) => *(Bus Driver Identity)* => Justify All Q[7:0] [t-1] => *(Multiplier Identity)* => Justify All IN [15:8] [t-2] =>
=> *(Inbus Controllability)* => Satisfied (#3) (#4)
(#2) => *(Bus Driver Identity and Threading)* => Justify All A[7] [t-1] **IND** Justify All A[6:0] [t-1]
(#3) => *(Accumulator Sign Identity)* => Justify All Sign [t-2] => *(MUX #1 Identity)* Justify All SL [t-2] => *(Sign Logic Transparency)* => Justify All M[7] [t-2] => *(Multiplicand Identity and Threading)* => Justify All IN[7] [t-3]
(#4) => *(Accumulator Identity)* => Justify All AcIn[6:0] [t-2] => *(Adder Identity)* => Justify All AdIn[6:0] [t-2] => *(MUX #2 Identity)* => Justify All M[6:0] [t-2] => *(Multiplicand Identity and Threading)* => Justify All IN[6:0] [t-3]
Both (#3) and (#4) satisfied through *Inbus Controllability*
No conflicts on conditions generated on control signals

BOTTLENECKS WITH CONTROLLABLE CONTROL SIGNALS

CONTROLLABILITY: *AcIn[6:0] at [t,t+1], Shift Input to Multiplier at [t,t+1]*
OBSERVABILITY: *None*

HANDLING CONTROL LOGIC

Control FSM



5 States 3 state bits S[2:0]

ADDITIONAL CONTROLLABILITY BOTTLENECKS

S[0] at [t, t+1, t+2]

TRADE-OFF BOTTLENECK: S[1] (S1-S2 Loop Bypass)

Control Signals

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

C1: Load M & Q
C2: Control MUX
C3: Clear Q[7]
C4: Load A[0]
C5: Clear A[0:7]
C6: Load A[1:7]
C7: Shift A.Q
C8: Output Q&A

Fig. 13. Property-based analysis example on a binary sign-magnitude multiplier.

output bus is provided. The propagation analysis is similar. The circuit is highly transparent and no major bottlenecks have been identified. The few minor bottlenecks are also reported in the figure. As a result, we expect the test translation process to be very efficient for this circuit.

The control logic handling mechanism of Section 5.2.4 is applied on the multiplier controller, as shown also in Fig. 13. The normal operation FSM provides eight control signals to the datapath and has five states, requiring three state elements for encoding. When the above analysis is applied concurrently to both the datapath and the controller, taking into account the FSM behavior, an additional controllability bottleneck is identified on one of the state-encoding signals. The reason for this is that false paths may need to be utilized during the test translation, requiring illegal states or transitions in the controller. Furthermore, a trade-off bottleneck on a second state-encoding signal has been identified. More specifically, the ability to bypass the controller loop S1-S2-S1 decreases the test time for each vector by 14 clock cycles, at the additional cost of one more controllability bottleneck to be resolved. Such trade-off data can facilitate cost-effective DFT modifications.

The second circuit is a pipelined multiplier accumulator described in behavioral and RTL VHDL in [16]. The circuit is mainly datapath oriented and comprises combinational and sequential modules, feedback loops, reconvergent paths and variable bit-widths and a small control-like logic for the overflow calculation. In Fig. 14, we show the results of our channel-based scheme applied on this circuit. Justification analysis for the register REG22#1 and propagation analysis for the multiplier MUL#4 are provided in detail. The results of our module ordering heuristic for justification and propagation are shown and the complete set of testability bottlenecks identified through our methodology reported.

9. Experimental Validation

The proposed methodology for test justification and propagation analysis identifies the potential controllability and observability bottlenecks in the RTL design. This section describes the experimental validation framework employed for examining the analysis accuracy. Furthermore, results on the above example circuits are presented.

9.1. Validation Flow

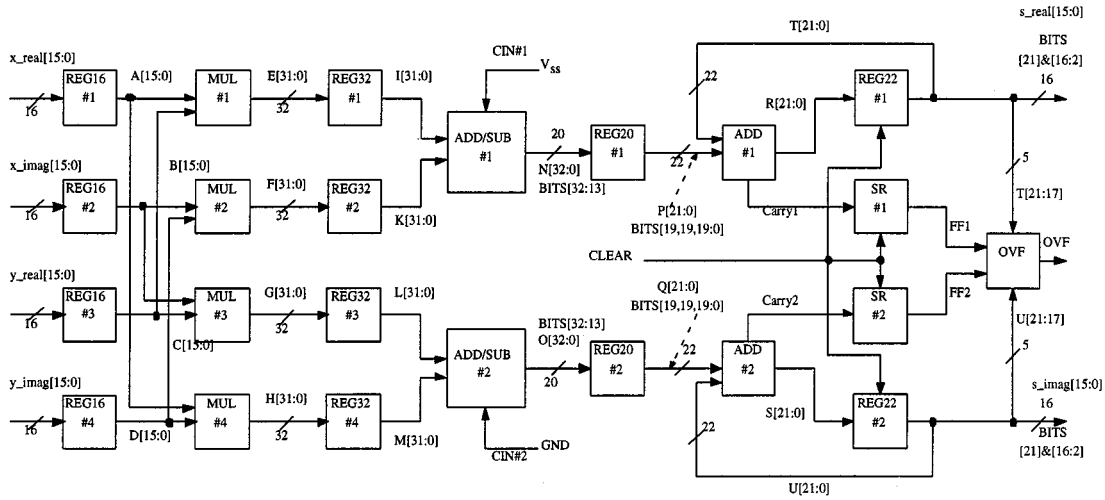
An overview of the validation flow is provided in Fig. 15. In compliance with the test framework for which the analysis methodology has been derived, our primary validation mechanism utilizes HITEC [17], a gate-level ATPG tool and is based on fault coverage comparison acquired from the fault simulator PROOFS [18]. Starting with an RTL description of the circuit, the described analysis methodology is applied, resulting in a list of controllability and observability bottlenecks. A gate-level model is further obtained through synthesis, on which the ATPG experiments are performed.

First, the ATPG tool is applied independently for each design module, resulting in local tests. Each local test is subsequently translated to the boundaries of the complete circuit and global test is obtained and fault simulated to provide the associated global fault coverage (GFC). The same experiment is then performed on an enhanced version of the design, wherein all the justification and propagation bottlenecks identified by the analysis are considered to be fully controllable or observable. The modified GFC, shown in the final column of the table of Fig. 15, is thus obtained. The underlying translation of the local to global test is manually performed.

9.2. Results

We applied the above experimental flow on TC100, the circuit of Fig. 3, the binary multiplier with and without the controller, and the pipelined multiplier accumulator described in the previous section. The analysis methodology indicated that TC100 and the binary multiplier without the controller are highly transparent and that no major test translation bottlenecks exist in the design. Consequently, we expect translation of local into global test to be highly successful. Summing up the total faults covered by the local tests and dividing by the local number of faults, we obtained 96.68% and 97.45% coverage, respectively. After the local test was translated to design boundary test, coverage dropped slightly to 92.80% and 93.27%, due to the few minor bottlenecks. A new translation after bottleneck resolution through test point insertion increased the coverage to 95.74% and 96.65%, indicating that our methodology identified accurately the bottlenecks of the test translation process. When the

RTL BLOCK DIAGRAM OF PIPELINED MULTIPLIER ACCUMULATOR (MAC)



JUSTIFICATION REQUIREMENT EXAMINATION EXAMPLE: REG22#1

Requirement: JUSTIFY (R[k], CLEAR), k=[0Q] at [t, t+1]=> *THROUGH PRIMARY INPUT WELL CAPABILITIES AND ADD#1 CHANNEL (P[k] at [t] MAPS TO R[k] at [t] IF T[21:0]=constant)* =>JUSTIFY P[k] at [t, t+1] **(justification of condition verifies its compliance through clear='1' at[t-1] over feedback loop)** => *THROUGH REG20#1 CHANNEL (N[j] at [t] MAPS TO P[t] at [t+1], j=[32K])* =>JUSTIFY N[j] at [t-1,t]=> *THROUGH ADD/SUB#1 CHANNEL (I[j-1] at [t] MAPS TO N[j] at [t] IF SAME (I[j-1], K[j-1]) at [t])* =>JUSTIFY SAME (I[j-1], K[j-1]) at [t-1, t]=> *THROUGH REG32#1,2 CHANNELS (E[j-1] at [t] MAPS TO I[j-1] at [t+1], F[j-1] at [t] MAPS TO K[j-1] at [t+1])* =>JUSTIFY SAME (E[j-1], F[j-1]) at [t-2, t-1]=> *THROUGH MUL#1,2 CHANNELS (A[m] at [t] MAPS TO at E[n] at [t], m=[15(], n=[31, 28(), B[m] at [t] MAPS TO F[n] at [t])* =>JUSTIFY SAME (A[m], B[m]) at [t-2, t-1] **(Controllability bottlenecks E[30:29], F[30:29])** => *THROUGH REG16#1,2 CHANNELS (x_real[m] at [t] MAPS TO A[m] at [t+1], x_imag[m] at [t] MAPS TO B[m] at [t+1])* =>JUSTIFY SAME (x_real[m], x_imag[m]) at [t-3, t-2]=> *THROUGH PRIMARY INPUT WELL CAPABILITIES* **=>Satisfied**

PROPAGATION REQUIREMENT EXAMINATION EXAMPLE: MUL#4

Requirement: PROPAGATE H[31:0] at [t]=> *THROUGH REG32#4 CHANNEL (H[31:0] at [t] MAPS TO M[31:0] at [t+1])* =>PROPAGATE M[31:0] at [t+1]=> *THROUGH ADD/SUB#2 CHANNEL (M[31:0] at [t] MAPS TO O[31:0] at [t] IF CONSTANT POTENTIAL L[31:0] at [t])* =>JUSTIFY CONSTANT POTENTIAL L[31:0] at [t+1] **(justification of condition verifies its compliance)** AND PROPAGATE O[31:0] at [t+1]=> *THROUGH REG#2 CHANNEL (O[32:13] at [t] MAPS TO Q[19:0] at [t+1])* **=> (Observability bottleneck on O[12:0])** PROPAGATE Q[19:0] at [t+2]=> *THROUGH ADD#2 CHANNEL (Q[21:0] at [t] MAPS TO S[21:0] at [t] IF CONSTANT POTENTIAL U[21:0] at [t])* =>JUSTIFY CONSTANT POTENTIAL U[21:0] at [t+2] **(justification of condition verifies its compliance)** AND PROPAGATE S[21:0] at [t+2]=> *THROUGH REG22#2 CHANNEL (S[21:0] at [t] MAPS TO U[21:0] at [t])* =>PROPAGATE U[21:0] at [t+3]=> *THROUGH DRAIN CAPABILITITIES OF PRIMARY OUTPUTS AND OVF BLOCK* **=>Satisfied (Observability bottleneck on U[1:0])**

ORDERING ALGORITHM RESULTS

JUSTIFICATION	**PROPAGATION**
1) REG22#1,2	*1) REG16#1,2,3,4*
2) OVF	*2) MUL#1,2,3,4*
3) SR#1,2	*3) REG32#1,2,3,4*
4) ADD#1,2	*4) ADD/SUB#1,2*
5) REG20#1,2	*5) REG20#1,2*
6) ADD/SUB#1,2	*6) ADD#1,2*
7) REG32#1,2,3,4	*7) REG22#1,2*
8) MUL#1,2,3,4	*8) SR#1,2*
9) REG16#1,2,3,4	*9) OVF*

LIST OF BOTTLENECKS

CONTROLLABILITY

1) Full Potential on E[30:29], F[30:29],G[30:29], H[30:29], P[21:20], Q[21:20], E[11:0], F[11:0], G[11:0], H[11:0] at [t, t+1]

2) Constant `0' Potential on CIN#1 at [t]

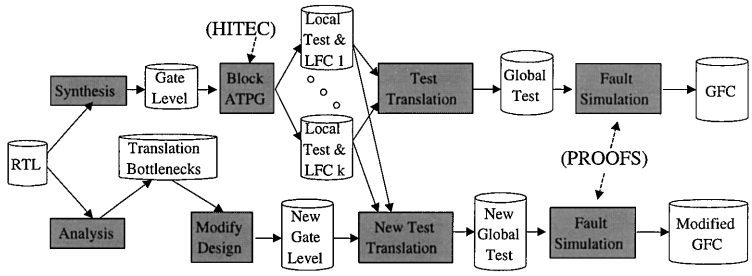
3) Constant `1' Potential on CIN#2 at [t]

4) Full Potential on (ONE OF {E[28:27], E[13:12], E[12]E[28]}), (ONE OF {F[28:27], F[13:12], F[12]F[28]}), (ONE OF {G[28:27], G[13:12], G[12]G[28]}), (ONE OF {H[28:27], H[13:12], H[12]H[28]}) at [t, t+1]

OBSERVABILITY:

Full Potential on N[12:0], O[12:0], T[1:0], U[1:0] at [t, t+1]

Fig. 14. Channel-based analysis example on a pipelined multiplier accumulator (MAC).



	(LFC i)	GFC	Modified GFC
TC100	96.68 %	92.80 %	95.74 %
Multiplier w/o Control	97.45 %	93.27 %	96.65 %
Multiplier with Control	94.52 %	68.46 %	90.64 %
MAC	96.44 %	70.22 %	91.35 %

Fig. 15. Experimental validation flow and results.

controller of the multiplier was also considered, the corresponding loss of coverage due to the test translation bottlenecks was considerably larger, resulting in a drop from 94.52% to 68.46%. Such precipitous fault coverage drops stem from the fact that the controller effects multiple modules. Resolving the bottlenecks almost eliminated the translation problems, as can be seen by the drastically increased coverage of 90.64%.

In the case of the MAC, our analysis identified numerous justification and propagation bottlenecks in the design. The coverage dropped from 96.44% achieved by the local tests to 70.22% achieved by the translated test on the original circuit. After the bottlenecks were resolved, the coverage of the new translated test increased to 91.35%, validating the accuracy of the reported bottlenecks.

The results are summarized in Fig. 15. The faults that remained uncovered after bottleneck resolution were identified to belong to the multiplexers that were used for inserting test points. Except from these faults that do not belong to the original circuit, our analysis methodology has identified precisely the test translation bottlenecks, as indicated by the above results. Further experiments on larger circuits are currently carried out, in order to demonstrate the scalability of our scheme.

10. Conclusion

We have introduced an efficient testability analysis methodology for large, modular designs. We address the challenges associated with the common test framework employed in testing such designs, where test is locally generated at module boundaries and consequently translated to test stimuli applicable at pinouts. In order to avoid the common complexity pitfall of large designs, we have developed two mechanisms for capturing test translation related behavior of modules. A traversal algorithm has been described that examines the satisfiability of the test justification and propagation requirements at each module's boundary, based on this behavior.

Our methodology handles in a uniform way both combinational and sequential, data and control path modules. Furthermore, it addresses problems related to the traversal process, such as feedback loops and reconvergent paths. The test translation bottlenecks are identified in a structural manner and combined among modules. The accuracy of our methodology has been demonstrated and validated experimentally on example circuits, proving its potential to guide efficient test engineering decisions. Our future research plans comprise a concise decision-making mechanism for enhancing the testability of a design. More specifically,

the identified bottlenecks can pinpoint appropriate DFT modifications such as in [3, 5, 7], support the binding phase of high-level test synthesis in a fashion similar to [4], or direct a logic synthesis tool as in [1], in order to provide additional test translation capabilities.

References

1. C-H. Chen, T. Karnik, and D.G. Saab, "Structural and Behavioral Synthesis for Testability Techniques," *IEEE Transactions on CAD of Integrated Circuits and Systems*, Vol. 13, No. 6, pp. 777–785, June 1994.
2. I. Ghosh, A. Raghunathan, and N.K. Jha, "A Design for Testability Technique for RTL Circuits using Control/Data Flow Extraction," *Proc. IEEE/ACM International Conference on Computer Aided Design*, 1996, pp. 329–336.
3. I. Ghosh, N. Jha, and S. Dey, "A Low-Overhead Design for Testability and Test Generation Technique for Core-Based Systems," *Proc. International Test Conference*, 1997, pp. 50–59.
4. A. Orailoğlu and I. Harris, "Microarchitectural Synthesis for Rapid BIST Testing," *IEEE Transactions on CAD of Integrated Circuits and Systems*, Vol. 6, No. 6, pp. 573–586, June 1997.
5. B. Pouya and N. Touba, "Modifying User-Defined Logic for Test Access to Embedded Cores," *Proc. International Test Conference*, 1997, pp. 60–68.
6. R.S. Tupuri and J.A. Abraham, "A Novel Test Generation Method for Processors using Commercial ATPG," *Proc. International Test Conference*, 1997, pp. 743–752.
7. P. Vishakantaiah, T. Thomas, J.A. Abraham, and M.S. Abadir, "AMBIANT: Automatic Generation of Behavioral Modifications for Testability," *Proc. IEEE International Conference on Computer Design*, 1993, pp. 63–66.
8. F. Corno, P. Prinetto, and M. Sonza Reorda, "Testability Analysis and ATPG on Behavioral RT-Level VHDL," *Proc. International Test Conference*, 1997, pp. 753–759.
9. J. Lee and J. Patel, "Testability Analysis Based on Structural and Behavioral Information," *Proc. 11th IEEE VLSI Test Symposium*, 1993, pp. 139–145.
10. P. Vishakantaiah, J.A. Abraham, and M.S. Abadir, "Automatic Test Knowledge Extraction from VHDL (ATKET)," *Proc. 29th ACM/IEEE Design Automation Conference*, 1992, pp. 273–278.
11. B.T. Murray and J.P. Hayes, "Test Propagation Through Modules and Circuits," *Proc. International Test Conference*, 1991, pp. 748–757.
12. M.C. Hansen and J.P. Hayes, "High-Level Test Generation Using Symbolic Scheduling," *Proc. International Test Conference*, 1995, pp. 586–595.
13. K.T. Cheng and A.S. Krishnakumar, "Automatic Functional Test Generation using the Extended Finite State Machine Model," *Proc. 30th ACM/IEEE Design Automation Conference*, 1992, pp. 86–91.
14. D. Moundanos, J.A. Abraham, and Y.V. Hoskote, "A Unified Framework for Design Validation and Manufacturing Test," *Proc. International Test Conference*, 1996, pp. 875–884.
15. J.P. Hayes, *Computer Architecture and Organization*, 3rd edition, McGraw-Hill, 1998.
16. P. Ashenden, *The Designer's Guide to VHDL*, 1st edition, Morgan-Kaufmann Publishers Inc., 1996.
17. T. Niermann and J. Patel, "HITEC: A Test Generation Package for Sequential Circuits," *Proc. European Conference on Design Automation*, 1992, pp. 214–218.
18. T. Niermann, W.T. Cheng, and J. Patel, "PROOFS: A Fast, Memory Efficient Sequential Circuit Fault Simulator," *Proc. 27th ACM/IEEE Design Automation Conference*, 1990, pp. 535–540.

Yiorgos Makris received the Diploma of Computer Engineering and Informatics from the University of Patras, Greece, in 1995 and the M.S. degree in Computer Engineering from the University of California, San Diego, in 1997. He is currently working towards his Ph.D. degree in Computer Engineering, at the University of California, San Diego. His research interests include testability analysis, test generation and DFT.

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. Prof. Orailoğlu's research interests include the high-level synthesis of fault-tolerant microarchitectures, and the synthesis of testable designs.