THE LFSR AND BCA VHDL MODELS FOR BUILT-IN SELF-TEST CIRCUITS

Jiří MITRYCH
PHOBOS, s.r.o.
Frenštát pod Radhoštěm
Czech Republic

Abstract

The various test structures are proposed for BIST techniques [1], [2]. A typical structure used for generation of pseudo-random test sets is the linear feedback shift register (LFSR). The BIST techniques have wide application in testing whole devices and embedded components. We focus on the analysis of the state coverage, fault coverage, and optimal structure of BIST schemes.

Keywords

Linear feedback shift register (LFSR), cellular automata (CA), middle grain FPGA structure, VHDL modelling, BIST structure optimal implementation

1. Introduction

There are given two examples of BIST design. Particularly the example of Linear feedback shift register (LFSR), and Binary cellular automata (BCA) [3]. On the basis of this examples it is shown that the time and complexity of simulation process of the both behavioural and register transfer VHDL models [4] will serve for optimisation of structure solution of BIST in the frame of FPGA equivalent gate grid optimisation [5].

One statement of the theory of LFSR says that a LFSR with characteristic polynomial being a primitive one has the cycle of maximum length. It can be shown, that the same statement is true for linear BCA. This problem is solvable. To solve this problem we have to solve the quadratic equation in the Galois field $GF(2^n)$.

The solution of this problem is known, but the computation complexity is approximately the same as the complete search. For the primitive polynomial of degree n we can get two different BCA with the vectors of the rules being reversed to each other. The best way how to model the mentioned type of BCA, with respect of the given goals, was to describe BCA by the VHDL model and to translate this description into the code suitable for simulation. For the simulation the V-system implemented by Mentor Graphics for HP workstation was used.

2. System Modelling

The best way of modelling LFSR or BCA is a VHDL description. This manner offers to be more flexible then a schematic way and it is convenient for using these structures like BIST in VHDL designs, which are technology independent.

Two procedures were constructed called LFSR and LHCA. The syntax for their using is:

LFSR (*output_state_vector,Clock,Preset)*;
LHCA (*output_state_vector,Clock,Preset)*;

Procedures were written for outputs from 2 up to 20 bits. LHCA is the cellular automata with 90 and 150 rules and maximum length of period and zero boundary conditions. Following are definitions of the cell rules:

90: $a_i(t+1) = a_{i-1}(t)$ xor $a_{i+1}(t)$
150: $a_i(t+1) = a_{i-1}(t)$ xor $a_i(t)$ xor $a_{i+1}(t)$

Both procedures automatically recognise the width of output vector and generate proprietary LFSR or BCA. A start seed is "00..01" in both cases and is done by *Preset* = 0. The active edge of clock signal *Clock* is the rising edge.

3. Simulation Arrangement

For studies of the testing features of LFSR and BCA we need some circuit for testing. The multiplier was chosen, be more preciously the multiplier 2x2 bits. For exploring stuck_at faults it was needed to translate the behavioural description to netlist, it means to structural description. Thus the Mult component was created. In this description it is possible to do stuck_at faults.

For simulation and comparing solutions a VHDL file with two components Mult and one multiplier MultCorrect for correct solution was created. One Mult is supplied by LFSR, the second by BCA. Outputs are compared with MultCorrect and the difference is highlighted by ChybaCA (for BCA supply) or ChybaLF (for LFSR supply) signal. The whole VHDL description follows:

```
Library ieee;

use ieee.std_logic_1164.all;
use work.bist.all;
```

```
entity Test_Mult is

   generic( SirkaT: Natural:=2);

   port(Reset  : in std_logic;
        Clk    : in std_logic;
        ChybaCA: out std_logic;
        ChybaLF: out std_logic;
        Soucin : out std_logic_vector
                 (2*SirkaT-1 downto 0));

   end;


architecture Test_Mult1 of Test_Mult is

   component Mult

   generic( Sirka: Natural:=SirkaT);

   port(A     : in std_logic_vector( Sirka-1
                downto 0);
        B     : in std_logic_vector( Sirka-1
                downto 0);
        Soucin: out std_logic_vector
               (2*Sirka-1 downto 0));

   end component;


   component MultCorrect

   generic( Sirka: Natural:=SirkaT);

   port(A     : in std_logic_vector( Sirka-1
                downto 0);
        B     : in std_logic_vector( Sirka-1
                downto 0);
        Soucin: out std_logic_vector
               (2*Sirka-1 downto 0));

   end component;


   signal TestVectorCA:
      std_logic_vector(2*SirkaT-1 downto 0);

   signal TestVectorLF:
      std_logic_vector(2*SirkaT-1 downto 0);

   signal VysledekCh:
      std_logic_vector(2*SirkaT-1 downto 0);

   signal SpravnyVysl:
      std_logic_vector(2*SirkaT-1 downto 0);

   signal VyslChLF:
      std_logic_vector(2*SirkaT-1 downto 0);

   signal SpraVyslLF:
      std_logic_vector(2*SirkaT-1 downto 0);


begin

   LHCA( TestVectorCA,Clk,Reset);

   LFSR( TestVectorLF,Clk,Reset);

   K1: Mult port map (TestVectorCA(2*
       SirkaT-1 downto SirkaT), TestVectorCA
       (SirkaT-1 downto 0) ,VysledekCh);

   K2: MultCorrect port map (TestVectorCA(2*
       SirkaT-1 downto SirkaT), TestVectorCA
       (SirkaT-1 downto 0), SpravnyVysl);

   K3: Mult port map (TestVectorLF(2*
       SirkaT-1 downto SirkaT), TestVectorLF
       (SirkaT-1 downto 0) ,VyslChLF);

   K4: MultCorrect port map (TestVectorLF(2*
       SirkaT-1 downto SirkaT), TestVectorLF
       (SirkaT-1 downto 0) ,SpraVyslLF);

   Soucin <= SpravnyVysl;

   ChybaCA <= '1' when
      (VysledekCh /= SpravnyVysl) else '0';

   ChybaLF <= '1' when
      (VyslChLF /= SpraVyslLF) else '0';

end;
```

4. Experimental Results

At first the comparison of output vectors of LFSR and BCA is presented. Let's see on Fig. 1. There are output vectors of 4 bits LFSR and BCA. Start seed is „0001". As it can be seen, The BCA has more changes between subsequent output vectors in a number of ones and zeros then LFSR. LFSR on the other hand has a groups of subsequent output vectors with the same number of ones and zeros. It's done by a shift principle.

These features are very interesting, because it looks like the time, which is needed to testing, is seed dependable for LFSR. At this moment there are no exhaustive solutions, but all simulations and comparing for seed „0001" were done. In this case (it means multiplier 2x2 bits) there are 7 signals, four of them are primary inputs. By simulations with stuck_at faults was found, that in 7 cases the BCA found the errors earlier then LFSR, in 6 cases LFSR and BCA found the error in the same step and only in one case LFSR found the error earlier then BCA.

One case for stuck_at_0 fault on B(0) primarily input is in Fig.2. The signal VysledekCh means output from multiplier with stuck_at_0 faults on B(0) and supplied by BCA. Signal VyslChLF means the same, but for LFSR. Signal SpravnyVysl is the output from correct multiplier.

Step	1.	2.	3.	4.	5.	6.	7.
LFSR	0001	0010	0100	1000	0011	0110	1100
BCA	0001	0011	0110	1011	0010	0101	1101

8.	9.	10.	11.	12.	13.	14.	15.
1011	0101	1010	0111	1110	1111	1101	1001
1001	0111	1000	0100	1110	1111	1100	1010

Fig. 1 Outputs of LFSR and BCA generators

5. FPGA Implementation Properties

The BIST is everyday compromise between test exhaustivity and adding hardware. It should be good to know how much hardware realisation of LFSR or BCA needs.

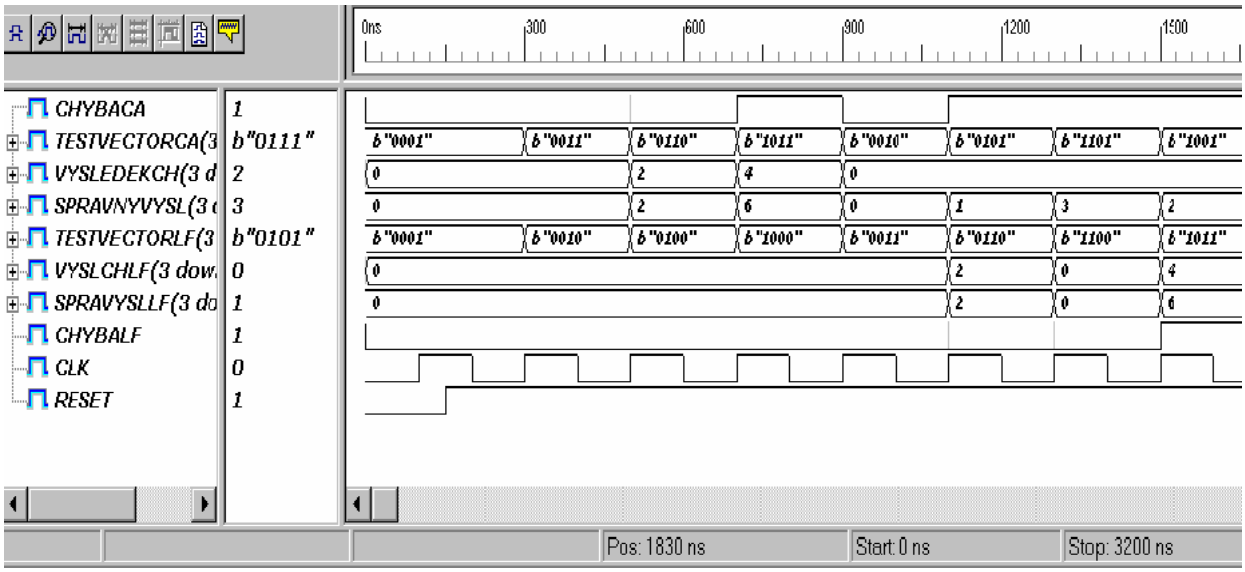

Fig. 2 Simulation outputs

---------- BCA 4 stages ----------

Synthesized design as a chip
Resource Usage Report of Random

Target Part: 42mx09-s
Combinational Cells: 2 of 336 (1%)
Sequential Cells: 4 of 348 (1%)
Total Cells: 6 of 684 (1%)
Clock Buffers: 0
IO Cells: 0

Details:

cm8: 1 comb: 1
xor2: 1 comb: 1
dfc1b: 1 seq: 1
dfm7a: 3 seq: 1

---------- BCA 15 stages ----------

Synthesized design as a block (not a chip)
Resource Usage Report of Random

Target Part: 42mx09-s
Combinational Cells: 2 of 336 (1%)
Sequential Cells: 15 of 348 (4%)
Total Cells: 17 of 684 (3%)
Clock Buffers: 0
IO Cells: 0

Details:

cm8: 1 comb: 1
xnor2: 1 comb: 1
dfm7a: 15 seq: 1

Fig. 3 BCA in number of cells (continuation from the previous column)

This solution is technology dependent, of course. Following is for ACTEL FPGA, 42MX family. The synthesis was done by Synplify running on PC. The solution for 4 or 15 stages is in Fig. 3. It's seen, that LFSR needs less cells then BCA.

---------- LFSR 4 stages ---------

Synthesized design as a block (not a chip)
Resource Usage Report of Random

Target Part: 42mx09-s
Combinational Cells: 1 of 336 (0%)
Sequential Cells: 4 of 348 (1%)
Total Cells: 5 of 684 (1%)
Clock Buffers: 0
IO Cells: 0

Details:

cm8: 1 comb: 1
dfc1b: 2 seq: 1
dfm7a: 2 seq: 1

---------- LFSR 15 stages ------

Synthesized design as a block (not a chip)
Resource Usage Report of Random

Target Part: 42mx09-s
Combinational Cells: 1 of 336 (0%)
Sequential Cells: 15 of 348 (4%)
Total Cells: 16 of 684 (3%)
Clock Buffers: 0
IO Cells: 0

Details:

cm8: 1 comb: 1
dfc1b: 13 seq: 1
dfm7a: 2 seq: 1

Fig. 4 LFSR in number of cells (continuation from the previous page)

6. Conclusion

As mentioned above, the LFSR is better from the point of view of the number of cells. In realisation, where the number of cells is critical, it's better to use LFSR. On

spite of this BCA can find the error earlier, so it can save the time for testing.

That's true that it is possible to put to LFSR another seed. But it increases number of cells, because in ACTEL FPGA there are D flip-flops with asynchronous Reset in sequential cell. To achieve Set results more cells that only one for flip-flop. From this point of view the BCA seems to be better.

This article is only the small piece of long way of comparing advantages and disadvantages LFSR and BCA. Much more simulation will be done to achieve more seriously conclusion, but now it is clear, that it is not simple to say, what is better for BIST, LFSR or BCA.

Acknowledgement

Support for GACR project "Research and Development of Built-in Diagnostics Means of Integrated Circuits" (No. 102/98/1003) is gratefully acknowledged.

References

[1] AGRAVAL, V. D., KIME, C. R., SALUJA, K. S. A Tutorial on Built-in Self-test, Part 1: Principles. IEEE Design & Test of Computers. 1993, vol. 10, no. 3, p. 73 - 82.

[2] HLAVICKA, J., NOVAK, O. Built-in Self-Test Equipment. In Proceedings of the conference DDECS '97. Beskydy Mountains (Czech Republic), 1997, p. 22 - 31.

[3] CIGANEK, P., DRABEK, V. Synthesis and Modelling of Binary Cellular Automata. In Proceedings of the conference DDECS '97. Beskydy Mountains (Czech Republic), 1997, pp. 79-85.

[4] VLCEK, K., MIKLIK, D. The VHDL Model of DEC-TED Memory Checker. In Proceedings of 9th European Workshop on Depend. Comp. Gdansk (Poland), 1998, p. 54 - 58.

[5] SOSNOWSKI, J., PAWLOWSKI, M. Universal and Application Dependent Testing of FPGAs. In Proceedings of EDCC-2 Workshop on Dependable Comp. Gliwice (Poland), 1996, p. 111 - 120.

[6] VLCEK, K. Compression and Error-Control Coding in Multimedia Communication. Praha: BEN technická literatura, 2000.

About author...

Jiří MITRYCH was born in 1957, graduated on Brno University of Technology in 1981. Since 1981 to 1992 he worked in Tesla Roznov like ASIC designer. This time he works as designer of ASIC based on FPGA ACTEL in PHOBOS company. At the time being, he is writing his Ph.D. thesis at the Department of Measurement and Control, FEI, Technical University of Ostrava. His main field of interests besides playing banjo is coding theory, design and diagnostics, and artificial intelligence.