RESEARCH ARTICLE

inventi Spreading Knowledge

Adopting Systematic Quality Signing and Verification Processes for Sequential Hardware Testing

Mohammed I Younis[1*], Kamal Z Zamli[2], Nor Ashidi Mat Isa[2]

Abstracts: This paper discusses an improvement of the strategy for Quality Signing and Verification Processes. Earlier studies demonstrate that the strategy relies on two processes: Quality Signing Process and Quality Verification Process, respectively. The Quality Signing Process is based on integration of black box (i.e. Combinatorial Interaction Testing) and white box (i.e. Mutation Testing) techniques in order to derive an optimum test suite for the Quality Verification Process. In this case, the generated test suite significantly improves the Quality Verification Process. Unlike the previous work, which deals only with combinatorial logic, an improvement of the strategy now addresses sequential logic, that is, by incorporating both the state of the system as well as the input parameter values as input in both processes. As a case study, this paper describes the step-by-step application of the strategy for testing a 12-bit Linear Feedback Shift Register in a hardware production line. The result demonstrates that the proposed strategy gives saving effort factor of 99.7%. Additionally, the result also demonstrates the need to consider Combinatorial Interaction Testing in both cumulative and normal mode of operation.

Key Words: Combinatorial interaction testing; t-way testing; multi-way testing; mutation testing, software testing; hardware testing.

INTRODUCTION

Combinatorial interaction testing (CIT) has been an active research area in the recent years. CIT [1-7] is based on generating subset of exhaustive set according to the degree of interaction (t) amongst parameter values such that each combination of t parameter values is covered at least one. As such, CIT is considered as a black box technique. In contrast, mutation testing is a white box testing, that is, depends on injection faults in the *system under test* (SUT) to determine the quality of the test suite, such that, the desired test suite is found that detects all faults already have been injected in SUT. Each strategy has its wide spectrum of applications in both hardware and software domains [8-19].

Younis and Zamli proposed a novel strategy that combines both CIT and mutation testing in order to derive optimized test suite for combinational ICs testing [20]. Herewith, the concept of the saving efforts factor is introduced as an evaluation factor. In this case, the saving efforts factor is defined as the size of exhaustive test suite minus optimized test suite to the size of exhaustive test suite. Empirical results demonstrated the effectiveness of such approach into a 4-bits magnitude comparator IC, where saving efforts factor of 256-9/256=96.48% has been obtained. Given such a result, the research question now is that can the strategy be extended and improve such a strategy for testing sequential circuits. The challenge here is that unlike combinatorial circuits, sequential circuits have the notion of states and memory.

In order to understand how improvement can be made to support sequential circuits, there is a need to revisit the aforementioned strategy. Briefly, the strategy consists for two processes, namely: *Test Quality Signing* (TQS) process and *Test Verification* process (TV). Briefly, the TQS process deals with optimizing the selection of test suite; obtained from CIT, for fault injection as well as performs the actual injection whilst the TV process analyzes for conformance as illustrated in Figure 1.

Complementing and building from the earlier work, this paper suggests to modify the both the TQS and TV processes to assist sequential circuit testing. Unlike combinational circuit testing, the sequential circuit testing considers another sub-research question: Is there a need to test all states of interaction? Answering these questions is the focus of this paper.

The rest of this paper is organized as follows. Section 2 discusses the modification for both TQS and TV processes. Section 3 gives a step-by-step example as prove of concept involving the 12-bit LFSR. Section 4 demonstrates the requirements for putting CIT in cumulative mode. Finally, Section 5 describes the conclusion and suggestion for future work.

Generalized the Quality Signing and Verification Processes

This section describes the generalization of the TQS and TV processes to include the sequential circuit testing.

The TQS process aims to derive an effective and optimum test suite (includes the sequential circuit testing) and works as follows.

- Start with an empty *Optimized Test Suite* (OTS), and empty *Signing Vector* (SV).
- Select the desired software class (for software testing). Alternatively, build an equivalent software class for the *Circuit Under Test* (CUT) (for hardware testing).
- Store these faults in *fault list* (FL).
- Inject the class with all possible faults.
- Let N be maximum number of parameters.
- Initialize CIT strategy with strength of coverage (t) equal one (i.e., $t = 1$).
- Let CIT strategy partition the exhaustive test space. The portioning involves generating one test case at a time for t coverage. If t coverage criteria are satisfied, then $t = t + 1$. Here, the system parameter values are considered both the input and the state of the system, that is, the initial (previous) state of the system in the case of sequential circuit.
- CIT strategy generates one *Test Case* (TC). Here, the test case is a combination of the input or previous state or both.
- Execute TC. The execution involves:
  - Load the system to initial state from TC part including the state field.
  - Apply the input from TC part including the input field.
  - Record the output(s).
- If TC detects any fault in FL, remove the detected fault(s) from FL, and add TC and its specification output(s) to OTS and SV, respectively.
- If FL is not empty or $t <= N$, go to 7.
- The desired optimized test suite and its corresponding output(s) are stored in OTS and SV, respectively.

The TV process involves the verification of fault free for each unit. The TV process for a single unit works as follows.

- for $i = 1..$Size(OTS) each TC in OTS do:
  - Subject the SUT to TC[i] (involve: loading the state in the case of sequential circuit, followed by applying the inputs), store the output in *Verification Vector* VV[i].
  - If VV[i] = SV [i], continue. Else, go to 3.
- Report that the CUT has been passing in the test. Go to 4.
- Report that the CUT has failed the test.
- The verification process ends.

Note that the highlighted steps (7-9) and the first step in signing and verification process are addressed loading the state of the system. Hence, the new version is addressing the sequential circuit issue. By removing these highlighted lines the signing and verification processes are degraded to the previous version in [20] that is limited to software and combinational circuits. In short, the new version involves two folds. First, it deals with the states of the SUT as well as the input parameter values. Thus, the input for CIT is considered the combination of both. Second, the TQS and TV processes involve the load of the state of SUT before faults detection (in the case of TQS process) and determining the outputs (in the case of TV process).

As noted in the second step of the TQS process, the rationale for taking equivalent software class for the CUT is to ensure that the cost and control of the fault injection be more practical and manageable as opposed to performing it directly to a real hardware circuit. Furthermore, the derivation of OTS is

[1]Computer Engineering Department, College of Engineering, University of Baghdad, Baghdad, Iraq.
E-mail: younismi@gmail.com
*Corresponding author

[2]School of Electrical and Electronics Engineering, Universiti Sains Malaysia, Penang, Malaysia.

Table 1: Derivation of OTS for 12-bit LFSR

t=	Cumulative Test Size	Live Mutant	Killed Mutant	% Mutant Score	Effective test size
1	2	22	52	70.27	2
2	11	17	57	77.03	4
3	34	10	64	86.49	7
4	73	6	68	91.89	8
5	142	6	68	91.89	8
6	317	2	72	97.30	10
7	647	1	73	98.65	11
8	811	0	74	100.00	12

Table 2: OTS and SV for the 12-bit LFSR

#TC	OTS TC (I1..I12)	SV Outputs(S1..S12)	Accumulative faults detected /74
1	FFFFFFFFFFFF	TFFFFFFFFFFF	38
2	TTTTTTTTTTTT	FTTTTTTTTTTT	52
3	TFTFTFTFTFTF	TTFTFTFTFTFT	55
4	TFTTFTFFTTFF	FTFTTFTFFTTF	57
5	TFTFFTTFFTTF	FTFTFFTTFFTT	59
6	FFFFTTTTFFTT	FFFFFTTTTFFT	61
7	FFTFTTFTTFFF	FFFTFTTFTTFF	64
8	FFFFFFTTFFFF	FFFFFFFTTFFF	68
9	FFTFFFTFFFFF	FFFTFFFTFFFF	70
10	FTFFTFFFFFFF	FFTFFTFFFFFF	72
11	TTFFFFFFFFFF	FTTFFFFFFFFF	73
12	TFTFFFFFFFFF	FTFTFFFFFFFF	74

Table 3: Derivation of OTS for 12-bit LFSR with CIT in Normal Mode

t=	Test Size	Live Mutant	Killed Mutant	% Mutant Score	Effective test size
1	2	22	52	70.27	2
2	10	17	57	77.03	5
3	21	11	63	85.14	8
4	47	6	68	91.89	9
5	110	6	68	91.89	9
6	225	4	70	94.60	9
7	449	2	72	97.30	10
8	825	1	73	98.65	11
9	1398	0	74	100.00	14
10	1951	0	74	100.00	15
11	2048	0	74	100.00	17
12	4096	0	74	100.00	20

faster in software than in hardware. Despite using equivalent class for the CUT, this verification process should work for both software and hardware systems including the sequential circuit.

Case Study

As a proof of concept, MC_MIPOG [7] has adopted as the CIT strategy implementation, and MuJava version 3 (described in [21, 22]) as a fault injection strategy implementation.

Using both tools (i.e., MC_MIPOG and MuJava), a case study problem involving a 12-bit *linear feedback shift registe* (LFSR) will be discussed here in order to evaluate the proposed generalization of the Quality Signing and Verification Processes discussed in section 2.

A 12-bit LFSR has 12 asynchronous inputs (to load the initial state, namely: I1..I12, enabled by "load" command), 12 D-type flip-flops connected as sequence generator (in this case, as an implementation of the polynomial: $x^{12} + x^{11} + x^{10} + x^4 + 1$), and finally, a zero avoidance circuit is "ORed" with the tapping polynomial as an input to the first flip-flop as illustrated in Figure 2.

Here, it should be noted that this version of the circuit is a realization of the LFSR in [23]. The equivalent class of the LFSR is given in Figure 3 (using the Java-programming language). Here, it is important to ensure that the software implementation obeys the hardware implementation strictly. By doing so, it can be undertaken the fault injection and produce the OTS in the software domain without affecting the logical of relation and parameter interactions of the hardware implementation.

Now, the TQS process can be applied as illustrated in Section2. Here, there are 74 faults injected in the system. To assist this work, MC_MIPOG [7] is used to produce the TC.

Following the steps in TQS process, Table 1 demonstrates the derivation of OTS. Here, it should be noted that the first 811 test cases could remove all the faults. Furthermore, only the first 164 test cases when t = 8 are needed to catch that last two live mutants. The efficiency of integration MC_MIPOG with MuJava can be observed (by taken only the effective TC) in the last column in Table 1.

Table 2 gives the desired OTS and SV, where T and F represent true and false, respectively. In this case, TQS process reduces the test size to twelve test cases only, which significantly improves the TV process.

To illustrate how the verification process is done (see Figure 2), assume that the second output (i.e., S2) is out-of-order (i.e., malfunction). Suppose that S2 output is always off (i.e., short circuit to "Ground"). This fault cannot be detected in TC1 (according to Table 2). Nevertheless, when TC2, the output vector ("VV") of faulty IC, is FFTTTTTTTTTT, and the SV is FTTTTTTTTTTT, the TV process can straightforwardly detects that the IC is malfunctioning (i.e., cut fails). Now, the saving efforts factor is *4096 – 12/4096 = 99.70 %.*

Normal vs. Cumulative Mode of Operation for CIT

This section repeats the experiment for LFSR by recording the mutant score for each degree of interaction independently, that is, by putting CIT in normal mode of operation during TQS process and fix t (i.e., non-cumulative process) as given in Table 3.

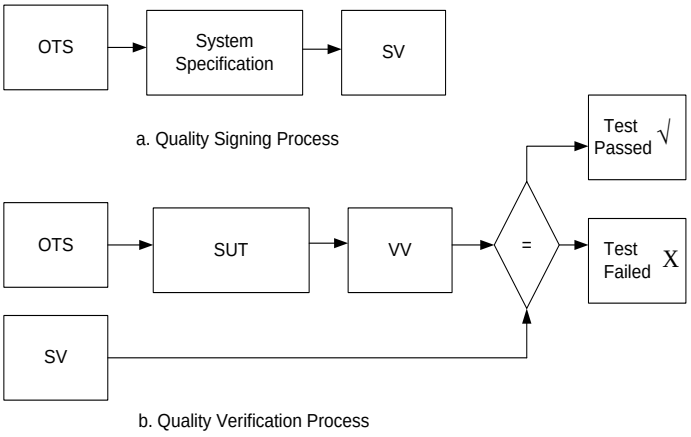

Figure 1: The Quality Signing and Verification Processes [20]

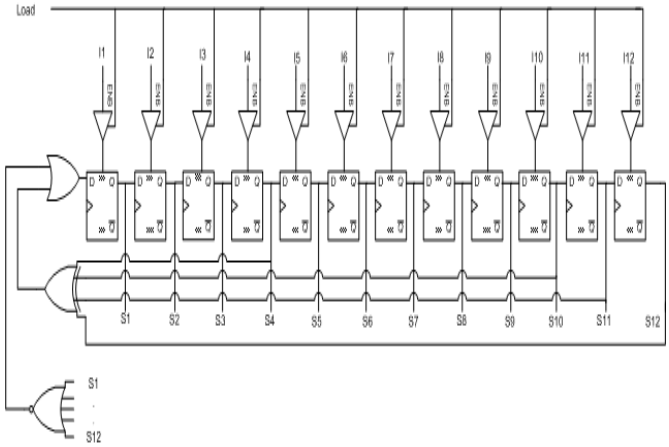

Figure 2: Schematic diagram for 12-bit LFSR Circuit Diagram

```
public class LFSR {
private static boolean s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12;
staic load (boolean i1,boolean i2,
        boolean i3,
        boolean i4,boolean i5,boolean i6, boolean i7,
        boolean i8,boolean i9,boolean i10, boolean i11, boolean i12)

{
s1=i1;s2=i2;s3=i3;s4=i4;s5=i5;s6=i6;s7=i7;s8=i8;s9=i9;s10=i10;s11=i11;s12=i12;
}

public static String getnext( boolean i1,boolean i2,
        boolean i3,
        boolean i4,boolean i5,boolean i6, boolean i7,
        boolean i8,boolean i9,boolean i10, boolean i11, boolean i12  ){

    load(i1,i2,i3,i4,i5,i6,i7,i8,i9,i10,i11,i12);
    String output=null;
      boolean  c;
      c=!(s1|s2|s3|s4|s5|s6|s7|s8|s9|s10|s11|s12);
      boolean xsr;
     xsr=s12^s11^s10^s4;
     s2=s1;
     s3=s2;
     s4=s3;
     s5=s4;
     s6=s5;
     s7=s6;
     s8=s7;
     s9=s8;
     s10=s9;
     s11=s10;
     s12=s11;
     s1=c|xsr;
     output=""+s1+s2+s3+s4+s5+s6+s7+s8+s9+s10+s11+s12;
      return output;
}
}
```

Figure 3: Equivalent class Java program for 12-bit LFSR

By considering the size of test suite from different points of view, it can be discussed the evaluation of CIT and mutation testing for optimization purpose. Referring to Table 3, it is noted that using exhaustive testing requires 4096 test cases. By taking the effective test size (i.e., by examine the effectiveness of the test suite) reduces the test cases to merely 20 test cases. Next, when relaxing *t* yields smaller test size (both the overall generated and the effective) and gives the 100% mutant score (for t=9, 10, and 11 where the test size=14, 15, and 17 respectively). Moreover, must of mutants are killed using low *t*. As such, considering the CIT approach can assist in reduction of test suite without going to exhaustive testing. On the other hand, to select suitable *t* is more questionable, that is, how the tester can predict suitable *t*? Here, it is required a more systematic manner, as this paper proposed to overcome these obstacles; this paper suggests the use of CIT in cumulative mode (starting from *t*=1). By comparing Tables 1 and 3, it is noted that the effective test size considering the CIT cumulative mode is less than that of normal mode. Moreover, 100% mutant score is achieved by considering cumulative mode up to t=8. In contrast, it is not until t=12 can the mutant score of 100% be achieved during normal mode. As such, it is more practical and systematic to consider the cumulative mode of operation for CIT. Finally, it is clear that combining both black and white boxes strategies is significantly required.

CONCLUSION

This paper has generalized the Quality Signing and Verification Processes to involve the sequential circuit hardware testing. The case study in hardware production line demonstrated that the proposed strategy could improve the saving efforts factor significantly. In addition, the justification for the requirement of the cumulative mode of operation for CIT has been discussed. Finally, as a part of future work, it is desired to investigate the application of the proposed strategy for computer-aided hardware design tool.

REFERENCES AND NOTES

1. Jenny tool, August 2011, http://www.burtleburtle. Net/bob/math/.
2. TVG tool, August 2011, http://sourceforge.net/projects/tvg/.
3. Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG: a general strategy for T-way software testing," in Proceedings of the International Symposium and Workshop on Engineering of Computer Based Systems, pp. 549–556, Tucson, Ariz, USA, March 2007.
4. Y. Lei, R. Kacker, D. R. Kuhn, V. Okun, and J. Lawrence, "IPOG-IPOG-D: efficient test generation for multi-way combinatorial testing," Software Testing Verification and Reliability, vol. 18, no. 3, pp. 125–148, 2008.
5. M. Forbes, J. Lawrence, Y. Lei, R. N. Kacker, and D. R. Kuhn, "Refining the in-parameter-order strategy for constructing covering arrays," Journal of Research of the National Institute of Standards and Technology, vol. 113, no. 5, pp. 287–297, 2008.
6. R. C. Bryce and C. J. Colbourn, "A density-based greedy algorithm for higher strength covering arrays," Software Testing Verification and Reliability, vol. 19, no. 1, pp. 37–53, 2009.
7. M. I. Younis, and K. Z. Zamli, "MC-MIPOG: A Parallel t-Way Test Generation Strategy for Multicore Systems," ETRI Journal, vol.32, no.1, pp. 73-83, 2010.
8. D. R. Wallace and D. R. Kuhn, "Failure modes in medical device software: an analysis of 15 years of recall data," International Journal of

Reliability, Quality, and Safety Engineering, vol. 8, no. 4, pp. 351–371, 2001.

9. D. R. Kuhn and M. J. Reilly, "An investigation of the applicability of design of experiments to software testing," in Proceedings of the 27th NASA/IEEE Software Engineering Workshop, pp. 91–95, IEEE Computer Society, December 2002.
10. D. R. Kuhn, D. R. Wallace, and A. M. Gallo Jr., "Software fault interactions and implications for software testing," IEEE Transactions on Software Engineering, vol. 30, no. 6, pp. 418–421, 2004.
11. D. R. Kuhn and V. Okun, "Pseudo-exhaustive testing for software," in Proceedings of the 30th Annual IEEE/NASA Software Engineering Workshop (SEW '06), pp. 153–158, April 2006.
12. R. Kuhn, Y. Lei, and R. Kacker, "Practical combinatorial testing: beyond pairwise," IT Professional, vol. 10, no. 3, pp.19–23, 2008.
13. D. T. Tang and C. L. Chen, "Iterative exhaustive pattern generation for logic testing," IBM Journal of Research and Development, vol. 28, no. 2, pp. 212–219, 1984.
14. S. Y. Boroday, "Determining essential arguments of Boolean functions," in Proceedings of the International Conference on Industrial Mathematics (ICIM '98), pp. 59–61, Taganrog, Russia, 1998.
15. A. K. Chandra, L. T. Kou, G.Markowsky, and S. Zaks, "On sets of Boolean n-vectors with all k-projections surjective," Acta Informatica, vol. 20, no. 1, pp. 103–111, 1983.
16. G. Seroussi and N. H. Bshouty, "Vector sets for exhaustive testing of logic circuits," IEEE Transactions on Information Theory, vol. 34, no. 3, pp. 513–522, 1988.
17. I. I. Dumer, "Asymptotically optimal codes correcting memory defects of fixed multiplicity," Problemy Peredachi Informatskii, vol. 25, pp. 3–20, 1989.
18. S. Ghosh and J. L. Kelly, "Bytecode fault injection for Java software," Journal of Systems and Software, vol. 81, no. 11, pp.2034–2043, 2008.
19. A. A. Avizienis, the Methodology of N-Version Programming, Software Fault Tolerance, John Wiley & Sons, New York, NY, USA, 1995.
20. M. I. Younis, and K. Z. Zamli, "A Strategy for Automatic Quality Signing and Verification Processes for Hardware and Software Testing," Inventi Impact: Tech Research & Reviews Vol. 2011, Issue 3, pp. 97-101, 2011.
21. Y.-S. Ma, J. Offutt, and Y. R. Kwon, "MuJava: an automated class mutation system," *Software Testing Verification and Reliability*, vol. 15, no. 2, pp. 97–133, 2005.
22. MuJava Version 3, August 2011, http://cs.gmu.edu/ ~offutt/mujava/.
23. Linear Feedback Shift Register, January 2012, http://www.altera.com/products/ip/dsp/signal_generation/m-nov-linear.html.

Acknowledgments**:** The authors acknowledge the help of Jeff Offutt, Jeff Lei, Raghu Kacker, Rick Kuhn, Myra B. Cohen, and Sudipto Ghosh for providing them with useful comments and the background materials.