Invariance-Based On-Line Test for RTL Controller-Datapath Circuits[*]

Yiorgos Makris, Ismet Bayraktaroğlu, Alex Orailoğlu
Computer Science and Engineering Department
University of California, San Diego
{makris, ibayrakt, alex}@cs.ucsd.edu

Abstract

We present a low-cost on-line test methodology for RTL controller-datapath pairs, based on the notion of path invariance. The fundamental observation supporting the proposed methodology is that the transparency behavior inherent in RTL components renders rich sources of invariance in a design. Furthermore, the algorithmic controller-datapath interaction provides additional sources of invariance. A judicious selection and combination of modular transparency, based on the algorithm implemented by the controller-datapath pair, yields a powerful set of invariant paths. Such paths enable a simple, yet very efficient on-line test capability, achieving fault security in excess of 90% while keeping the hardware overhead below 40% on complicated, difficult to test, benchmarks.

1. Introduction

The ability to test the functionality of a circuit during normal operation is becoming an increasingly desirable property of modern designs. Identifying and discarding faulty results before they are further utilized constitutes a powerful design attribute. However, such a capability incurs considerable cost in terms of area overhead and possibly performance degradation. Devising a low-cost, non-intrusive on-line test methodology that provides high fault security is therefore a challenging task.

Current state-of-the-art efforts in on-line test research [8] can be roughly categorized along two main directions, as depicted in figure (1). Approaches along the first direction [10, 13] utilize vectors and responses generated off-line, which are stored on-chip, possibly compacted. Whenever one of these vectors appears at the inputs during normal functionality, the response is checked against the expected response. Such approaches require inordinate hardware overhead for storing a sufficient number of test vectors. Their applicability is consequently limited largely to combinational circuits. Approaches along the second direction utilize coarse behavioral invariance either inherent in the design [1, 2] or imposed through error detection codes [11, 14], in order to check the correctness of the functionality. In this case, while the circuit computes function $f(x)$, 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 $f(x)$ and $g(x)$ holds. However, coarse behavioral design invariance is inherently available only in limited domains; furthermore, implementations such as [2] can necessitate appreciable area overhead. Invariably, error detection codes incur high area and performance overhead.

In this paper, we propose an on-line test methodology that employs simple invariant functions distributed across several paths in the design. An overview of the proposed scheme is provided in section 2. The applicability of the inherent transparency behavior of RTL modules for on-line test is examined in section 3. The hardware overhead problem is addressed through transparent path composition in section 4. In section 5 we utilize algorithmic path invariance for enhancing fault security. The latency problem is discussed in section 6. An example of the proposed methodology is given in section 7 and an experimental setup along with results on benchmark circuits is provided in section 8.

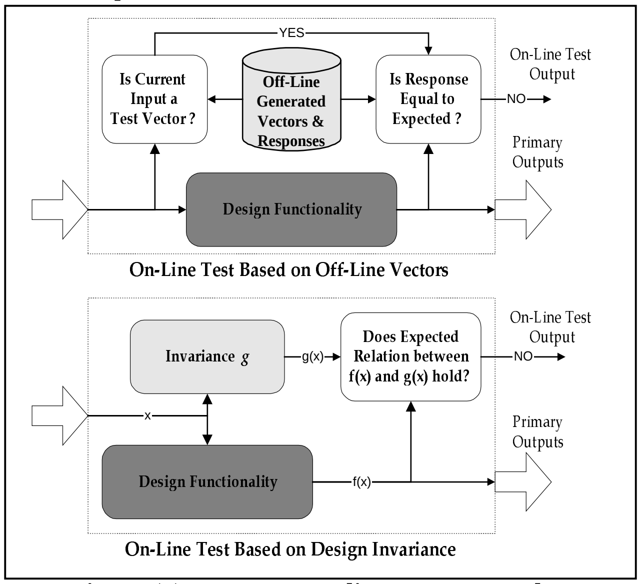

YES
Is Current Input a Test Vector ?
Off-Line Generated Vectors & Responses
Is Response Equal to Expected ?
On-Line Test Output
NO
Primary Outputs
Design Functionality
On-Line Test Based on Off-Line Vectors
Invariance g
g(x)
Does Expected Relation between f(x) and g(x) hold?
On-Line Test Output
NO
x
Primary Outputs
Design Functionality
f(x)
On-Line Test Based on Design Invariance

Figure (1): Common on-line test approaches

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

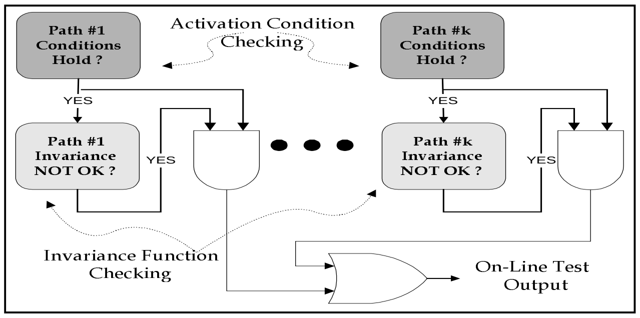

Path #1 Conditions Hold ?
Activation Condition Checking
Path #k Conditions Hold ?
YES
YES
Path #1 Invariance NOT OK ?
YES
Path #k Invariance NOT OK ?
YES
Invariance Function Checking
On-Line Test Output

Figure (2): Proposed methodology overview

2. Methodology overview

An overview of the proposed on-line test methodology is shown in figure (2). Given a controller-datapath pair, several invariant paths are identified based on the transparency behavior and the algorithmic interaction of the RTL modules. The invariant behavior of each path is activated under a set of conditions. Checking the invariant paths during normal circuit functionality requires two elements. The first element is the logic that examines whether the conditions hold, while the second element is the actual invariance checker. In the event that an invariant path is activated but the invariance relationship does not hold, the on-line test output indicates an error.

The distribution of invariance checking throughout the design results in simpler invariant relations and therefore lower area overhead for the checkers, as compared to a global invariance checker. Furthermore, the invariant paths rely on transparency of simple RTL modules, extending the applicability domain of invariance-based on-line test to general RTL designs. However, achieving high fault security with low area overhead requires that a small number of key invariant paths be selected, capable of covering a high number of faults in the design. The identification, evaluation, and selection of these paths are discussed in the following sections.

3. Modular transparency & on-line test

Modular transparency constitutes a key design attribute, enabling several off-line hierarchical test methodologies [5, 7, 12]. Transparency [3, 5] provides a mechanism for traversing a hierarchical design in order to test a module through reachability paths. In this section, we examine the applicability of modular transparency for on-line test, in an effort to integrate off-line and on-line test approaches.

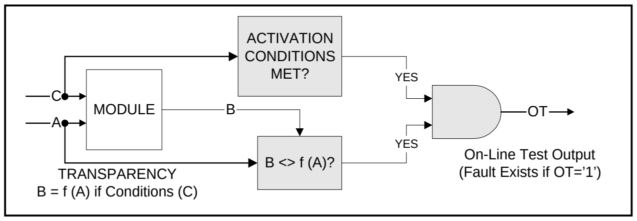

ACTIVATION CONDITIONS MET?
YES
C
A
MODULE
B
OT
B <> f (A)?
YES
On-Line Test Output (Fault Exists if OT='1')
TRANSPARENCY
B = f (A) if Conditions (C)

Figure (3): On-line test via modular transparency

Transparency behavior is defined as *bijective* functions of RTL modules. Such behavior is activated under certain conditions imposed on the inputs of the module in transparency mode. Common transparency functions, such as *Identity* and *Inversion*, establish simple relations between the inputs and the outputs of a module. Checking this relation provides an on-line test capability detecting any fault affecting the transparency, as shown in figure (3).

We now examine the efficiency of this scheme for detecting faults in simple RTL modules. Consider the 8-bit 2-to-1 MUX shown in figure (4)(a), comprising two transparency functions. The proposed on-line test scheme will detect 100% of the faults in the MUX. Additionally, consider a module where the transparency functions do not cover the complete functionality of the circuit. In figure (4)(b) we show an 8-bit SUBTRACTOR, comprising two transparency functions. The corresponding on-line test mechanism results in 80% fault coverage in the SUBTRACTOR, implying that a large number of faults can be detected through a few transparency functions. The idea is readily extensible to sequential modules. In figure (4)(c) we demonstrate the same scheme for an 8-bit REGISTER, where 99% of the faults are detected by the two transparency functions shown. Finally, in figure (4)(d) we show how this idea is applied on a 4-bit LOADABLE COUNTER. While counters cause serious problems to ATPG tools, there exist transparency functions that cover many faults. For example, the two transparency functions in figure (4)(d) cover 88% of the COUNTER faults.

4. Transparency-based path invariance

Incorporating a checking mechanism for each transparency function of every module in the design results in inordinate hardware overhead. In order to reduce

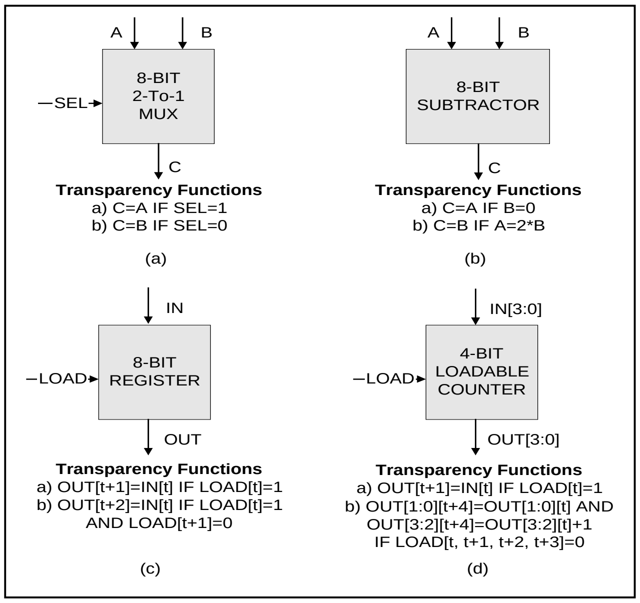
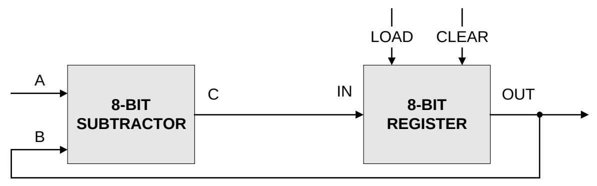


Figure (4): Examples of modular transparency



Subtractor Transparency Functions:
a) C[t] = A[t] IF B[t] = 0
b) C[t] = B[t] IF A[t] = 2*B[t]
c) C[t] = 0 IF A[t] = B[t]

Register Transparency Functions:
a) OUT[t+1] = IN[t] IF LOAD[t] = 1
b) OUT[t+1] = OUT[t] IF LOAD[t] = CLEAR [t] =0
c) OUT[t+1] = 0 IF LOAD[t] = 0 AND CLEAR [t] = 1

Composite Path Transparency Function:
OUT[t+4] = A[t+1] AND C[t+3] = 0 IF CLEAR[t] = 1 AND CLEAR[t+1] = 0 AND LOAD[t+1] = 1 AND CLEAR[t+2] = 0 AND LOAD[t+2] = 1 AND CLEAR[t+3] = 0 AND LOAD[t+3] = 0 AND A[t+2] = 2*A[t+1] AND A[t+1] = A[t+3]

Figure (5): Transparency path composition example

this cost, several modular 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 may 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.

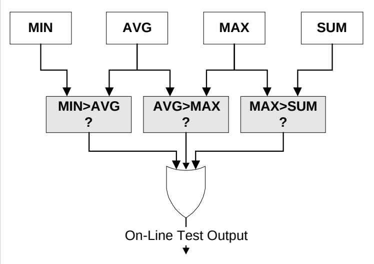
As an example of transparent path composition, consider the circuit shown in figure (5). Three transparency functions are provided for each of the two modules in the design, requiring six distinct checking mechanisms. However, a path spanning 5 clock cycles can be composed, comprising all of the six transparency functions. Although the activation conditions of the six transparency functions still need to be checked, the path transparency function requires only one check instead of six. As a result, path composition significantly reduces the hardware required.

Off-line test path composition has been extensively studied [3, 5, 7, 12] in order to provide transparent reachability paths for hierarchical test. The basic path construction capability is therefore available and can be extended to automate the composition of transparency-based invariant paths. However, the aforementioned off-line path composition approaches need to be tuned to the particularities of on-line test. Ideally, the algorithm should compose a small number of long paths covering all the modular transparency functions, in order to minimize hardware overhead. However, the activation frequency of the path also plays an important role in on-line test, 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, since the path can be fully controlled during off-line test application. In on-line test, however, we rely on normal operation to activate the transparent paths. Therefore, the transparency frequency determined by the complexity of the activation conditions needs to be taken into account in order to balance the number, the cost, and the efficiency of the transparent paths. The hierarchical test path composition algorithm introduced in [6] can be modified in order to address these issues. In order to reduce the cost of the checker for the composite path function, simple modular transparency functions such as identity and inversion are utilized.

5. Algorithmic path invariance

While modular transparency functions and the composite invariant paths are capable of detecting many faults, the resulting fault coverage may not be adequate to ensure high fault security in the design. Additional sources of invariance, capable of checking for faults that 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 will be detected through algorithmic invariance checking. Algorithmic invariance frequently has an equivalence relation to the conditions. In this case we verify not only the transparency existence when the activation conditions hold, but also the lack of transparency when the activation conditions do not hold, thus increasing the number of detected faults. An example of algorithmic invariance and the consequent on-line test scheme is shown in figure (6). The example algorithm keeps track of the minimum, maximum, sum and average of an array of numbers. Several invariant attributes can be identified on this algorithm. For example, during normal circuit operation, the values of the four design registers are related through the inequality $MIN \leq AVG \leq MAX \leq SUM$. Any fault violating this algorithmic invariance will be detected.

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 on-line



Algorithm:

```
SUM:=A[0];
AVG:=A[0];
MIN:=A[0];
MAX:=A[0];
For i:=1 to 99
    Begin
        If A[i]>MAX then MAX:=A[i];
        If A[i]<MIN then MIN:=A[i];
        SUM:=SUM+A[i];
        AVG:=SUM/(i+1);
    End
```

Algorithmic Invariance: MIN<=AVG<=MAX<=SUM

Figure (6): Algorithmic path invariance example

test scheme continuously checks for errors. In case algorithmic invariance exists only on part of the algorithm, relying on a particular control path to be taken, the activation frequency will reflect the execution frequency of the specific part of the algorithm. In short, algorithmic path invariance complements the transparency path invariance in terms of both fault coverage and activation frequency.

6. Invariance activation frequency

Unlike off-line test where faults are detected before the circuit performs its intended operation, fault detection latency, the time interval between fault occurrence and fault detection, is a critical issue during on-line test. The later a fault is detected after its occurrence, the higher the chances that an erroneous result may slip undetected. Latency has not only been a difficult problem for on-line test methodologies to solve, but also hard to estimate. While some on-line test methods [2, 11, 14] guarantee zero-latency, they incur excessive area overhead, thus limiting their applicability. The invariance-based on-line test methodology proposed in [1] reduces the area overhead yet at the cost of introducing latency. Within input monitoring techniques, such as [10, 13], latency is heavily dependent on the values that appear at the inputs of the circuit during normal 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 on-line test. Therefore, invariant path activation frequency is expected to be significantly 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 the activation frequency, invariant paths can be roughly categorized into three types. The first type comprises invariant paths that are always active, such as unconditional algorithmic invariance. This is the most desirable type of invariance since all faults that can be detected through this checking mechanism will have zero latency. The second type 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, since the frequent activation will help keep latency low. Conditional algorithmic invariance is a good example of this type. The third type 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 such invariance checking mechanisms in the design, especially when alternative invariant paths exist.

The activation frequency of an invariant path can be estimated in two different ways. The first method is a static analysis of the controller-datapath interaction. Starting from the final states of the controller, we trace backwards 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, a dynamic *profiling* mechanism is employed to estimate 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.

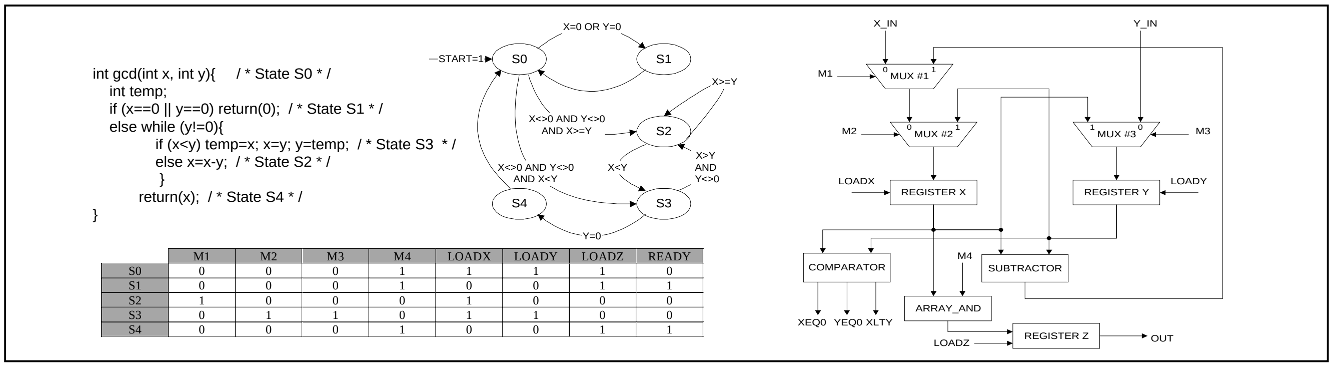
From a latency perspective, a larger number of shorter but more frequently activated invariant paths should be preferred over a few, long but rarely activated invariant paths. The number of invariant paths, their activation frequency, and the number of activation conditions (and therefore the length 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 balance the area overhead, the fault coverage attainable, and the fault detection latency.

7. Example

The proposed methodology is demonstrated through the greatest-common-divisor (GCD) circuit, on which three invariance checks are described. The first check exploits a transparency function of a single module, the second check utilizes a transparency-based invariant path, and the third check uses algorithmic invariance of the design. Figure (7) shows the GCD algorithm and circuit.

The first invariance check example is based on the transparency function *"$C[t]=B[t]$ if $A[t]=2*B[t]$"* of the SUBTRACTOR. The on-line test mechanism is shown in figure (8)(a). A shifter and two comparators suffice for checking the condition and the transparency function. The important attribute of this check is its activation frequency, since it is activated almost once per GCD calculation. The reason for this is that the *subtract-&-swap* GCD algorithm terminates when $y=0$, which implies in turn that $x=y$ two clock cycles earlier. This requires that either the initial values of x and y be equal or that $x=2*y$, which is the activation condition of the SUBTRACTOR transparency. The superior fault coverage, the simplicity of checking and the high frequency of activation make this invariance check a highly efficient on-line test mechanism.

The second example is a transparency-based invariant path. The composite function of this transparent path is *"$OUT[t+4]=X[t]$ if $X[t]=Y[t]$ AND $X[t]\neq 0$"*. Any fault in



```
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; x=y; y=temp;  / * State S3  * /
            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

Figure (7): The GCD benchmark circuit

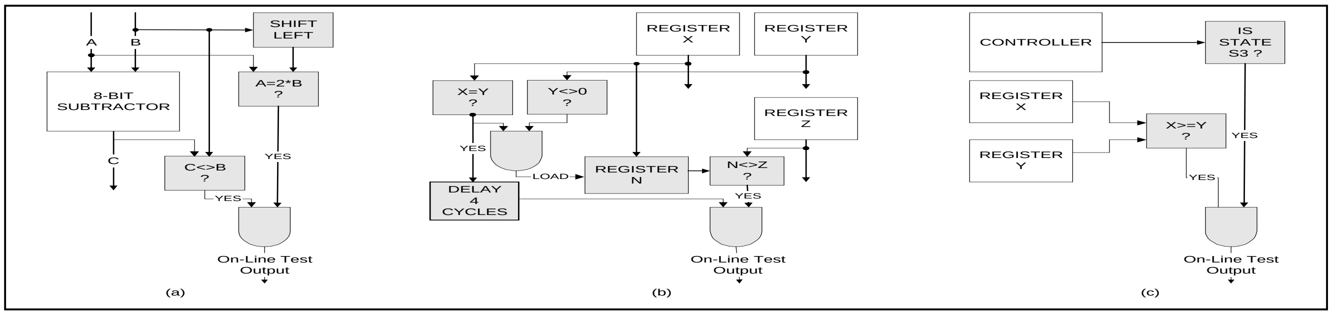


Figure (8): Invariance checking examples on the GCD benchmark

the functions on the path or in the corresponding controller behavior will be detected by the checking mechanism of figure (8)(b). An additional register N is required to store the value of register X whenever the condition “X=Y AND X≠0” holds. The stored value will be compared 4 clock cycles later to the output of the circuit, in order to check the path function *“OUT[t+4]=X[t] if X[t]=Y[t] AND X[t]≠0”*. The hardware cost required is certainly justified by the large number of transparency functions on the path. The activation frequency of this path is also once per GCD calculation, for the same reason as in the first example.

The third example identifies algorithmic invariance in the GCD circuit. The first part of the *if statement* within the *while loop* describes the swapping which is performed in state S3 of the controller, which is reached only when condition $x<y$ holds. Performing a swap while $x \geq y$ would be algorithmically invalid during normal operation of the GCD calculation. As shown in figure (8)(c), this algorithmic invariance provides an on-line test mechanism capable of catching controller faults. The hardware cost is low and the activation frequency high since the algorithmic invariance is always active.

8. Experimental results

The proposed on-line test methodology is evaluated on three difficult-to-test sequential benchmark circuits, using the experimental setup shown in figure (9). The fault security and fault coverage achieved by the identified path invariance, as well as the area overhead imposed by the invariance checking hardware are examined. The first benchmark (GCD) is the greatest-common-divisor circuit that we examined in section 7. The second benchmark (MINMAX) calculates the minimum, maximum and average of a series of numbers. The third benchmark (MUL) is a shift-&-add 8-bit multiplier.

We first apply gate-level ATPG using HITEC [9] in order to obtain the *deterministic off-line test fault coverage* as a reference point. Subsequently, 1000 random inputs are generated and fault simulated for each design using HOPE [4] and the *random off-line test fault coverage* is obtained, along with the set of covered faults. The design is then augmented according to the proposed methodology and the same random inputs are fault simulated. Only the on-line test output is considered a primary output and only the faults covered in off-line random vector fault simulation

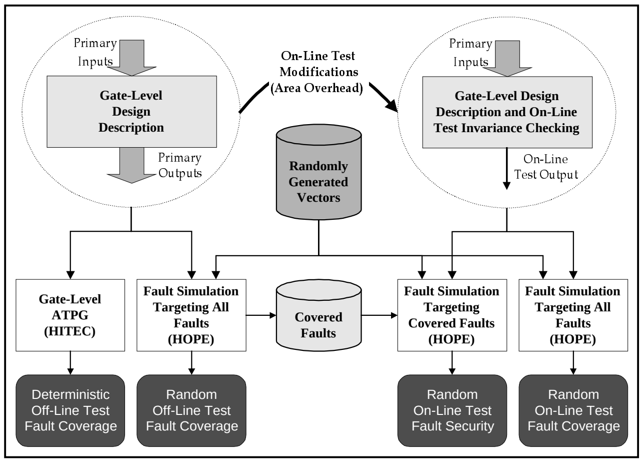

Primary Inputs
Gate-Level Design Description
Primary Outputs
On-Line Test Modifications (Area Overhead)
Primary Inputs
Gate-Level Design Description and On-Line Test Invariance Checking
On-Line Test Output
Randomly Generated Vectors
Gate-Level ATPG (HITEC)
Fault Simulation Targeting All Faults (HOPE)
Covered Faults
Fault Simulation Targeting Covered Faults (HOPE)
Fault Simulation Targeting All Faults (HOPE)
Deterministic Off-Line Test Fault Coverage
Random Off-Line Test Fault Coverage
Random On-Line Test Fault Security
Random On-Line Test Fault Coverage

Figure (9): Experimental validation

Table (1): Experimental results

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

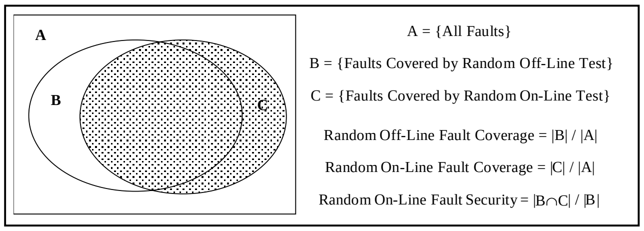

A
B
C
A = {All Faults}
B = {Faults Covered by Random Off-Line Test}
C = {Faults Covered by Random On-Line Test}
Random Off-Line Fault Coverage = |B| / |A|
Random On-Line Fault Coverage = |C| / |A|
Random On-Line Fault Security = |B∩C| / |B|

Figure (10): Fault coverage and fault security

are targeted. The fault coverage achieved indicates the percentage of faults that can be detected both on the original and on the modified design, thus providing the *random on-line test fault security*. The random vectors are also fault simulated in the modified design targeting all faults, in order to obtain the *random on-line test fault coverage*. Figure (10) depicts the faults covered by each experiment and the relations between them. Interestingly, the on-line test may also detect faults that are not detected by off-line test, since checking is performed at internal circuit locations and not just at the primary I/Os.

The results are summarized in Table (1), where the area overhead of the proposed scheme is also shown, assuming only *2*-input gates and an equivalent of four gates for each flip-flop. To reduce the cost, the hardware used for the transparency functions is combined and optimized and only one output pin is used. The results demonstrate that the proposed on-line test methodology achieves fault security exceeding 90%, while keeping the area overhead below 40%. In addition, the achieved *random on-line test fault coverage* is only 6% worse than *random off-line test fault coverage*. It is important to note that a large portion of the undetected faults is due to primary I/O faults that no invariance-based, on-line test methodology can detect.

9. Conclusion

A novel on-line test methodology for RTL controller-datapath pairs is presented in this paper. Based on the modular transparency of the datapath RTL components and the algorithmic behavior imposed by the controller, invariant paths are identified in the design. The capacity of these invariant paths for on-line test is evaluated in terms of fault security, activation frequency, and area overhead, and a set of appropriate paths is selected. The compliance of these paths to the invariant behavior is checked whenever the latter is activated, thus providing an efficient on-line test capability. By exploiting fine-grained design invariance, the proposed methodology contributes a novel on-line test direction, applicable in general RTL circuits.

References

[1] I. Bayraktaroğlu, A. Orailoğlu, "Low-Cost On-Line Test for Digital Filters", *VTS*, pp. 446-451, 1999.

[2] A. Chatterjee, R. K. Roy, "Concurrent Error Detection in Non-Linear Digital Circuits with Applications to Adaptive Filters", *IEEE ICCD*, pp. 606-609, 1993.

[3] S. Freeman, "Test Generation for Data-Path Logic: The F-Path Method", *IEEE JSSC*, vol. 23, no. 2, pp. 421-427, 1988.

[4] H. K. Lee, D. S. Ha, "HOPE: An Efficient Parallel Fault Simulator for Synchronous Sequential Circuits", *IEEE TCAD*, vol. 15, no. 9, pp. 1048-1058, 1996.

[5] Y. Makris, A. Orailoğlu, "DFT Guidance Through RTL Test Justification and Propagation Analysis", *ITC*, pp. 668-677, 1998.

[6] Y. Makris, A. Orailoğlu, "RTL Test Justification and Propagation Analysis for Modular Designs", *JETTA*, vol. 13, no. 2, pp. 105-120, 1998.

[7] B. T. Murray, J. P. Hayes, "Hierarchical Test Generation Using Precomputed Tests for Modules", *IEEE TCAD*, vol. 9, no. 6, pp. 594-603, 1990.

[8] M. Nicolaidis, Y. Zorian, "On-Line Testing for VLSI - A Compendium of Approaches", *JETTA*, vol. 12, no. 1-2, pp. 7-20, 1998.

[9] T. Niermann, J. H. Patel, "HITEC: A Test Generation Package for Sequential Circuits", *EDAC*, pp. 214-218, 1992.

[10] K. K. Saluja, R. Sharma, C. R. Kime, "A Concurrent Testing Technique for Digital Circuits", *IEEE TCAD*, vol. 7, no. 12, pp. 1250-1260, 1988.

[11] C. Stroud, M. Ding, S. Seshadri, I. Kim, S. Roy, S. Wu, R. Karri, "A Parametrized VHDL Library for On-Line Testing", *ITC*, pp. 479-488, 1997.

[12] P. Vishakantaiah, J. A. Abraham, D. G. Saab, "CHEETA: Composition of Hierarchical Sequential Tests using ATKET", *ITC*, pp. 606-615, 1993.

[13] I. Voyiatzis, A. Paschalis, D. Nikolos, C. Halatsis, "R-CBIST: An Effective RAM-Based Input Vector Monitoring Concurrent BIST Technique", *ITC*, pp. 918-925, 1998.

[14] C. Zeng, N. Saxena, E. J. McCluskey, "Finite State Machine Synthesis with Concurrent Error Detection", *ITC*, pp. 672-679, 1999.