1998 International Computer Symposium
Workshop on Computer Architecture
December 17-19, 1998, N.C.K.U., Tainan, Taiwan, R.O.C.

BEHAVIORAL SYNTHESIS-FOR-TESTABILITY FOR CONDITIONAL STATEMENTS WITH MULTIPLE BRANCHES

C.-Z. Yung and S.-J. Wang
Institute of Computer Science
National Chung-Hsing University
Taichung 402, Taiwan, ROC
E-mail:{czyung, sjwang}@cs.nchu.edu.tw

Abstract

As the digital design moves to higher levels of abstraction, high-level test synthesis methodologies attract many research interests. Many conditional statements in behavioral descriptions tend to produce testability problems in synthesized circuits, so it would be better if they are taken care of in the early stage of the design cycle. In this paper we present a HLTS methodology based on BIST. The presented methods transform conditional *case* statements in the original design to a functionally equivalent description that eliminates testability problems exist in the original design. Experimental results are provided to show the effectiveness of these methods.

1. Introduction

The rapid increasing of VLSI density creates great challenge to the design and testing of VLSI circuits. Given the complexity of contemporary VLSI circuits, it is very difficult to control or observe signals inside a chip, which in turns makes circuit testing difficult. Design-for-testability (DFT) methodologies are thus developed to improve the quality of VLSI testing and reduce testing cost. Two well-known and widely used DFT techniques are scan design and Built-In Self-Test (BIST).

The complexity of circuits also push the digital circuit design to move toward higher levels of abstraction. CAD tools that accept and optimize designs in Register-Transfer Level (RTL) have been used for years. High-level synthesis, which translates designs specified in the behavioral domain to the structural domain (RTL), is also getting popular recently. In order to consider testability issues in the early stage of a design cycle, high-level test synthesis is the subject of intense research in recent years.

High-level synthesis for testability (HLTS) tries to transform a design description to another one with equivalent functionality and improved testability [1]. Many HLTS techniques have been presented, including techniques at RT level [2]-[8] and behavioral level [9]-[13]. Behavioral synthesis for testability can be targeted for ATPG [9] or BIST [10]-[13]. These techniques try to find behavioral statements that may cause testability problems and modify the statements for better testability. Conditional statements are the ones that are most likely to cause testability problems. These statements include conditional *loop* statement [9] and *if-then-else* statement [13].

Conditional *case* statements are commonly used in behavioral and RTL descriptions to provide multiple branch points. This statement may decrease testability if all branch points are not taken with the same probability. Under BIST environment, this unequal probability distribution may adversely impact the testability of synthesized circuits: branches are applied with different number of test patterns, and the quality of applied test patterns may also be degraded.

In this paper we discuss how to improve the testability of behavioral descriptions with multiple branches. In the next section we give preliminary information. The testability problems cause by conditional *case* statements are discussed in Sec. 3, and our approach to this problem is presented in Sec 4. The experimental results are given in Sec. 5; these results show that our methods effectively solve the testability problems with insignificant overhead. Concluding remarks are given in Sec 6.

2. Preliminaries

Built-In Self-Test (BIST) is a widely used DFT technique [14]. In a circuit with BIST, test patterns are generated on chip, and output responses are also analyzed on chip. In order to achieve this goal, the BIST structure reconfigures part of the functional circuit to be a test pattern generator (TPG) and some other to be an output response analyzer (ORA). The rest circuit consists of the circuit under test (CUT). A TPG is usually made of a linear feedback shift register (LFSR). Test patterns generated by the TPG are fed to the CUT, while output responses of the CUT are collected and analyzed by the ORA. The most popular ORA design is the multi-input signature register (MISR), which is also an LFSR. Fig. 1 gives a simple illustration of a general BIST structure.

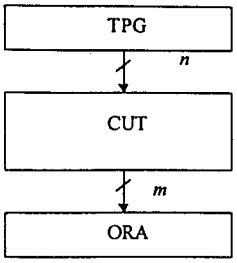


Fig. 1. The BIST structure

Three testing methodologies are generally used in BIST: exhaustive, pseudo-exhaustive, and pseudorandom testing. This classification is made according to the applied test patterns. Consider the BIST structure shown in Fig. 1. In an exhaustive testing, all 2^n input vectors are applied to the CUT, and this approach guarantees that all non-redundant faults that do not produce sequential behavior will be detected. The problem is the large amount of test vectors required.

Exhaustive testing is not practical for larger circuits. In pseudoexhaustive testing, a group of input lines of the CUT can be driven by the same test signal as long as they do not affect the same output. In this way, the number of test signals may be much smaller than the number of CUT input lines, which greatly reduces the number of distinct test patterns required.

If the sequence of test patterns is pseudorandom, it is called a pseudorandom testing. This technique is usually applied when the number of exhaustive test patterns is too large. Fault simulation is conducted to decide the number of test patterns required for a certain level of fault coverage. The sequence of pseudorandom test vectors has many properties like those of random sequences. However, they are not really random since the sequence of patterns is deterministic and thus repeatable. A TPG that generates pseudorandom patterns is called a pseudorandom pattern generator (PRPG). LFSRs with primitive characteristic polynomials are most common PRPGs. MISRs can also be used as pseudorandom pattern generator [15]. For example, if the ORA in Fig. 1 is an MISR and its outputs are used to drive another functional circuit, then this MISR acts as a PRPG for the circuit it drives.

Randomness [16] is derived from the entropy calculation as defined in information theory. The extreme values for randomness are zero, which indicates the variable is really a constant, and one, which indicates the variable has a uniform probability distribution. In pseudorandom testing the input of a CUT should have a randomness as close to one as possible; otherwise, the quality of testing may be questionable.

3. Testability Problems

The BIST structure discussed above may encounter some testability problems. For example, consider a simple ALU structure shown in Fig. 2. The ALU has two inputs coming from registers R_1 and R_2, and its output is stored in R_3. The function executed by the ALU is controlled by R_4, which contains signals produced by the control circuit.

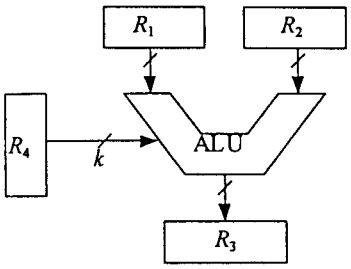


Fig. 2. A simple ALU.

In order to exhaustively test the ALU, registers R_1, R_2, and R_4 should become a test pattern generator which generates exhaustive test patterns for the ALU. With exhaustive testing, all non-redundant stuck-at faults in the circuit under test (ALU in this example) are guaranteed to be detected. The only problem is the number of test patterns is usually prohibitive.

In order to reduce the number of test vectors needed and thus reduce testing time, either pseudorandom testing or pseudoexhaustive testing can be used. With these strategies, the registers may be configured in various ways. For example, during a BIST session R_1 and R_2 may become (or the contents in them are from) a single TPG, while register R_4 is an independent TGP. When pseudorandom test patterns are applied, fault simulation is usually conducted first to decide the number of test patterns needed to achieve a required level of fault coverage.

Suppose that the number of different functions in the ALU is N. From the point of view of pseudorandom testing, the ideal case is $N = 2^k$, where each function corresponds to exactly one control code. In this case, when we apply L test patterns to the ALU, each functional unit is exercised by about $L/2^k$ patterns. However, in real circuits it is more likely that we have $N < 2^k$. In this case, a functional unit may be exercised by more than one control code, and some control codes may be unused (i.e., they do not activate any functional unit). This situation brings up some testability problems that are summarized below..

Suppose that there are unused control codes (the control codes are stored in register R_4 in Fig. 2). In this case, some test cycles will be wasted during BIST sessions since all patterns will be generated in R_4 but none of the functional units are activated for the unused codes.

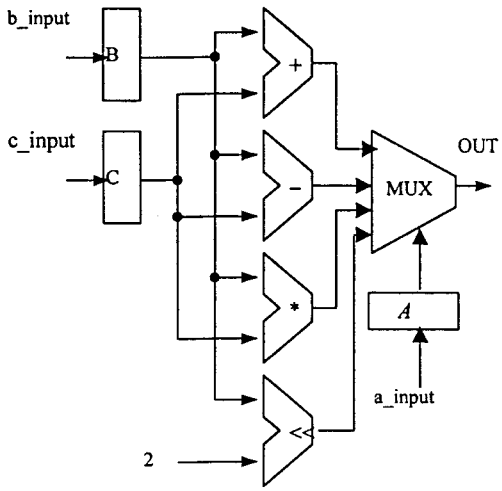
- Let functional unit FU_i be activated by m_i different control codes, where $2^k > m_i \geq 1$. The number of test patterns accepted by FU_i would be around $L \times m_i / 2^k$ during a BIST session. Thus if $m_i \neq m_j$ for two units FU_i and FU_j, the number of test patterns exercised by the two units will be different.
- Perhaps the most serious problem is that the *randomness* of test vectors accepted by a function unit may be changed in these cases, which will affect the quality of tests.

The above problems can be illustrated with the example shown in Fig. 3. Fig. 3(a) is a piece of Verilog code, and Fig. 3(b) is the block diagram of the circuit described in Fig. 3(a).

```
//Sequential Logic
always @(posedge CLK)
begin
        :
    A = a_input;
    B = b_input;
    C = c_input;
        :
end

//Combinational Logic
always @(...A or B or C...)
begin
        :
    case(A)
        4'b0001: OUT= C+B;
        4'b1001: OUT= C-B;
        4'b1111: OUT= C*B;
        default: OUT= C<<2;
    endcase
        :
end
```

(a)



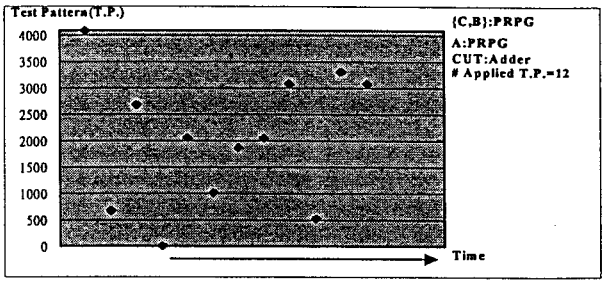
(b)

Fig. 3. (a) A description with case statement, (b) the corresponding circuit.

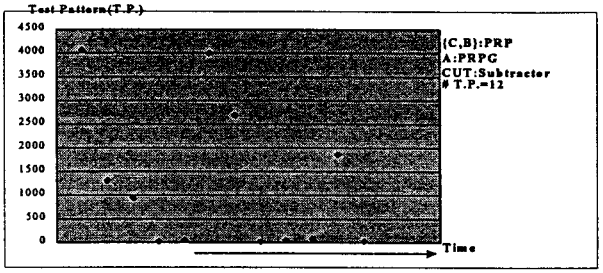
In order to implement BIST environment for the circuit, registers B and C become a single PRPG during testing time, and register A becomes an independent PRPG. Since both B and C are 6-bit wide, in all there are $2^{12} = 4098$ different input combinations. In our experiment, we generate 200 test patterns with PRPG $\{C, B\}$, while register A is also configured as a PRPG. The generated test patterns are shown in Fig. 4, in which Y-axis shows the pattern (in decimal) while X-axis is the time. The test patterns accepted by the adder, subtractor, multiplier, and shifter are shown in Fig. 4(a), 4(b), 4(c), and 4(d), respectively.

Among the four functional units, the shifter can be activated by 13 different control codes, while each of the other three units can be activated by exactly one control code. Therefore, with randomly generated control code in register A, the probability that the shifter is activated is 13/16, while the probability that any one of the other three units is activated is 1/16. This can be seen from the number of test patterns generated for the four units shown in Fig. 4.

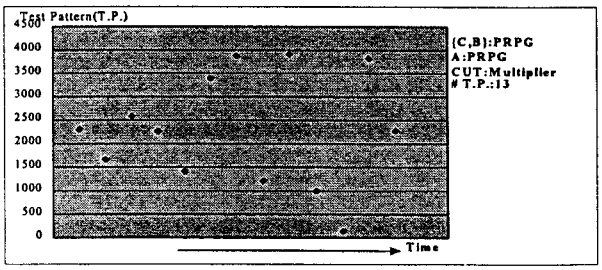
Ideally, the test patterns received by each functional unit should be uniformly distributed among possible patterns. Unfortunately, the patterns distributed to a given unit may not be random any more, which is evident in Fig. 4(b). This loss of randomness will affect the quality of test.



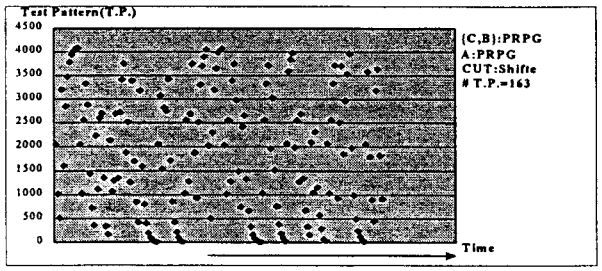
(a)



(b)



(c)



(d)

Fig. 4. Distribution of test patterns exercised by a functional unit when register A is a PRPG: (a) adder, (b) subtractor, (c) multiplier, (d) shifter.

PRPGs may be replaced with MISRs. MISRs also generate random patterns, but the patterns are nondeterministic. In Fig. 5 we show the distribution of test patterns for the circuit shown in Fig. 3(b) with register A configured as an MISR. It can be seen that the same testability problems persist.

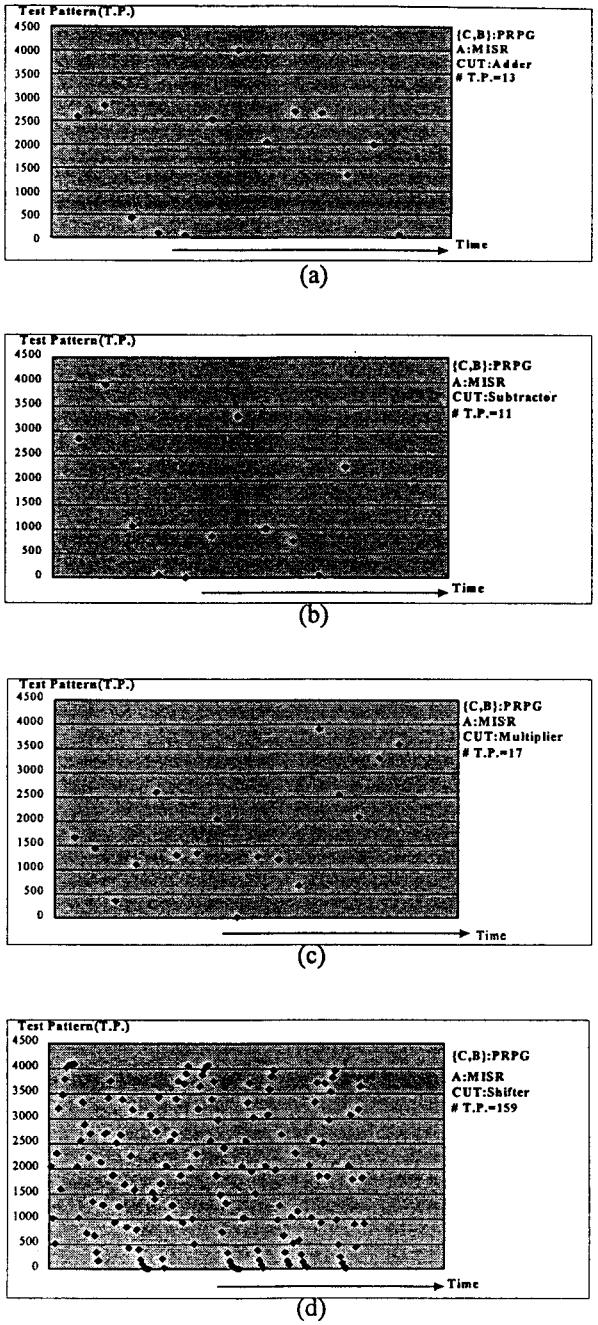


Fig. 5. Distribution of test patterns exercised by a functional unit when register *A* is an MISR: (a) adder, (b) subtractor, (c) multiplier, (d) shifter.

4. Testability Enhancement

From the discussion given above it is clear that we have to equalize the probability of selection of any functional unit in order to improve testability under BIST environment. In this section we present our approach to this problem. First we presented two methods that solve the testability problems when pseudorandom testing is applied. In the later part of the section we will discuss some difficulties that will appear when exhaustive testing is applied and outline possible solutions to this problem.

4.1. Method 1

The most obvious and straightforward way to evenly distribute test patterns among all functional units is to include an extra counter. When a circuit operates in test mode, the counter replaces the control part of the circuit to generate required control codes. This can be easily done by modifying the original behavioral or RTL description. For example, consider the circuit shown in Fig. 3 again. The modified circuit is shown in Fig. 6, in which Fig. 6(a) shows how to modify the original Verilog code for testability enhancement and Fig. 6(b) shows the block diagram of the new circuit.

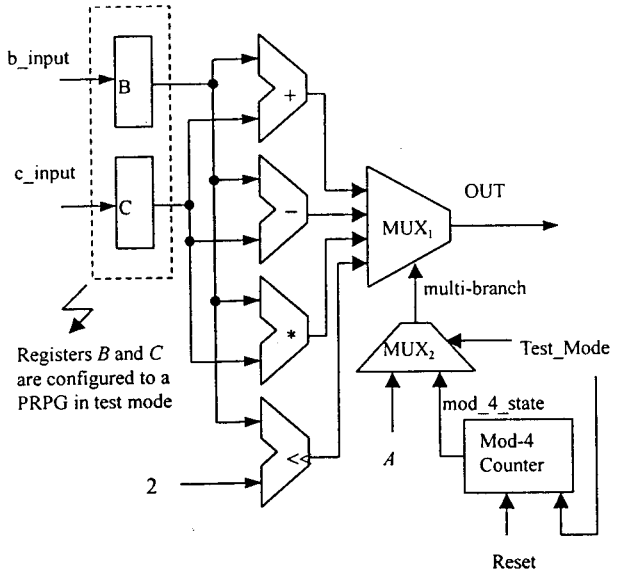
```
always @(B or C)  //Feedback path of LFSR
  k = C[n-1]^...^B[0];

always @(posedge CLK or posedge Reset)
       //Sequential Logic
          :
  if (Reset)
    (C,B,A) = {0,0,0};
  else
    if (Test_Mode)
      {C[n-1],C[n-2:0],B[n-1],B[n-2:0]} =
                {k,C[n-1:1],C[0],B[n-1:1]};
//In test mode, B and C are configured as a PRPG
    else
      {C,B,A} = {c_input,b_input,a_input};
          :

always @(posedge CLK or posedge Reset)
       //mod-4 counter
  if (Reset) mod_4_state=0;
  else
    if (mod_4_state==4'b0011) mod_4_state=0;
    else
      if (Test_Mode)
        mod_4_state = mod_4_state+1;

always @(... Test_Mode or mod_4_state or A or B
or C ...)  //Cequential Logic
  begin
          :
    if (Test_Mode)
      begin
        multi_branch = mod_4_state;
        {branch1,branch2,branch3} =
                  {4'b0000,4'b0001,4'b0010};
      end
    else
      begin
        multi_branch = A;
        {branch1,branch2,branch3} =
                  {4'b0001,4'b1001,4'b1111};
      end
    case (multi_branch)
      branch1:OUT = B+C;
      branch2:OUT = B-C;
      branch3:OUT = B*C;
      default:OUT = B<<1;
    endcase
          :
  end
```

(a)



(b)

Fig. 6. Testability Enhancement by Method 1:
(a) Verilog code, (b) block diagram.

In this method, contents of the Mod-4 counter replace the conditional variable of the *case* statement during test mode. As a result, test results of the four functional units are transferred to the output with equal probability, which solves the testability problems discussed in Sec. 3.

The approach presented here, however, does suffer from other testability problem. In Fig. 6(b), the path from register A to MUX_1 is never exercised in test mode. Therefore, part of the control circuit in MUX_1 is not tested, nor is the data path entering MUX_2 from register A. The untested circuit will eventually affect the final fault coverage. The method presented below is developed to deal with this problem.

4.2. Method 2

An alternate approach is to replace register A with a simple synchronous sequential circuit A'. During a test session, register A is configured as a special counter that generates control code is such a way that test results of all functional units are allowed to be evenly transmitted to the output of the multiplexer. In this method, the conditional variable of the *case* statement is not replaced by other variable in test mode. Instead, an extra circuit is used to generate required states in A. The modified Verilog code and the corresponding circuit block diagram is shown in Fig. 7.

In test mode register A repeatedly goes through four states {0001, 1001, 1111, 0000}. Actually, the last state can be any 4-bit vector that is not one of {0001, 1001, 1111}, since any one of them corresponds to the 'default' condition in the *case* statement.

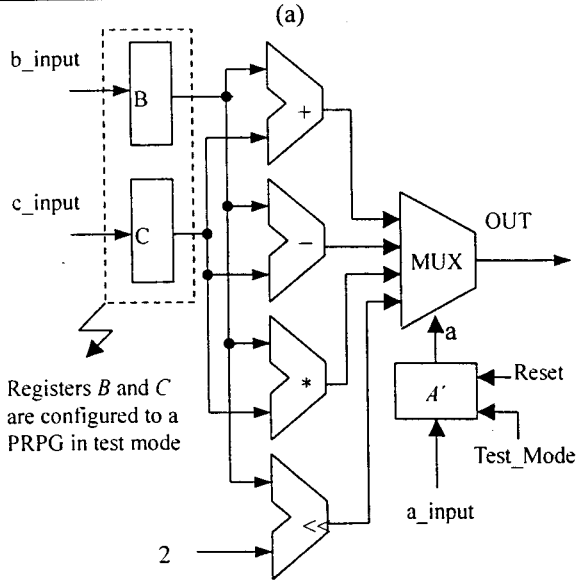
This method achieves the same goal as method 1, since the patterns generated by PRPG {C,B} are applied to all units, and the results are transmitted to the output with equal probability.

```
always @(B or C)  //Feedback path of LFSR
  k = C[n-1]^ … ^B[0];

always @(posedge CLK or posedge Reset)
    //Sequential Logic
        :
  if (Reset)
    (C,B,A) = {0,0,0};
  else
    if (Test_Mode)
     begin
      if (A==4'b0001) A = 4'b1001;
      else if (A==4'b1001) A = 4'b1111;
      else if (A==4'b1111) A = 4'b0000;
    // The next state of 1111 can be any
    // element not in {0001, 1001, 1111}
      else A = 4'b0001;
      {C[n-1],C[n-2:0],B[n-1],B[n-2:0]} =
                {k,C[n-1:1],C[0],B[n-1:1]};
//In test mode, B and C are configured as a PRPG
     end
    else
      {C,B,A} = {c_input,b_input,a_input};
        :

always @(…A or B or C …)  //Combinational Logic
  begin
        :
    case (A)
      4'b0001:OUT = B+C;
      4'b1001:OUT = B-C;
      4'b1111:OUT = B*C;
      default:OUT = B<<1;
     endcase
        :
  end
```

(a)



(b)

Fig. 7. Testability Enhancement by Method 2:
(a) Verilog code, (b) block diagram.

The advantage if this approach is that it alleviates most of the testability problems occur in method 1. First of all, there are no untested data paths in the test circuit (e.g., MUX_2 in Fig 6(b)). With properly selected state sequence,

all stuck-at faults on signal lines a (which control the MUX in Fig 7(b)) are detectable. For example, the state sequence described above ({0001, 1001, 1111, 0000}) satisfies this requirement. Whether the control part of the MUX is fully tested really depends on the test patterns, that is, the state sequence generated by A'.

4.3. Exhaustive Testing

In some cases the above methods are also applicable to exhaustive testing. This happens when the number of functional units is an odd number. The number of exhaustive test patterns is always a power of 2. Let the number of test patterns be 2^n, and the number of functional units be p. If p is an odd number, then 2^n and p are mutually prime. As a result, all functional units are exhaustively tested and all their results are transferred to the output after $p \times 2^n$ test cycles. On the other hand, if p is an even number, some test results can not be transmitted for error checking. For example, if $p = 4$, then in the above methods each functional unit is only able to transfer 1/4 of the 2^n test results to the output for analysis, no matter how many more test cycles are applied.

The above problem can be solved with more complicated test circuits. Due to space limitation, the detail implementation is not discussed here [17].

5. Experimental Results

In order to show the effectiveness of the proposed methods, we have conducted some experiments, and the results are given in this section.

First we will show the distribution of test vectors when our methods are implemented. We shall consider the circuit shown in Fig. 3 again in order to compare with the results shown in Fig. 4 and Fig. 5. Here registers $\{C,B\}$ are configured into a single PRPG in the test mode, while register A is modified with either one of the two methods presented in Sec. 4 (the results are the same). The distribution of test patterns is shown in Fig. 8.

From the patterns shown in Fig. 8 it is clear that all functional units are exercised with the same frequency. In other words, among the 200 patterns applied to them, each unit has 50 results transmitted to the output for error checking. The test patterns exercised by each unit are well distributed; that is, the randomness of patterns accepted by a unit is improved compared with the original circuit.

Next we will show the impacts on area and speed due to the modifications. Based on the Verilog code given in Fig. 3(a), we synthesize the following three circuits.

1. Circuit I: Registers $\{C,B\}$ are configured into a single PRPG in the test mode, while register A behaves as a 4-bit PRPG during test mode.
2. Circuit II: Registers $\{C,B\}$ become a PRPG in the test mode, and register A is modified according to Method 1.
3. Circuit II: Registers $\{C,B\}$ become a PRPG in the test mode, and register A is modified according to Method 2.

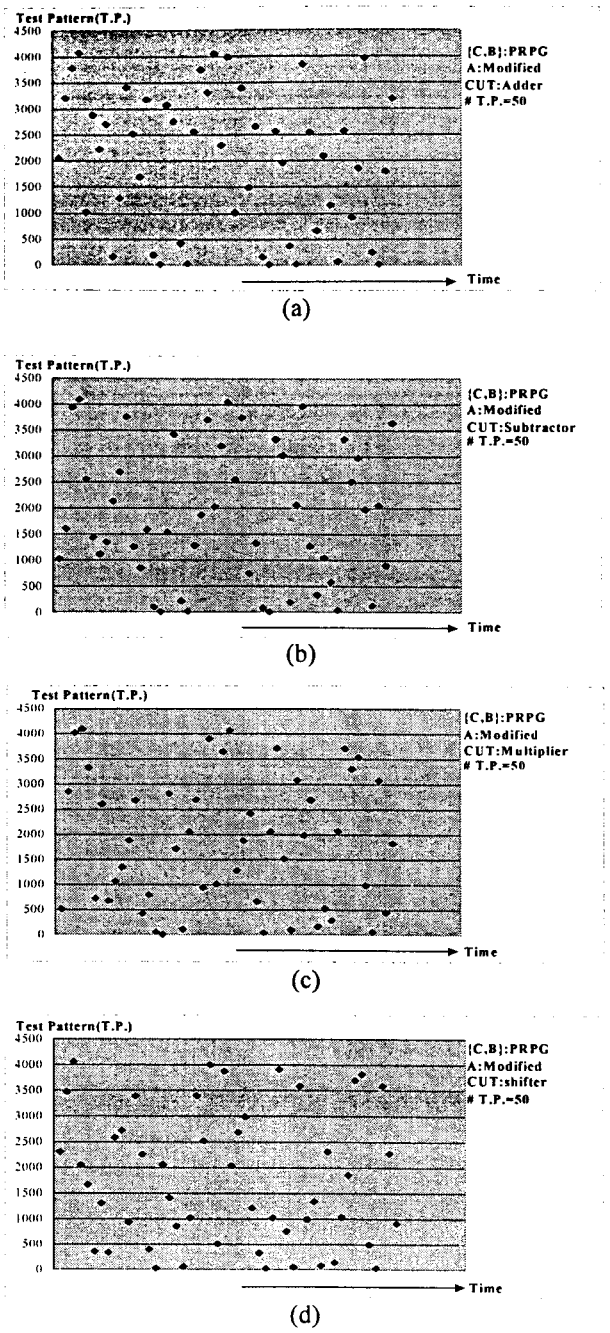


Fig. 8. Distribution of test patterns exercised by a unit when register A is modified for testability enhancement: (a) adder, (b) subtractor, (c) multiplier, (d) shifter.

Two sets of circuits are synthesized. In the first set the width of registers B and C are 8-bit, and the results are shown in Table I. In the second set the width of B and C are16-bit, and the results are listed in table II. A is always a 4-bit register. The circuits are synthesized with Design Compiler from Synopsis with a 0.25μm cell library.

From the results in Tables I and II, it can be seen that circuits modified with Method 2 produces negligible area and delay overhead, while area and delay penalty caused by Method 1 can be significant for smaller circuits. In both cases, the overhead reduces as the circuit size increases.

Table I. Area and Delay of synthesized circuits: 8-but data registers *B* and *C*

Circuit Type	Combinational Area	Noncombinational Area	Total Area	Area Overhead (%)	Data Arrival Time	Delay Overhead (%)
I	23029.93	3852.25	26882.18	0.00	5.58	0.00
II	23988.37	4460.50	28448.87	5.85	6.08	8.96
III	23232.67	3852.25	27084.92	0.75	5.56	0.00

Table II. Area and Delay of synthesized circuits: 16-but data registers *B* and *C*

Circuit Type	Combinational Area	Noncombinational Area	Total Area	Area Overhead (%)	Data Arrival Time	Delay Overhead (%)
I	76564.04	7096.25	83660.29	0.00	10.27	0.00
II	77494.84	7704.50	85199.34	1.84	10.27	0.00
III	76766.77	7096.25	83863.02	0.24	10.27	0.00

6. Concluding Remarks

Conditional *case* statements may create testability problems under BIST environment once the circuit is synthesized. In this paper we discussed how these statements affect the testability of synthesized circuits, and presented two methods to transform the original specification to a more testable design. These methods effectively improve the testability of synthesized circuits with very low overhead, and the area and performance penalty diminish as the circuit grows in size.

Reference

[1] T. Thomas, P. Vishakantantaiah, and J. A. Abraham, "Impact of behavioral modifications for testability," in *Proc. IEEE VLSI Test Symp.*, pp. 427-432,1994.

[2] V. Chickermane, J Lee, and J. H. Patel, "A comparative study of design for testability methods using high-level and gate-level descriptions," in *Proc. Int. Conf. Computer-Aided Design*, pp. 620-624, 1992.

[3] S. Bhattacharya, F. Brglez, and S. Dey, "Transformations and resynthesis for testability of RT-level control-data path specifications," *IEEE Trans. VLSI Systems*, vol. 1, no. 3, pp. 304-318, Sep. 1993.

[4] K. T. Cheng and V. D. Agrwal, "A partial scan method for sequential circuits with feedback," *IEEE Trans. Comput.*, vol. 39, pp. 544-548, Apr. 1990.

[5] D. H. Lee and S. M. Reddy, "On determining scan flip-flops in partial-scan designs," in *Proc. Int. Conf. Computer-Aided Design*, pp. 322-325, 1990.

[6] V. Chickermane and J. H. Patel, "An optimization based approach to the partial scan design problem," in *Proc. Int. Test Conf.*, pp. 377-386, Sep. 1990.

[7] S. Bhattacharya and S. Dey "H-Scan: a high level alternative to full-scan testing with reduced area and test application overhead," in *Proc. VLSI Test Symp.*, Apr. 1996.

[8] S. Dey, M Potkonjak, "Non-scan design for testability of RT-level data paths," in *Proc. Int. Conf. Computer-Aided Design*, pp. 640-645, 1994.

[9] F. F. Hsu, E. M. Rudnick, and J. H. Patel, "Enhancing high-level control-flow for improved testability," *Int. Conf. Computer-Aided Design*, pp. 322-328, Nov. 1996.

[10] C.A. Papachristou, J. Carletta, "Test synthesis in the bBehavioral domain," *Int. Test Conf.*, Oct. 1995.

[11] S. Dey and M. Potkonjak, "Transforming behavioral specifications to facilitate synthesis of testable designs," in *Proc. Int. Test Conf.*, pp. 184-193, 1994

[12] A. Majumdar, R. Jain, and K. Saluja, "Incorporating testability considerations in high-level synthesis," *J. Electronic Testing: Theory and applications*, pp. 43-55, Feb. 1994.

[13] K.A. Ockunzzi, C.A. Papachristou, "Testability enhancement for behavioral descriptions containing conditional statements," in *Proc. Int. Test. Conf.*, 1996.

[14] M. Abramovici, M. A. Breuer, A. D. Friedman, Digital system testing and testable design, *W. H. Freeman and Company*, 1990.

[15] K. Kim, D. S. Ha, and J. G. Tront, "On using signature register as pseudorandom pattern generator in built-in self-testing," *IEEE Trans. On Computer-Aided Design*, Vol. 7, No. 8, pp. 919-928, Aug., 1988.

[16] K. Thearling and J. A. Abraham, "An easily computed functional level testability measure," *Int. Test Conf.*, pp.381-390, Oct. 1989.

[17] C. Z. Yung, *Modifying behavioral description to improve testability problems caused by conditional statements*, (in Chinese), Master Thesis, Inst. of Computer Science, National Chung-Hsing University, Taiwan.