Enhancing Reliability of RTL Controller-Datapath Circuits via Invariant-Based Concurrent Test

Yiorgos Makris, Ismet Bayraktaroglu, and Alex Orailoglu

***Abstract*—We present a low-cost concurrent test methodology for enhancing the reliability of RTL controller-datapath circuits, based on the notion of path invariance. The fundamental observation supporting the proposed methodology is that the inherent transparency behavior of RTL components, typically utilized for hierarchical off-line test, renders rich sources of invariance within a circuit. Furthermore, additional sources of invariance are obtained by examining the algorithmic interaction between the controller, and the datapath of the circuit. A judicious selection & combination of modular transparency functions, based on the algorithm implemented by the controller-datapath pair, yields a powerful set of invariant paths in a design. Compliance to the invariant behavior is checked whenever the latter is activated. Thus, such paths enable a simple, yet very efficient concurrent test capability, achieving fault security in excess of 90% while keeping the hardware overhead below 40% on complicated, difficult-to-test, sequential benchmark circuits. By exploiting fine-grained design invariance, the proposed methodology enhances circuit reliability, and contributes a low-cost concurrent test direction, applicable to general RTL circuits.**

***Index Terms*—Algorithmic invariance, concurrent test, controller-datapath, path invariance, transparency.**

ACRONYMS[1]

ALU	Arithmetic Logic Unit
ASIC	Application Specific Integrated Circuit
ATPG	Automatic Test Pattern Generation
BIST	Built-In Self-Test
DFT	Design for Test
DSP	Digital Signal Processor
GCD	Greatest Common Divisor
RTL	Register Transfer Level

NOTATION

f, g	functions
x, y	variables
t	time step
t	time step
A, B, C, X, Y, Z, N	RTL word variables
IN, OUT	I/O variables
$S_0 - S_4$	controller states
$SEL, LOAD, CLEAR$	control signals
$START, READY$	environment signals

Manuscript received October 5, 2001; revised April 15, 2003. Responsible Editor: J. Bowles.

Y. Makris is with the Electrical Engineering Department, Yale University, New Haven, CT 06520 USA (e-mail: yiorgos.makris@yale.edu).

I. Bayraktaroglu is with Sun Microsystems, MS USUN03-315, Sunnyvale, CA 94085 USA (e-mail: ismet.bayraktaroglu@sun.com).

A. Orailoglu is with the Computer Science and Engineering Department, UC San Diego, San Diego, CA 92093, USA (e-mail: alex@cs.ucsd.edu).

Digital Object Identifier 10.1109/TR.2004.829175

[1]The singular and plural of an acronym are always spelled the same.

I. INTRODUCTION

THE ability to test the functionality of a circuit during usual operation is becoming an increasingly desirable property of modern designs. Identifying & discarding faulty results before they are further used constitutes a powerful design attribute of ASIC performing critical computations, such as DSP, and ALU. While this capability can be provided by hardware duplication schemes, such methods incur considerable cost in terms of area overhead, and possible performance degradation. Devising a low-cost, nonintrusive concurrent test method which enhances circuit reliability by providing high fault-security is therefore a challenging task.

Current state-of-the-art efforts in concurrent & on-line test research [1] can be roughly categorized in 2 main directions, as shown in Fig. 1. Approaches along the first direction [2], [3] use vectors & responses generated off-line, which are stored on-chip, possibly compacted. Whenever one of these vectors appears at the inputs during usual functionality, the actual response is checked against the stored anticipated response. Such approaches, however, require inordinate hardware overhead for storing a sufficient number of test vectors & unpredictable time until all vectors are applied, therefore suffering from very long fault-detection latency. Their applicability is consequently limited largely to combinational circuits. Approaches along the second direction use coarse behavioral invariance either inherent in the design [4], [5], or imposed through error detection codes [6], [7], in order to check the correctness of the functionality. In this case, while the circuit computes $f(x)$ for input x, an additional function, $g(x)$, with a well-defined, simple-to-check relation to $f(x)$, is also computed. The operational health of the circuit is verified by checking that the relation between $f(x)$ & $g(x)$ holds. Fault-detection latency is not a problem in such methods, because the coarse invariance function, or the error detection code are constantly active & continuously checked. However, coarse behavioral design invariance is inherently available only in limited domains; furthermore, despite being nonintrusive, typical implementations of coarse invariance functions can necessitate appreciable area overhead [4]. Invariably, error detection codes, which constitute, in essence, a mechanism for introducing invariance in a circuit, incur high area & performance overhead due to their intrusive nature.

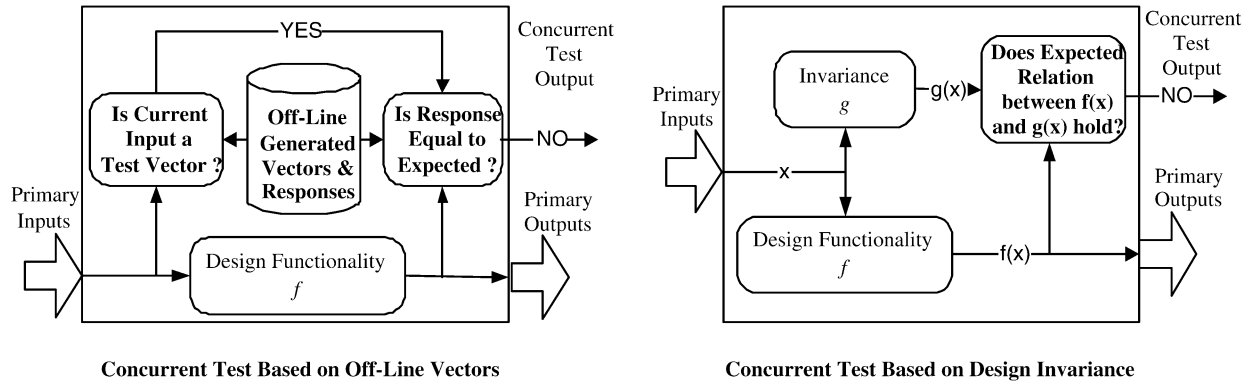


Fig. 1. Typical concurrent test approaches.

In an effort to enhance circuit reliability, and in contrast to the aforementioned approaches, a low-cost, nonintrusive, concurrent test method is proposed for arbitrary RTL controller-datapath circuits, using simple invariance functions distributed across several paths in the design. Invariance is established by examining the algorithmic controller-datapath interaction & composing inherent transparency functionality of RTL components into transparency-based & algorithmically invariant paths. The proposed method is:

- generic, because it can be applied to any RTL controller-datapath circuit;
- nonintrusive, because it neither interacts with, nor imposes additional burden on, usual circuit operation; and
- low-cost, because transparency-based & algorithmic invariance functions are typically very simple to implement.

On the downside, unlike global circuit invariance, invariant paths are not necessarily constantly active. Therefore, because fault detection latency depends on invariance activation frequency, it is essential that low-cost, frequently activated, invariant paths providing high fault-security be carefully selected.

Section II is an overview of the proposed method. Section III examines the applicability of the inherent transparency functionality of RTL modules for concurrent test. Section IV addresses the hardware overhead problem through transparent path composition. Section V uses algorithmic path invariance for enhancing fault security. Section VI discusses the latency problem. Section VII is an example of the proposed methodology. Section VIII provides an experimental setup along with results on difficult-to-test, sequential circuits.

II. Methodology Overview

Fig. 2 shows an overview of the proposed concurrent test methodology. Given a controller-datapath pair, several invariant paths are identified based on the transparency behavior, and the algorithmic interaction of the RTL components. The invariant behavior of each path is activated under a—possibly empty—set of conditions defined on internal signal entities of the design. Therefore, checking the correctness of each invariant path during usual circuit operation requires 2 elements:

1) the logic that examines whether the conditions hold, thus activating the invariant path; and
2) the actual checker that verifies that the invariance function is not violated.

If an invariant path is activated, but the invariance relationship does not hold, then the concurrent test output indicates that an error has occurred.

The distribution of invariance checking throughout the design results in simpler invariance relationships, and therefore lower area overhead for the invariance checkers, as compared to a global design invariance checker. Additionally, the used invariant paths rely on transparency functions of simple RTL modules, extending the applicability domain of invariance-based concurrent test to general RTL controller-datapath designs, as opposed to the limited scope of global design invariance, which can be found only in very few circuits. Furthermore, as demonstrated in Fig. 2, whenever the activation conditions of invariant paths are met, the invariance is checked in parallel with used circuit interaction, thus constituting a nonintrusive concurrent test scheme. However, unlike global circuit invariance which holds continuously, invariant paths only detect faults when activated, pointing out a trade-off between fault detection latency & area overhead. Consequently, achieving high fault security requires that a small number of frequently activated, key invariant paths be selected, capable of covering most faults in both the datapath, and the controller. The identification, evaluation, and selection of these paths are discussed in this paper.

III. Concurrent Test Via Transparency

Modular transparency constitutes a key design attribute, based on which several off-line hierarchical test methodologies have been devised [8]–[10]. Hierarchical test methods address the complexity of test generation in a divide & conquer fashion, wherein test vectors are locally generated for each module, and subsequently translated & applied from the global circuit boundary. For this purpose, transparency-based hierarchical test-paths are required, to justify test vectors from the primary inputs to the inputs of the module under test, and propagate test responses from the outputs of the module under test to the

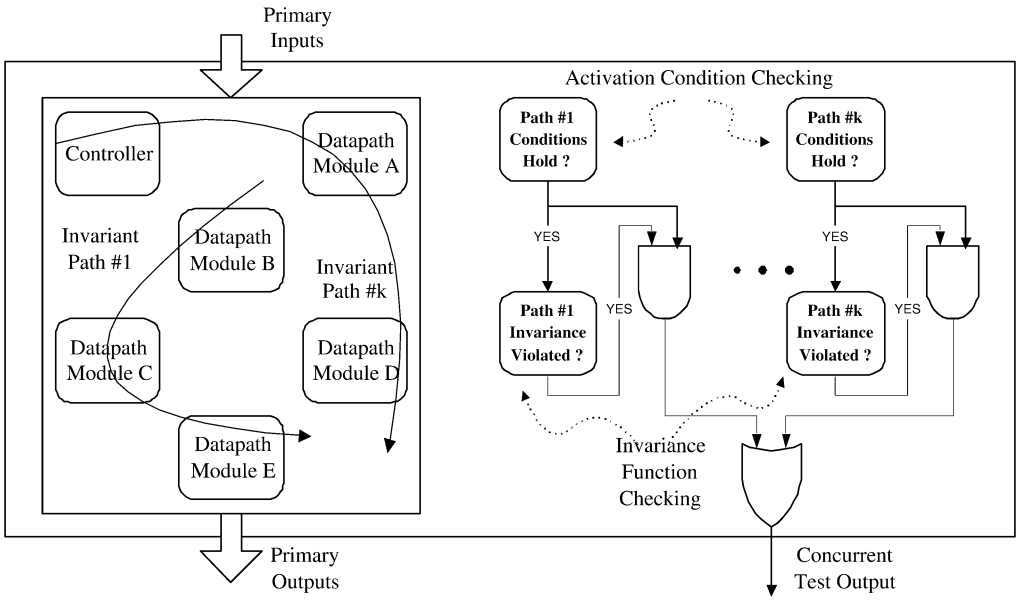


Fig. 2. Concurrent test based on path invariance.

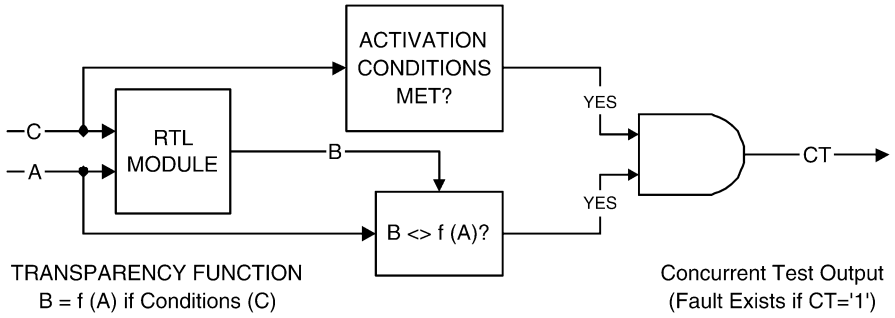


Fig. 3. Concurrent test via transparency functions.

primary outputs. Transparency behavior of RTL components, whether inherent in the design [11]–[14] or explicitly incorporated through DFT [15], provides a simple & rapid mechanism for traversing a circuit in order to access & test each module through hierarchical test paths. This section examines the use of transparency functions of RTL modules in concurrent test, in an effort to integrate off-line & on-line test, and amortize the corresponding cost.

Transparency is typically defined as *surjective*, *injective*, or *bijective* functions. A function $f : X \rightarrow Y$ is *surjective* (or *onto*) *iff* for every y in the codomain Y there is at least one x in the domain X such that $f(x) = y$. A function $f : X \rightarrow Y$ is *injective* (or *1-to-1*) *iff* for every y in the codomain Y there is at most one x in the domain X with $f(x) = y$. A function $f : X \rightarrow Y$ is *bijective* (or *1-to-1 and onto*) *iff* it is both *injective* & *surjective*, i.e., if for every y in the codomain Y there is exactly one x in the domain X with $f(x) = y$. The *onto* property of *surjective* & *bijective* functions preserves the ability to obtain all possible vectors while traversing a module. Similarly, the *1-to-1* property of *injective* & *bijective* preserves the ability to distinguish between correct, and erroneous responses. Transparency functions commonly used in hierarchical off-line test, such as *Identity* & *Inversion*, establish simple relations between the inputs, and the outputs of the transparent module. We check these relations every time the activation conditions hold during usual operation of the circuit in order to provide a concurrent test scheme capable of detecting any fault that distorts the transparency of the module, as shown in Fig. 3.

Now examine the efficiency of this scheme for detecting faults in simple RTL modules. Consider the 8-bit 2-to-1 MUX in Fig. 4(a), along with its 2 inherent transparency functions, $C = A\ IF\ SEL = 1\ \&\ C = B\ IF\ SEL = 0$. The simple concurrent test mechanism in Fig. 3 detects 100% of the faults in the MUX. This result is anticipated because the 2 transparency functions comprise the complete functionality of the MUX, thus implicitly duplicating it. Now consider a module where the transparency functions do not cover its complete functionality, and therefore their hardware implementation is appreciably cheaper Fig. 4(b) shows an 8-bit SUBTRACTOR, along with two transparency functions, $C = A\ IF\ B = 0\ \&\ C = B\ IF\ A = 2^{*}B$. The proposed concurrent test mechanism results in 80% fault coverage in the SUBTRACTOR, implying that many faults can be detected through a few, judiciously selected, transparency functions. Furthermore, the hardware for implementing the transparency functions is very simple, requiring only shift & compare operations.

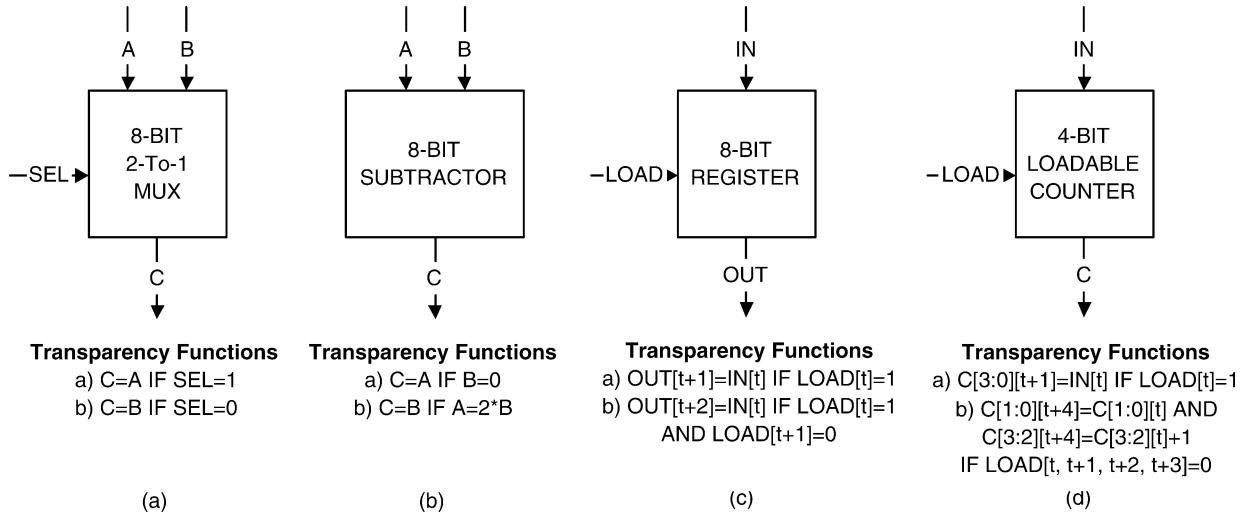


Fig. 4. Examples of transparency functions in RTL components.

The idea is readily extendible into sequential modules. Fig. 4(c) demonstrates the same scheme for an 8-bit REGISTER, where 99% of the faults are detected by the 2 transparency functions $OUT[t+1] = IN[t]\ IF\ LOAD[t] = 1\ \&OUT[t+2] = IN[t]\ IFLOAD[t] = 1\ AND\ LOAD[t+1] = 0$. Finally, Fig. 4(d) shows how this scheme is applied to a 4-bit LOADABLE COUNTER. Despite the fact that counters cause serious problems to ATPG algorithms in general, there exist interesting transparency functions which cover many faults. For example, the 2 transparency functions: $C[3:0][t+1] = IN[3:0][t]\ IF\ LOAD[t] = 1\ \&C[1:0][t+4] = C[1:0][t]\ AND\ C[3:2][t+4] = C[3:2][t]+1\ IF\ LOAD[t,t+1,t+2,t+3] = 0$ cover 88% of the COUNTER faults. Fault coverage can be calculated either through ATPG [16], or through exhaustive fault simulation [17] on the modified circuit, using only the concurrent test output as an observable primary output. In support of the proposed concurrent test scheme, a library of RTL components & corresponding transparency functions is devised, and achieves fault coverage ranging between 75%–100%.

As demonstrated through the examples in this section, a judicious selection of a few transparency functions is adequate for detecting a large number of faults in RTL modules. However, several issues need to be addressed before this observation is successfully converted into a methodology.

1) The cost of the hardware required for checking all the necessary transparency functions for each RTL module would make such a scheme unrealistic.
2) The fault security which can be achieved through checking only the transparency behavior of each module might not be adequate in itself, necessitating additional sources of design invariance.
3) The activation frequency of the conditions impacts directly the latency of detecting a fault in the design through transparency functions.

Sections IV–VI describe how these issues are addressed in the proposed methodology.

IV. Transparency-Based Path Invariance

Incorporating a checking mechanism for each transparency function of every RTL module in the design would result in inordinate hardware overhead. To reduce this cost, several transparency functions can be combined on a transparent path, such that only one checking mechanism for the complete path suffices for all the constituent transparency functions. Such a path can span not only across several modules of the design, but also across several clock cycles. In this manner, complicated sequential behavior of the design can also be checked along with the simple transparency functions.

The circuit in Fig. 5 is an example of transparent path composition. Three transparency functions are provided for each of the 2 RTL modules in the design, requiring 6 distinct checking mechanisms. However, a path spanning 5 clock cycles can be composed, comprising all 6 transparency functions. Although we still need hardware for monitoring the activation conditions of all 6 transparency functions, the path transparency function requires only 1 check instead of 6. As a result, path composition appreciably reduces the required hardware, while preserving the attained fault coverage.

Off-line test path composition has been extensively studied [8], [9], [11], [15] to provide transparent reachability paths for hierarchical testing. The basic path construction capability is therefore available, and can be extended to automate the composition of transparency-based invariant paths. However, these off-line path composition approaches need to be tuned to the particularities of concurrent testing. Ideally, the algorithm composes a few long paths covering all the modular transparency functions, to minimize hardware overhead. However, the activation frequency of the path is important in concurrent testing, due to its latency impact. Off-line test path composition algorithms verify that there is no conflict among the activation conditions on the path but do not consider activation frequency, because the path can be fully controlled during off-line test applications. Concurrent testing, however, relies on usual operation to activate the transparent paths. Therefore, the transparency fre-

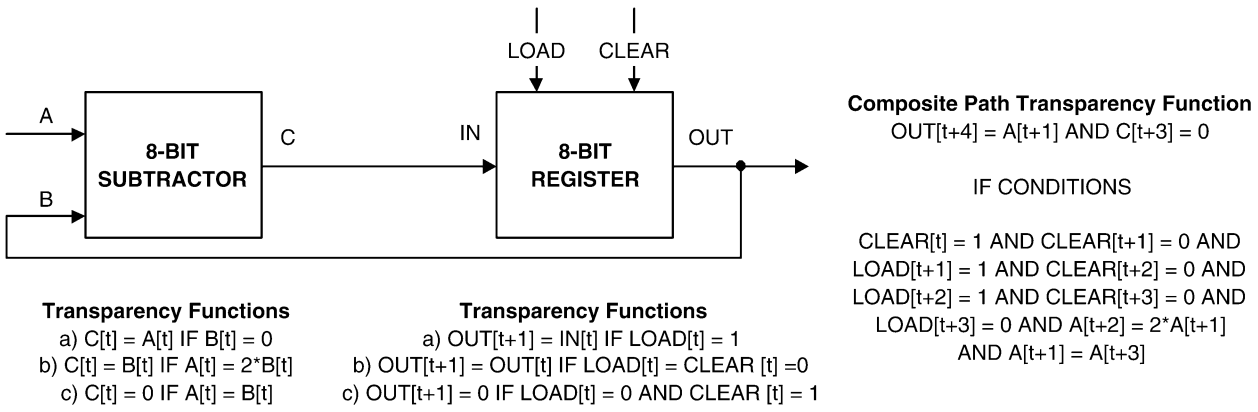


Fig. 5. Transparency path composition example.

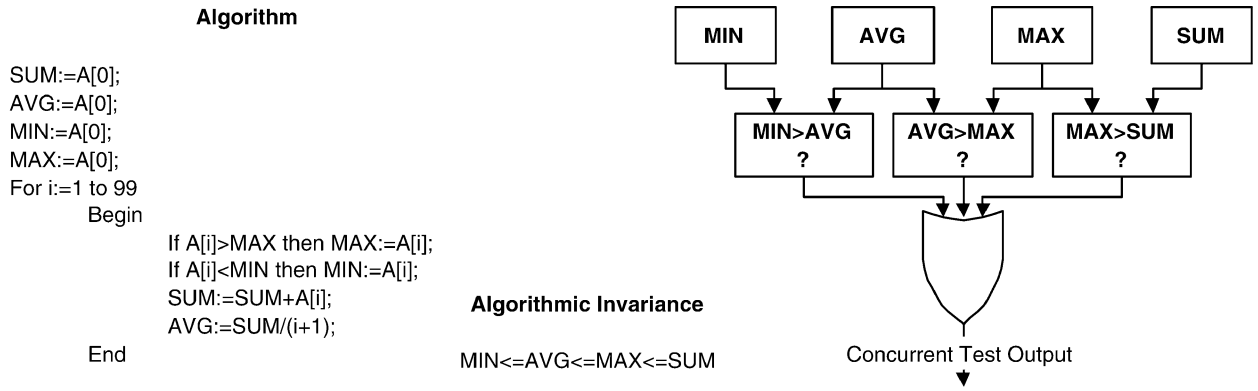


Fig. 6. Algorithmic path invariance example.

quency determined by the complexity of the activation conditions needs to be considered to balance the number, the cost, and the efficiency of the transparent paths. Finally, to reduce the cost of the checker for the composite path function, simple modular transparency functions, such as identity & inversion, are used. These issues are addressed in a modified version of the hierarchical test path composition algorithm & tool introduced in [14].

V. Algorithmic Path Invariance

While modular transparency functions & the composite invariant paths can detect many faults, the resulting fault coverage might not be adequate to ensure high fault security in the design. Additional sources of invariance, capable of checking for faults which can not be detected through transparency functions, are therefore required. Such invariance can be obtained from the algorithmic interaction between the datapath, and the controller of the design. Algorithmic invariance captures the restrictions imposed by the controller on the datapath. Any fault causing a deviation from this restricted behavior is detected through algorithmic invariance checking. Algorithmic invariance frequently has an equivalence relation to the conditions. In this case, verify not only the existence of invariance when the activation conditions hold, but also the lack of invariance when the activation conditions do not hold, thus increasing the number of detected faults.

Fig. 6 is an example of algorithmic invariance, and the consequent concurrent test scheme. The example algorithm keeps track of the minimum, maximum, sum, & average of an array of numbers, and can be implemented as a simple controller-datapath pair. The datapath comprises the registers & arithmetic units, while the controller comprises a counter & comparators for controlling the loop & enabling the registers, respectively. Several invariant attributes can be identified in this algorithm. For example, during usual circuit operation, the values of the 4 design-registers are related through the inequality $MIN \leq AVG \leq MAX \leq SUM$. Any fault violating this algorithmic invariance is detected through the additional hardware for concurrent testing shown in Fig. 6.

The most interesting cases of algorithmic invariance cover faults in the controller, where transparency rarely exists. Additionally, algorithmic invariance has a clear advantage in terms of activation frequency; algorithmically invariant paths are always active, and therefore the concurrent test scheme continuously checks for errors. If algorithmic invariance exists only on part of the algorithm, then, relying on a particular control path to be

taken, the activation frequency reflects the execution frequency of the specific part of the algorithm. In short, algorithmic path invariance complements transparency path invariance in terms of both fault coverage & activation frequency.

VI. Invariance Activation Frequency

Unlike off-line testing where faults are detected before the circuit performs its intended operation, fault detection latency, the time interval between fault occurrence & fault detection, is a critical issue during concurrent testing. The later a fault is detected after its occurrence, the higher the chances that an erroneous result might slip undetected. Latency has not only been a difficult problem for concurrent test methodologies to solve, but also hard to estimate. While some concurrent test methods [4], [6], [7] guarantee zero-latency, they incur excessive area overhead, thus limiting their applicability. The invariance-based concurrent test methodology in [5] reduces the area overhead at the cost of introducing latency. Within input monitoring techniques, such as [2], [3], latency heavily depends on the values which appear at the inputs of the circuit during usual operation. Similarly, in the proposed methodology, latency depends on the activation frequency of the invariant paths. Invariant path activation, however, requires only a few conditions to be met, and not a complete vector, thus enabling all vectors satisfying these conditions to be used for concurrent test. Therefore, invariant path activation frequency is anticipated to be appreciably higher than the activation frequency of techniques using stored test vectors. In addition, the proposed scheme provides flexibility in judiciously selecting among alternative invariant paths, based on the complexity of the activation conditions.

Depending on activation frequency invariant paths can be categorized into 3 types:

- Type 1 comprises invariant paths that are always active, such as unconditional algorithmic invariance. This is the most desirable type of invariance because all faults which can be detected through this checking mechanism will have zero latency.
- Type 2 comprises invariant paths that are not always active, but the activation conditions are frequently met during normal operation. This type of invariance is also highly desirable, because the frequent activation will help keep latency low. Conditional algorithmic invariance is a good example of this type.
- Type 3 comprises invariant paths that have low probability of activation. As an example of this type, transparency relying on specific values of wide datapath signals has low activation probability, assuming uniform distribution. Low priority should be given to the incorporation of this type of invariance checking mechanisms in the design.

Activation frequency of an invariant path can be estimated in two methods.

- Method #1 is a static analysis of the controller-datapath interaction. Starting from the final states of the controller, trace backward the datapath conditions that need to hold for the algorithm to terminate. This information is used to characterize the severity of the activation conditions for each invariant path. When this type of analysis becomes too expensive due to the large number of controller states and paths, Method #2 is used.
- In Method #2, a dynamic profiling mechanism estimates the activation frequency. In profiling, a large number of random vectors are simulated, and the number of times that the activation conditions of an invariant path are satisfied is noted. This provides an indication of the relative activation frequency of alternative invariant paths, thus guiding selection among them.

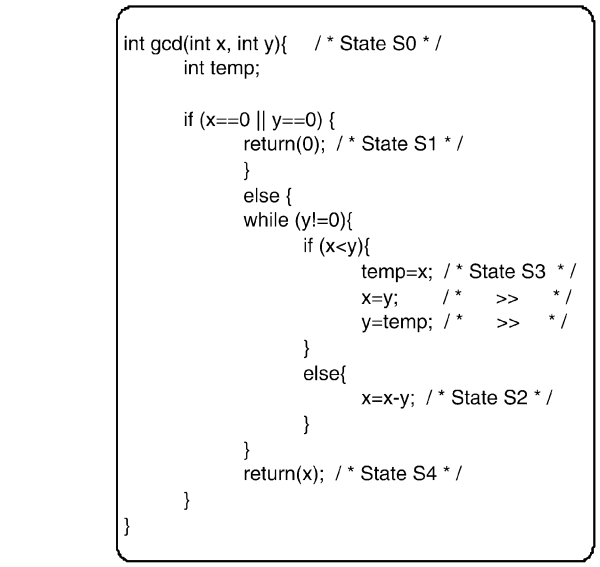
From a latency perspective, many short, but frequently activated, invariant paths should be preferred over a few, long but rarely activated invariant paths. The number of invariant paths, their activation frequency, and the number of activation conditions (and therefore the length of the path) are conflicting objectives. The given classification of invariant paths, along with a fault coverage analysis of each, provides the necessary parameters for developing heuristics to exploit this trade-off; and iteratively balance area overhead, attainable fault coverage, and fault detection latency.

VII. Example

The invariance-based concurrent test is demonstrated on a difficult-to-test benchmark, the GCD circuit, in Fig. 7, on which three example tests are performed.

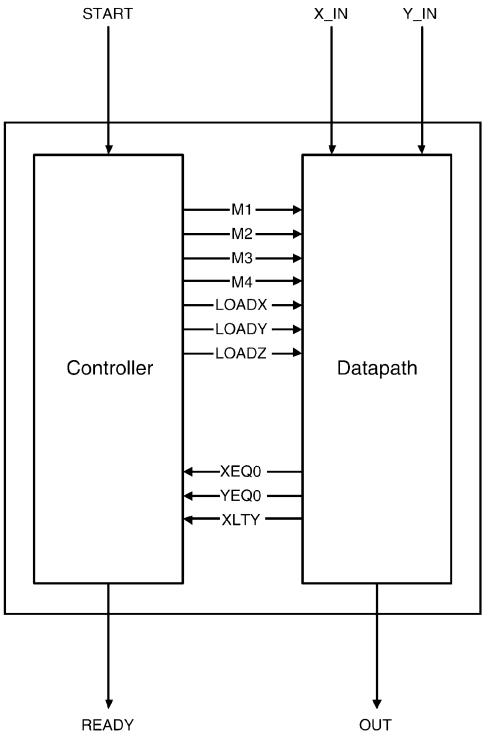
- Example #1 uses transparency of a single RTL module.
- Example #2 monitors a transparency-based invariant path.
- Example #3 exploits algorithmic invariance.

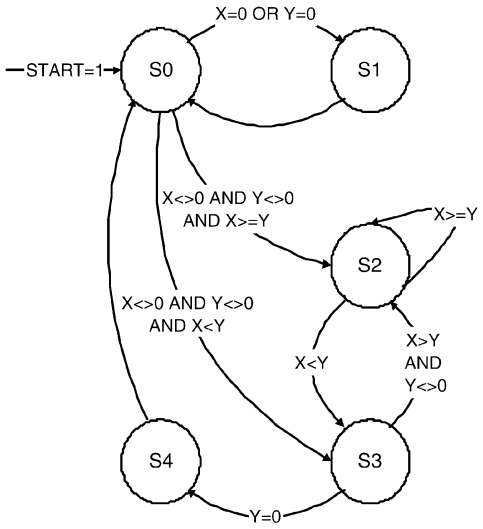
Example #1 is based on the transparency function $C[t] = B[t]\ if\ A[t] = 2^{*}B[t]$ of the SUBTRACTOR. As discussed in Section III, this function covers many faults in the SUBTRACTOR. Furthermore, checking that the condition $A[t] = 2^{*}B[t]$ holds can be easily performed through a shifter & a comparator. Additionally, a simple comparator suffices for checking the transparency function $C[t] = B[t]$. Fig. 8(a) shows the hardware implementation of this simple concurrent test mechanism. The most important attribute of this invariance check, however, is its frequency of activation. Within the controller-datapath implementation of the GCD circuit, this transparency function of the SUBTRACTOR module will be activated almost once per GCD calculation. This is a conclusion which can be reached either by analyzing the GCD algorithm or by examining the controller-datapath interaction of the implementation. This subtract & swap GCD algorithm terminates when $y = 0$, necessitating that two clock cycles earlier $x = y$ for this to be achieved through subtraction & swapping. This in turn requires that either the initial values of x & y are equal, or that $x = 2 * y$ at some point in the algorithm, which is the activation condition of the SUBTRACTOR transparency function. These three features (the good fault coverage, the simplicity of checking the

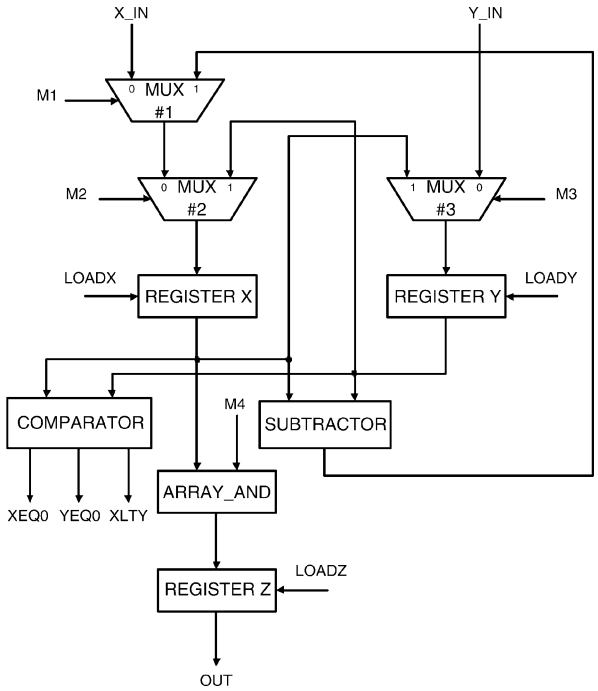


```
int gcd(int x, int y){    / * State S0 * /
        int temp;

        if (x==0 || y==0) {
                return(0);  / * State S1 * /
                }
                else {
                while (y!=0){
                        if (x<y){
                                temp=x;  / * State S3  * /
                                x=y;       / *      >>      * /
                                y=temp;  / *      >>      * /
                        }
                        else{
                                x=x-y;  / * State S2 * /
                        }
                }
                return(x);  / * State S4 * /
        }
}
```







	M1	M2	M3	M4	LOADX	LOADY	LOADZ	READY
S0	0	0	0	1	1	1	1	0
S1	0	0	0	1	0	0	1	1
S2	1	0	0	0	1	0	0	0
S3	0	1	1	0	1	1	0	0
S4	0	0	0	1	0	0	1	1

Fig. 7. The GCD benchmark circuit.

conditions & the transparency function, and the high frequency of activation) make this invariance check a very efficient concurrent test mechanism.

Example #2 is a transparency-based invariant path, resulting from the path composition methodology of Section IV. The composite function of this transparent path is $OUT[t+4] = X[t]\ if\ X[t] = Y[t]\ AND\ X[t] <> 0$. This composite transparency function exercises several transparency functions of the GCD datapath modules, and spans across 4 clock cycles, thus also exercising the corresponding controller behavior. The transparency functions of the modules exercised through this path are:

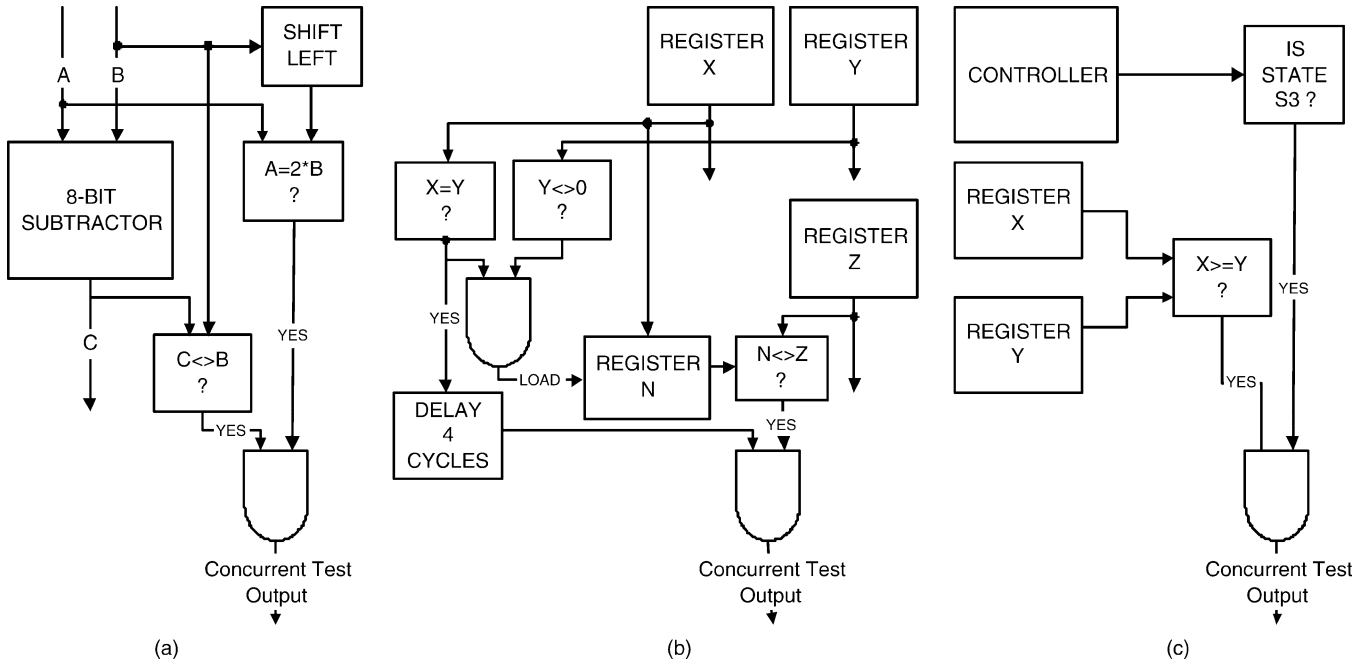


Fig. 8. Invariance checking examples on the GCD benchmark.

REGISTER X:

i) $\mathrm{REG\_X\_OUT}[t+1] = \mathrm{REG\_X\_IN}[t]$ IF $\mathrm{LOAD}X[t] = '1'$

REGISTER Y:

i) $\mathrm{REG\_Y\_OUT}[t+1] = \mathrm{REG\_Y\_IN}[t]$ IF $\mathrm{LOAD}Y[t] = '1'$

ii) $\mathrm{REG\_Y\_OUT}[t+1] = \mathrm{REG\_Y\_OUT}[t]$ IF $\mathrm{LOAD}Y[t] = '0'$

REGISTER Z:

i) $\mathrm{REG\_Z\_OUT}[t+1] = \mathrm{REG\_Z\_IN}[t]$ IF $\mathrm{LOAD}Z[t] = '1'$

MUX# 1:

i) $\mathrm{OUT}[t] = \mathrm{RIGHT\_INPUT}[t]$ IF $M1[t] = '1'$

MUX# 2:

i) $\mathrm{OUT}[t] = \mathrm{RIGHT\_INPUT}[t]$ IF $M2[t] = '1'$

ii) $\mathrm{OUT}[t] = \mathrm{LEFT\_INPUT}[t]$ IF $M2[t] = '0'$

MUX# 3:

i) $\mathrm{OUT}[t] = \mathrm{LEFT\_INPUT}[t]$ IF $M3[t] = '1'$

SUBTRACTOR:

i) $C[t] = '0'$ IF $A[t] = B[t]$

ARRAY_AND:

i) $\mathrm{OUT}[t] = \mathrm{IN}[t]$ IF $M4[t] = '1'$

COMPARATOR:

i) $\mathrm{XEQ0}[t] = '1'$ IF $X[t] = '0'$

ii) $\mathrm{YEQ0}[t] = '1'$ IF $Y[t] = '0'$

iii) $\mathrm{XLT}Y[t] = '1'$ IF $X[t] < Y[t]$

CONTROLLER:

i) State Sequence $S0 - S2 - S3 - S4$

ii) State Sequence $S2 - S2 - S3 - S4$

Any fault causing a discrepancy between the outcome of these functions on the faulty & the nonfaulty circuit will be detected by the concurrent checking mechanism shown in Fig. 8(b). In this case, an additional register N is required in order to store the value of register X whenever the condition $X = Y\ AND\ X <> 0$ holds. The stored value is compared to the output of the circuit 4 clock cycles later, to check the transparent path function $OUT[t+4] = X[t]\ if\ X[t] = Y[t]\ AND\ X[t] <> 0$. The hardware required is still simple, and its cost is justified by the large number of modular transparency functions covered by this path. The activation frequency of this transparency-based invariant path is also once per GCD calculation, for the same reasons as in the example check #1; the condition $x = y$ has to be satisfied before the GCD algorithm terminates. Therefore, checking this transparency-based invariant path also constitutes an effective concurrent test.

Example #3 identifies algorithmic invariance in the calculation performed by the GCD circuit. The activation frequency analysis given in example #1 has already demonstrated how algorithmic invariance can assist in choosing modular transparency behavior & effective transparent paths. In addition, consider the first part of the *if statement* within the *while loop* of the GCD algorithm. This part of the code performs the swapping which in the implementation shown is performed in state $S3$ of the controller, reached only when the condition $x < y$ holds. Performing a swap while $x \geq y$ would be algorithmically invalid during usual operation of the GCD calculation. This algorithmic invariance can be exploited to provide an additional concurrent test capability, demonstrated in Fig. 8(c). Such algorithmic invariance checks are mostly capable of detecting controller faults, although depending on the circuit, datapath algorithmic invariance may also exist. The hardware cost is very low, while the activation frequency is not a problem in this case, because the algorithmic invariance is always active.

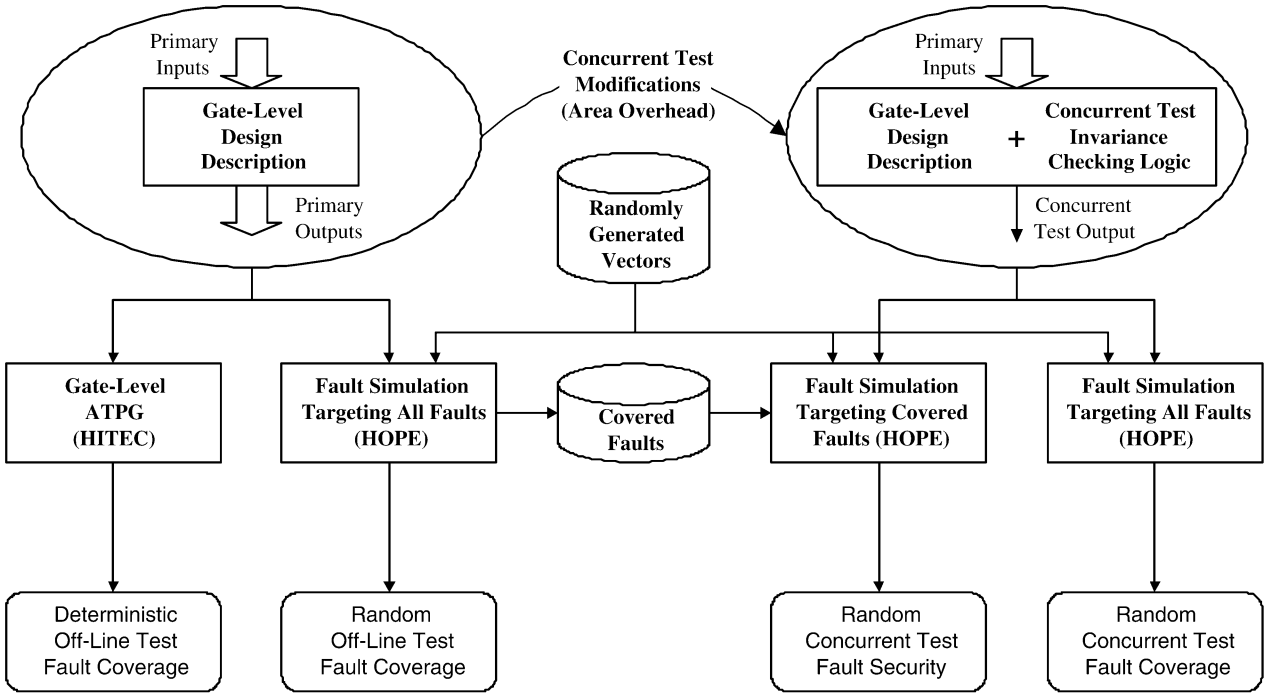


Fig. 9. Experimental validation setup.

TABLE I
EXPERIMENTAL RESULTS

Benchmark Circuit	*Deterministic Off-Line Test Fault Coverage*	*Random Off-Line Test Fault Coverage*	*Random Concurrent Test Fault Security*	*Random Concurrent Test Fault Coverage*	*Gates and Flip-Flops Before Changes*	*Gates and Flip-Flops After Changes*	*Area Overhead*
GCD	863/973 Faults (88.69%)	822/973 Faults (84.48%)	744/822 Faults (90.51%)	781/973 Faults (80.26%)	480 Gates and 27 F-Fs	646 Gates and 39 F-Fs	36.39%
MINMAX	1117/1185 Faults (94.26%)	1054/1185 Faults (88.94%)	952/1054 Faults (90.32%)	996/1185 Faults (84.05%)	534 Gates and 56 F-Fs	781 Gates and 64 F-Fs	36.80%
MULT	798/834 Faults (95.68%)	769/834 Faults (92.20%)	693/769 Faults (90.11%)	719/834 Faults (86.21%)	356 Gates and 30 F-Fs	471 Gates and 48 F-Fs	39.28%

Many more invariance checking examples can be demonstrated & examined on the GCD circuit until a beneficial ratio of fault coverage & hardware cost is achieved. To further reduce the cost, the hardware used for checking individual transparency functions & conditions can be combined & optimized. Finally, although several concurrent checking mechanisms are incorporated in the design, only one concurrent test output pin is required, driven by the OR function of the distributed checking mechanisms.

VIII. EXPERIMENTAL RESULTS

To evaluate the proposed concurrent test methodology, examine the fault security & fault coverage achieved by the identified path invariance, as well as the area overhead imposed by the invariance checking hardware. For this purpose, the experimental setup in Fig. 9 is applied to three benchmark designs. These benchmarks are difficult-to-test sequential circuits, implemented as controller-datapath pairs, on which ATPG has a hard time reaching high fault coverage. The first benchmark is the GCD circuit examined in the example of Section VII. The second benchmark is the MINMAX circuit, included in a set of benchmarks by Politecnico di Torino [18]. The third benchmark is a shift-&-add 8-bit multiplier described in [19].

First apply gate-level ATPG using HITEC [16] to obtain the *deterministic off-line test fault coverage* as a reference point. Subsequently, for the main part of the experimental validation, use fault simulation of random input values for each design. For the GCD circuit, 1000 random pairs of numbers are generated, and the corresponding GCD are calculated. The number of clock cycles of each GCD operation depends on the actual inputs. Therefore, to imitate usual circuit operation, we had to simulate the circuit functionality through a computer program, which calculates the number of clock cycles necessary for the GCD calculation of the 1000 random pairs. In this case, the total length of patterns is around 24, 000. This was not a difficulty in the MINMAX circuit where 1000 randomly generated patterns are applied to the circuit; and the minimum, maximum, & av-

erage numbers are calculated according to the benchmark algorithm. Finally, in the $MULT$ benchmark every multiplication takes 17 clock cycles, so in total 17, 000 patterns are required to simulate 1000 random multiplications.

These vectors are fault simulated on the design using HOPE [17], and the *random off-line test fault coverage* is obtained, along with the set of covered faults. The design is then modified according to the proposed concurrent test methodology, wherein path invariance checking hardware & a concurrent test output pin are added to the design. Only the concurrent test output is considered a primary output in the modified design. The same random vectors are subsequently fault simulated on the modified design, targeting only the faults covered in off-line random vector fault simulation. The fault coverage achieved in this experiment indicates the percentage of faults which can be detected both on the original & on the modified design, thus providing the *random concurrent test fault security*. In addition, the random vectors are fault simulated, targeting all faults in the modified design, to obtain the *random concurrent test fault coverage*.

Table I summarizes the results, where the area overhead of the proposed scheme is also shown, assuming only 2-input gates & an equivalent of four gates for each flip-flop. To reduce the cost, the hardware used for the transparency functions is combined & optimized, and only one output pin is used. The results demonstrate that the proposed concurrent test method achieves fault security exceeding 90%, while keeping the area overhead below 40%. In addition, the achieved *random concurrent test fault coverage* is only 6% worse than *random off-line test fault coverage*. It is important that a large portion of the undetected faults are due to primary I/O faults which no invariance-based, concurrent test methodology can detect.

REFERENCES

[1] M. Nicolaidis and Y. Zorian, "On-line testing for VLSI—A compendium of approaches," *Journal of Electronic Testing: Theory and Applications*, vol. 12, no. 1–2, pp. 7–20, 1998.

[2] K. K. Saluja, R. Sharma, and C. R. Kime, "A concurrent testing technique for digital circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 7, pp. 1250–1260, 1988.

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

[4] A. Chatterjee and R. K. Roy, "Concurrent error detection in nonlinear digital circuits with applications to adaptive filters," in *International Conference on Computer Design*, 1993, pp. 606–609.

[5] I. Bayraktaroglu and A. Orailoglu, "Concurrent test for digital linear systems," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 20, pp. 1775–1791, 2001.

[6] C. Stroud, M. Ding, S. Seshadri, I. Kim, S. Roy, S. Wu, and R. Karri, "A parametrized VHDL library for on-line testing," in *International Test Conference*, 1997, pp. 479–488.

[7] C. Zeng, N. Saxena, and E. J. McCluskey, "Finite state machine synthesis with concurrent error detection," in *International Test Conference*, 1999, pp. 672–679.

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

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

[10] Y. Makris, J. Collins, A. Orailoglu, and P. Vishakantaiah, "TRANSPARENT: A system for RTL testability analysis, DFT guidance and hierarchical test generation," in *Custom Integrated Circuits Conference*, 1999, pp. 159–162.

[11] S. Freeman, "Test generation for data-path logic: The F-path method," *IEEE Journal of Solid-State Circuits*, vol. 23, pp. 421–427, 1988.

[12] B. T. Murray and J. P. Hayes, "Test propagation through modules and circuits," in *International Test Conference*, 1991, pp. 748–757.

[13] P. Vishakantaiah, J. A. Abraham, and M. S. Abadir, "Automatic test knowledge extraction from VHDL (ATKET)," in *Design Automation Conference*, 1992, pp. 273–278.

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

[15] ——, "DFT guidance through RTL test justification and propagation analysis," in *International Test Conference*, 1998, pp. 668–677.

[16] T. Niermann and J. H. Patel, "HITEC: A test generation package for sequential circuits," in *European Conference on Design Automation*, 1992, pp. 214–218.

[17] H. K. Lee and D. S. Ha, "HOPE: An efficient parallel fault simulator for synchronous sequential circuits," *IEEE Transactions on Computer-Aided Design of Integrated Circuits and Systems*, vol. 15, pp. 1048–1058, 1996.

[18] F. Corno, M. S. Reorda, and G. Squillero, "RT-level ITC 99 benchmarks and first ATPG results," *IEEE Design and Test of Computers*, vol. 17, no. 3, pp. 44–53, 2000.

[19] J. P. Hayes, *Computer Architecture and Organization*, 3rd ed: McGraw-Hill, 1998.

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

Ismet Bayraktaroglu received the B.S. and M.S. degrees in electrical engineering from Bogazici University, Istanbul, Turkey, in 1994 and 1996, respectively; and the Ph.D. degree in computer engineering from the University of California, San Diego, in 2002. He is currently a member of the technical staff at Sun Microsystems. His research interests include built-in self-test (BIST), diagnosis of BIST designs, test pattern compression, and concurrent test of DSP.

Alex Orailoglu received the S.B. degree (*cum laude*) in applied mathematics from Harvard University, Cambridge, MA; and the M.S. and Ph.D. degrees in computer science from the University of Illinois, Urbana-Champaign. From 1983 to 1987 he was a Senior Member of Technical Staff at Gould Research Laboratories, Rolling Meadows, IL. In 1987 he joined the University of California, San Diego, where he is currently a Professor in the Computer Science and Engineering Department. His research interests include digital and analog test, fault-tolerant computing, computer-aided design, and embedded processors. Prof. Orailoglu is a member of the IEEE Test Technology Technical Council (TTTC) Executive Committee. He serves in numerous technical and organizing committees and he is a Golden Core member of the IEEE Computer Society.