An Alternative to Scan Design Methods for Sequential Machines

KEWAL K. SALUJA AND RAMASWAMI DANDAPANI

Abstract — **The problem of testing sequential machines using a checking experiment is investigated. An algorithm is given to augment sequential machines by adding extra input(s) to make them testable. We also present a circuit modification method, similar to scan methods, such that the augmented machine can be tested by the checking experiment. A justification of our method for a VLSI environment is given by determining the overheads.**

Index Terms — **Built-in self test, checking experiments, scan design, sequential machines, testable design.**

I. Introduction

Most of the methods proposed to test digital circuits rely on achieving a separation between sequential and combinational parts of a digital circuit using scan design techniques [1] and testing the combinational portion by different methods [2]–[4]. In some cases, circuits can also be tested functionally [5].

Alternatively, sequential circuits can be tested using checking experiments [6]. This approach has been gaining ground, for in this approach 1) testing is almost independent of the implementation and fault model, and 2) compacting techniques can be used on output sequences, provided a machine is appropriately modified [7]–[10]. A method was given in [11] to modify a single-output sequential machine and order the checking experiment so that only one observation was sufficient for testing. Results with respect to multiple-output sequential machines are given in [12].

In this correspondence, we present a method of modifying a multiple-output sequential machine by adding extra input(s). A machine so modified can be tested by a checking experiment and determined to be faulty or fault free by observing one output value only. We then present a uniform design approach to convert a given sequential circuit into a checking experiment testable circuit and we discuss the necessary hardware overhead. Section II contains notation and Section III presents the augmentation algorithm and formal results in brief. These sections are included here for completeness of this correspondence. Details of the results can be found in [12] and [13]. Section IV contains a circuit modification technique to arrive at testable realization.

II. Motivation and Notation

Checking experiments for a machine which has a synchronizing sequence (SS) and a distinguishing sequence (DS) can be constructed using these and transition checking sequences. Fujiwara *et al.* [7] and Pradhan [8] proposed the use of extra inputs to simplify testing by augmenting a machine so that it contains SS and DS. Venkatraman and Saluja [9] proposed the use of a transition counter (TC) as a data compactor (DC) to compact the output of the machine under test. Hassan [10] uses a linear feedback shift register (LFSR) as a DC. Unfortunately, these methods do not extend to multiple-output machines in a straightforward manner. Also, in these methods, a multitude of values of compacted data need to be observed, thus making the methods, in their present form, unsuitable for built-in self testing (BIST). In the next section, we give a summary of the results from [12] and [13] which overcome these problems. We must point out that the length of the test sequence in the case of checking experiments can be substantially more than the test sequences generated for a specific implementation of a machine. Yet, with the use of data compression, the overall tesing time may be reduced if a checking experiment can be conducted at system clock. This is so because conventional testing is done at tester (external) speed which can be two to three orders of magnitude slower than the machine (internal) clock [14].

Let G be a machine with n states, m inputs, l outputs, and let the size of input and output alphabets be m_i and l_o, respectively. Let δ and λ denote next state and output functions. Let us assume that G has a DS. Clearly, on application of a DS, the start state of G can be uniquely determined by observing the output. Consider the compaction of the output sequence through a compaction function F. If the DS is such that the start state of G can be uniquely determined by observing the output sequence compacted by F, then such a DS is called an FDS. A conventional checking experiment (CE) consists of applications of SS, DS, and transfer sequences (TS). An F-CE is a CE in which, instead of DS, FDS is used. Three compaction functions which are used for FDS in this paper are transition counter (TC) [9], [15], LFSR [10], and syndrome (SYN) [16]. Formal definitions of these can be found in [13].

III. Testable Design — Augmentation Algorithm and Formal Results

In this section, we present an algorithm to modify a sequential machine G to G^* using an additional input ε such that G^* has FDS. To each output of G^* is connected a verifier which contains F registers to store compacted data, an *RR* register to generate reference data, etc., as shown in Fig. 1(a). The structure of testing G^* is shown in Fig. 1(b). States of G^* can be determined by observing the contents of F registers. For simplicity of presentation, we shall assume that all outputs of G^* use identical compaction functions, i.e., $F_0 = F_1 = \cdots = F_{l-1} = F$. Also, $l < \lceil \log_2 n \rceil$ and $n = q^l$ for some integer $q > 2$. If states of G or G^* are represented by integers, then with the above assumption, each state i can be uniquely represented by an l triple $(i_{l-1}, i_{l-2}, \cdots, i_0)$, $0 \le i_j < q$. Also, the output function λ can be represented as $(\lambda^{l-1}, \lambda^{l-2}, \cdots, \lambda^0)$. Thus, λ^i refers to the output function realized on the ith output lead.

Definition: Let i be a positive integer and ε a binary variable. Define functions β_1, β_2, and β_3 as $\beta_1(i, \varepsilon) = \beta_3(i, \varepsilon) = i \pmod 2$; $\beta_3(i, \varepsilon) = 1$ for $i > 0$ and 0 for $i = 0$.

If ε_k denotes a sequence of k ε's, then $\beta(i, \varepsilon_k)$ is interpreted recursively as $\beta(i, \varepsilon)\beta(\delta(i), \varepsilon_{k-1})$.

Algorithm 1: Augument G to G^* using an extra input ε.

Step 1: Select states s_i and s_j in G such that there is an input I_k and $\delta(s_j, I_k) = s_i$. Name the state $s_i = q^l - 1$ and $s_j = 0$. The remaining states are named from $q^l - 2$ to 1 arbitrarily.

Step 2: Define δ for the additional input ε as follows:

$$\delta((i_{l-1}, i_{l-2}, \cdots, i_1, i_0)\varepsilon) = (D(i_{l-2}), D(i_{l-2}), \cdots, D(i_0))$$

where $D(j) = j - 1$ for $j > 0$ and 0 for $j = 0$.

Step 3: Specify $\lambda(i, \varepsilon) = (\lambda^{l-1}(i_{l-1}, \varepsilon), \lambda^{l-2}(i_{l-2}, \varepsilon), \cdots, \lambda^0(i_0, \varepsilon))$ where each λ^k can be chosen from $\beta_1, \beta_2, \beta_3$. For inputs other than ε, λ and δ for G^* remain the same as for G.

Example: For a machine G with $n = 9, l = 2$, the next states and three possible output functions which result due to Steps 2 and 3 above are given in Table I.

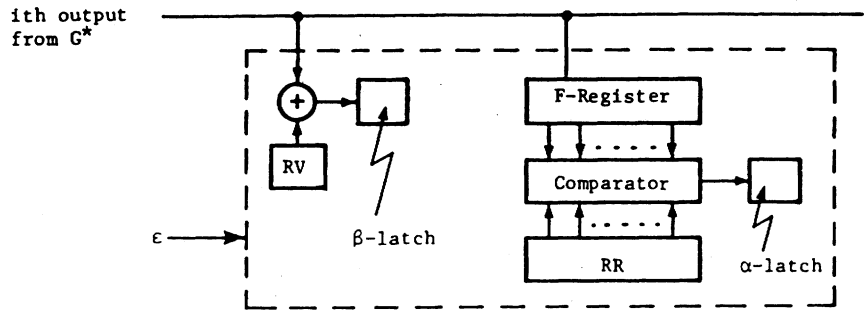
Augmentation algorithms presented in [9]–[11] are special cases of Algorithm 1 above. One can formally prove some properties of G^*, but we shall state only the final results due to space limitation. Proofs of these results can be found in [13].

Manuscript received November 27, 1985; revised December 19, 1985. This work was supported in part by ARCB and ARGS Grants, Australia.

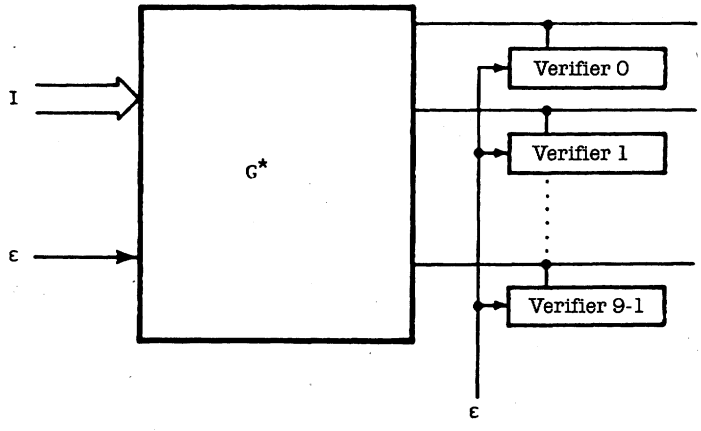
K. K. Saluja was with the Department of Electrical and Computer Engineering, University of Newcastle, Newcastle, N.S.W. 2308, Australia. He is now with the Department of Electrical and Computer Engineering, University of Wisconsin, Madison, WI 53706.

R. Dandapani is with the Department of Mathematics and Computer Science, Youngstown State University, Youngstown, OH 44555.

IEEE Log Number 8607593.



(a)



(b)

Fig. 1. Modified machine G^* and its verifier. (a) Verifier on ith output. (b) Structure for testing G^*.

TABLE I
NEXT STATES AND OUTPUTS FOR AN EXTRA INPUT ε FOR A MACHINE WITH $n = 9, l = 2$, AND $q = 3$

PS		NS for input $= \varepsilon$		Outputs $\lambda = \lambda^1, \lambda^0$		
i	(i_1, i_0)	i	(i_1, i_0)	$\lambda_1 = \beta_1, \beta_1$	$\lambda_2 = \beta_2, \beta_2$	$\lambda_3 = \beta_3, \beta_3$
8	(2,2)	4	(1,1)	0,0	1,1	1,1
7	(2,1)	3	(1,0)	0,1	1,1	1,1
6	(2,0)	3	(1,0)	0,0	1,0	1,0
5	(1,2)	1	(0,1)	1,0	1,1	1,1
4	(1,1)	0	(0,0)	1,1	1,1	1,1
3	(1,0)	0	(0,0)	1,0	1,0	1,0
2	(0,2)	1	(0,1)	0,0	0,1	0,1
1	(0,1)	0	(0,0)	0,1	0,1	0,1
0	(0,0)	0	(0,0)	0,0	0,0	0,0

Theorem 1: If G is a single-output machine, then G^* is strongly connected. If G is a multiple-output machine, then G^* may or may not be strongly connected.

An algorithm to generate an ordered-F-CE (O-F-CE) to test G satisfying the conditions

$$\text{R1): } l < \lceil \log_2 n \rceil, \qquad \text{R2): } n = q^l,$$

$$\text{and} \quad \text{R3): strongly connected}$$

is given below:

Algorithm 2: Generation of O-F-CE to test G.

Step 1: Apply Algorithm 1 to G. The resulting machine G^* has an F-CE.

Step 2: Construct l_0 predecessor/successor tables [PST-0 to PST-($l_0 - 1$)] [12]. These tables contain the inputs needed to transfer G^* from a given state to another with a specific output value.

Step 3: Set O-F-CE to SS.

Step 4: Add the pairs (sequence to transfer to state i from state 0, FDS) for $i = 0, 1, \cdots, n-1$ *in order* to O-F-CE.

Step 5: For $i = 0, \cdots, n-1$, do the following.

Repeat the substep below for $k = 0, 1, \cdots, l_0 - 1$.

Substep: If column i of PST-k has x in row j, then add the triple (sequence to transfer to state j, x, FDS) to O-F-CE. Repeat for all rows. The order in which rows are chosen is not important. □

The proof of this algorithm can be found in [13].

Theorem 2: A multiple-output machine satisfying R1), R2), and R3) can be augmented by using Algorithm 1 such that the augmented machine can be determined to be faulty or fault free by observing only one value at the end of a checking experiment.

In fact, conditions R1) and R2) are imposed on G for the notational simplicity of the proof of Algorithm 2 in [13]. Thus, these

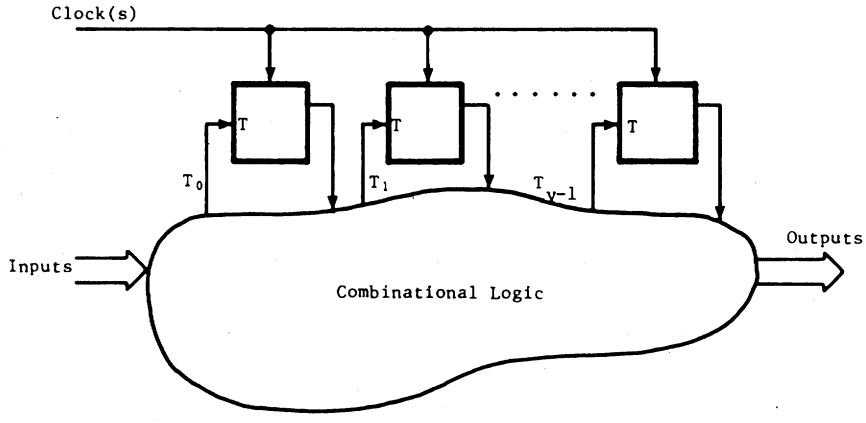


Fig. 2. Structure of the original machine.

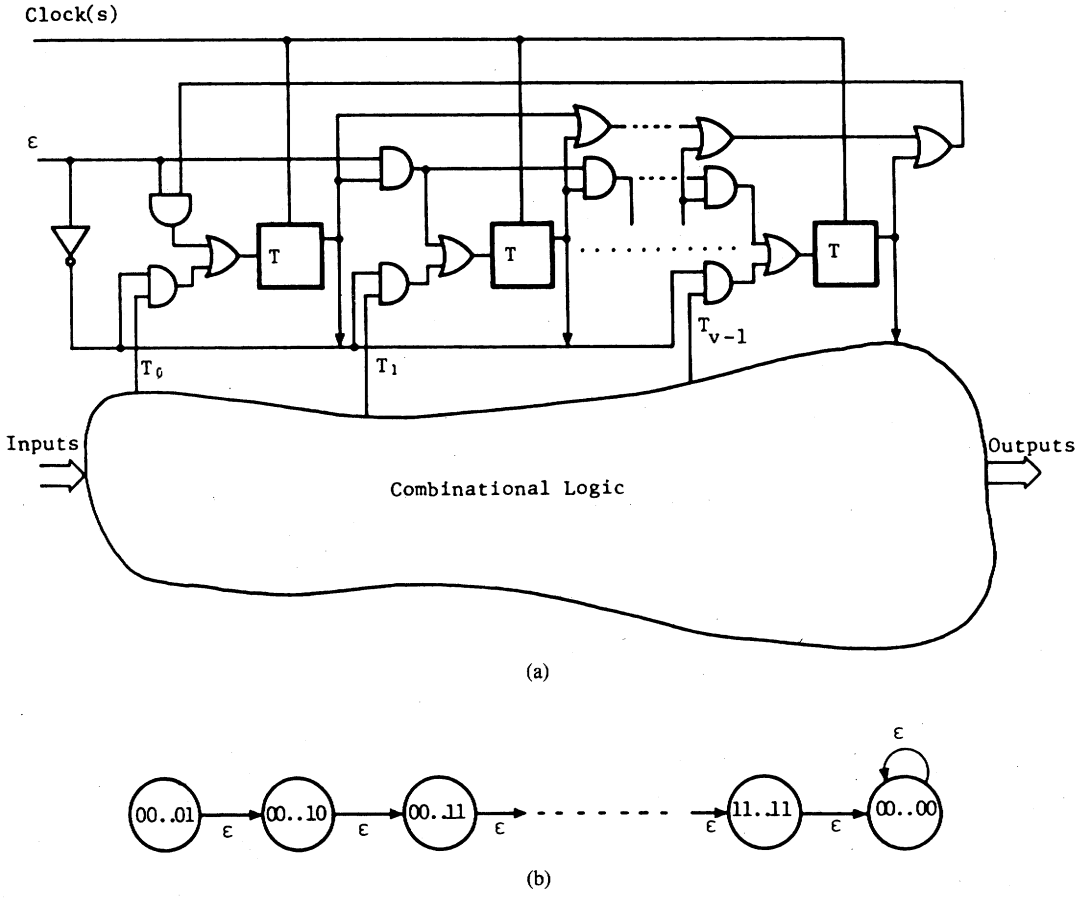


Fig. 3. An implementation of augmentation algorithm. (a) Modified circuit. (b) State diagram of augmented part.

two restrictions can be removed without invalidating Theorem 2. Removal of R3) may require extra input(s) to make G^* strongly connected. Using an approach similar to the one for single-output machines, a multiple-output machine can be made strongly connected by using an extra input ε_1 (in addition to ε). This will also guarantee transitions from any state to any other state with transfer sequences of length n or less [11]. The length of transfer sequences can be reduced even further by using more than one extra input [13].

Thus, we can state the following theorem.

Theorem 3: Any sequential machine can be augmented by using one or more extra inputs in such a way that the augmented machine can be determined to be faulty or fault free by observing only one value at the end of a checking experiment.

IV. Implementation Details

In Section III, we gave an algorithm to augment a machine for testability. A knowledge of the state table was assumed therein. We now present a systematic way of modifying a sequential circuit such that the modified circuit is a realization of the augmented machine. For the application of this method of modification, a knowledge of the state table is not necessary, although for the complete design of the checking experiment, a knowledge of the state table is required.

For simplicity of presentation, we consider only a special case of Algorithm 1 with $l = 1$, i.e., $\delta(i, \varepsilon) = i - 1$ for $i > 0$ and 0 for $i = 0$.

Although the state assignment is now known to us, for notational simplicity we assume that states 0 and $n - 1$ are $(0, 0 \cdots, 0, 0)$ and $(0, 0, \cdots, 0, 1)$, respectively. As all the remaining states are chosen arbitrarily (Algorithm 1), we can assume the assignment shown below for the remaining states without any loss of generality.

State	Assignment
$n - 2$	$(0, 0, \cdots, 1, 0)$
$n - 3$	$(0, 0, \cdots, 1, 1)$
⋮	⋮
1	$(1, 1, \cdots, 1, 1)$

We further assume that the original machine has been realized using T-type latches/flip-flops. Thus, the structure of the original machine is as shown in Fig. 2. Note that in this realization, no regularity of structure has been assumed.

We now modify the circuit of Fig. 2 and obtain the circuit of Fig. 3(a) by making use of an extra input ε. The augmented part of the machine in the form of a state diagram resulting from the above modification of the circuit is shown in Fig. 3(b). Modification in the output circuit depends on the choice of output compactor, and therefore on the choice of λ. For example, for λ_2 and λ_3 functions, only three extra gates are required in the output logic. Thus, not including output logic, the overhead is $4\nu - 1$ two-input gates where ν ($= \log_2 n$) is the number of flip-flops (latches) in the original machine. Notice that this gate count is based on the assumption that the original machine is realized using T-type latches/flip-flops. If the original machine used D-type latches/flip-flops, this gate count will increase only to the extent of converting D-type devices to T-type devices.

It is difficult to determine the percentage overhead for this method as it will depend on the size of the original machine. By assuming a particular realization and making some simple assumptions, one can arrive at figures about percentage overhead (see, for example, [10] for overhead assuming PLA realization). However, such figures can be deceptive, for PLA's are often very sparse, and further, PLA's can be designed for testability [17] with yet different and efficient methods.

We believe that the checking experiment approach will be most suitable for small machines, e.g., for verification of control unit of a microprocessor/microcomputer. Such parts are often realized as random logic to meet the necessary speed requirements. To enhance testability of such parts, designers often resort to scan design methods. The method proposed here has a remarkable similarity to scan design methods. Yet, with our method, test generation and test verification phases can largely be eliminated, although the test generation phase is not eliminated completely for derivation of the checking experiment requires construction of a state table. Although the length of the checking experiment may be substantially longer than a generated test sequence, it offers the following major advantages over other methods.

1) The test sequence is independent of the implementation of the machine; thus, the circuit design and layout do not affect the test sequence.

2) The test sequence is independent of the fault model. In a VLSI environment, this may be a major advantage because most of the existing fault models have been found to be either unsuitable or inadequate.

3) In a sense, checking experiments are analogous to exhaustive testing/verification testing [2] of combinational logic circuits. Thus, it offers all the advantages of exhaustive testing schemes.

We conclude this section with a simple example. Let us consider a control unit of a microprocessor having as many as ten flip-flops. Although such a control unit will not have 2^{10} states, yet from the checking experiment point of view, it must be tested for $n = 2^{10}$ states. Let us further assume the size of input alphabet $m_i = 32$ and $l = 1$ for the application of the checking experiment. Then

$$\text{test length [11]} = 4n^2 + \tfrac{3}{2}m_i n^2 + n \simeq 52 \times 10^6.$$

As the circuit can be tested at machine clock, for a 10 MHz microprocessor IC, it will take only 5.2 s to verify the control unit.

V. Conclusions

In this paper, we studied the testing of sequential machines using checking experiments. An algorithm was given to augment the state table of a machine by the addition of extra input(s) to make it testable. The test sequences proposed in the paper require that only one observation be made to decide whether a fault was present in the machine. This implies that the whole experiment can run at internal clock frequency as opposed to test clock frequency.

We presented a circuit modification method to obtain an augmented machine. This method does not require the knowledge of a state table and has remarkable similarity to the scan methods. The generation of the test sequence and modification method are quite algorithmic and can be automated.

We believe that the method proposed in this paper can be used to test certain crucial parts of a computer such as a control unit. With the checking experiment approach suggested in this paper, the function of such a unit can be verified and fault-free operation guaranteed. Further, such a unit, after verification, can form a "hard core" for testing of the remaining units in a digital system.

Acknowledgment

The authors are thankful to Prof. C. R. Kime of the University of Wisconsin, Madison, and Dr. J. E. Smith of Madison for many helpful discussions.

References

[1] T. W. Williams and K. P. Parker, "Design for testability — A survey," *Proc. IEEE*, vol. 71, pp. 98–112, Jan. 1983.

[2] E. J. McCluskey, "Built-in verification test," in *Dig. 1982 Int. Test Conf.*, Philadelphia, PA, Oct. 1982, pp. 1983–190.

[3] J. Losq, "Referenceless random testing," in *Dig. Int. Fault Tolerant Comput. — 6*, Pittsburgh, PA, June 1976, pp. 108–113.

[4] R. Dandapani, J. H. Patel, and J. A. Abraham, "Design of generators for built-in test," in *Dig. 1984 Int. Test Conf.*, Philadelphia, PA, Oct. 1982.

[5] J. A. Abraham, "Functional level test generation for complex digital system," in *Dig. IEEE Int. Test Conf.*, 1981, pp. 461–462.

[6] F. C. Hennie, "Fault detecting experiments for sequential circuits," in *Proc. 5th Annu. Symp. Switching Circuit Theory and Logical Design*, Princeton, NJ, Nov. 1964.

[7] H. Fujiwara, Y. Nagao, T. Sasao, and K. Kinoshita, "Easily testable sequential machines with extra inputs," *IEEE Trans. Comput.*, vol. C-24, pp. 821–826, Aug. 1975.

[8] D. K. Pradhan, "Sequential network design using extra inputs for fault detection," *IEEE Trans. Comput.*, vol. C-32, pp. 319–323, Mar. 1983.

[9] C. S. Venkatraman and K. K. Saluja, "Transition count testing of sequential machines," in *Dig. 10th Symp. Fault Tolerant Comput.*, Kyoto, Japan, Oct. 1980, pp. 167–172.

[10] S. Z. Hassan, "Signature testing of sequential machines," *IEEE Trans. Comput.*, vol. C-33, pp. 762–764, Aug. 1984.

[11] K. K. Saluja and R. Dandapani, "Testable design of sequential machines using compaction functions and checking experiments," in *Dig. 22nd Annu. Allerton Conf.*, IL, Oct. 1984, pp. 827–836.

[12] ——, "Generalized compaction function testable design of sequential machines using checking experiments," in *Dig. 15th Int. Symp. Fault Tolerant Comput.*, June 1985, pp. 613–620.
[13] ——, "An alternative to scan design methods for sequential machines," Dep. Elec. and Comput. Eng., Univ. Newcastle, Australia, Tech. Rep. EE8545.
[14] T. W. William, "VLSI testing," *IEEE Computer,* vol. 17, pp. 126–136, Oct. 1984.
[15] J. P. Hayes, "Transition counting testing of combinational logic circuits," *IEEE Trans. Comput.*, vol. C-25, pp. 613–620, June 1976.
[16] J. Savir, "Syndrome testable design of combinational circuits," *IEEE Trans. Comput.*, vol. C-29, pp. 442–451, June 1980, and vol. C-29, pp. 1012–1013, Nov. 1980.
[17] K. K. Saluja, K. Kinoshita, and H. Fujiwara, "An easily testable design of programmable logic arrays for multiple faults," *IEEE Trans. Comput.*, vol. C-32, pp. 1038–1046, Nov. 1983.