GENETIC PROGRAMMING IN FEEDBACK REGISTERS DESIGNING

Ireneusz Gościniak

University of Silesia in Katowice, Institute of Computer Science
41-200 Sosnowiec, 39 Będzińska str., Poland

Abstract: In BIST structures feedback registers play the role of test generators and test response compactors. Linear feedback shift registers (LFSR) are here of predominating importance. These registers are relatively simple in designing. Non-linear feedback shift registers designing to diagnostic aims is considerably more complicated. The possibility to use the genetic programming to design the non-linear feedback shift registers is presented in the article. Usefulness of this approach to design the registers helpful in the BIST structures is testified by numerous examples. *Copyright 2004 IFAC*

Keywords: register, feedback, design, tests, testability, programming

1. INTRODUCTION

Linear feedback shift registers are of large importance in diagnostics. It is possible to find structures of test pattern generators and test response compactors basing on LFSR in many papers. Their widest description is presented in study by Hławiczka (1997). In Badura (1992) the possibilities of non-linear feedback shift registers applying in BIST structures are discussed. In respect of designing for this group of registers it is relatively difficult to obtain essential features such as: cyclical graph as well as graph with the longest cycle. Gościniak (1996) discusses the possibility of linear modification of non-linear registers (creating the hybrid register) what improves their diagnostic properties. The possibility of genetic algorithm using for the designing of linear feedback shift register and linear modification of hybrid registers is presented in the paper by Gościniak and Chodacki (2003). The results obtained in the last-mentioned paper (Gościniak, Chodacki, 2003) determine to make attempt to design non-linear feedback shift registers using genetic methods. Genetic programming described by Koza (1992) gives positive results in electronic circuits designing. The below-presented article presents the results of investigations obtained during non-linear feedback shift registers designing with applying of genetic programming.

2. FEEDBACK SHIFT REGISTERS

The general structure of feedback shift register can be presented as in fig. 1. The multi-output function of non-linear feedback can be divided into n single-output component functions. If component function is presented as the function built of XOR gates this function creates linear feedback. Register, which has the linear and non-linear functions of feedback, is called hybrid register.

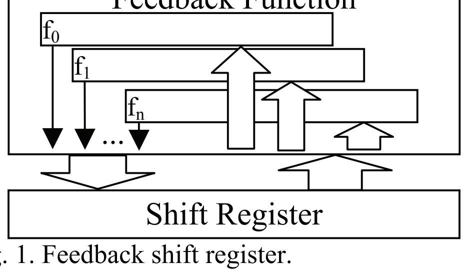

Fig. 1. Feedback shift register.

Feedback shift registers used in testing aims should have the following features:

- graph of register has the longest cycle or the longest path,
- graph of register is cyclic,
- graph of register has the cycle of given length,
- graph of register has the given sequence of states,
- graph of register has definite sequence of states in cycle of the given length,
- graph of register has states in growing sequence.

Additionally the possibility to design registers, which generate code combination by means of genetic programming, is checked (Wagner 1977).

3. FEEDBACK REGISTER GENERATING CODE COMBINATION

To calculate code combination of definite binary sequence the following algorithm (Wagner 1977) can be used:
To write down s - bits binary number (binary sequence), e.g. A = 0110b
To add to A the number formed through serial shift of one number position, receiving in next steps the sequence (Table 1):

Table 1. Scheme of code sequence calculation.

Bit 0	Bit 1	Code sequence
a_0=0	a_3=0	0
a_1=1	a_0=0	1
a_2=1	a_1=1	3
a_3=0	a_2=1	2

After adding the bit to check code combination. If it is essential (table is correct), the process of adding becomes finished. The received code combination is the fundamental sequence, so the register, which corresponds with it, will have the smallest necessary length to realise considered binary sequence.

4. THE GRAMMAR IN THE BNF REPRESENTATION AND DERIVATION TREES

The GPKernel library (Hurner, 1996) with proper for solved problems modifications was applied in investigations. The GPKernel Library generates functions basing on grammar described in configuration file. The BNF - Backus - Naur - Form is used as the representation form. An example of grammar is below-presented:

```
S := <fe>;
<fe> :=   "(" <f0> ")" |
"(" <f1> <fe> ")" |
"(" <f2> <fe> <fe> ")" ;
<f0> := "D1" | "D2" ;
<f1> := "NOT" ;
<f2> := "OR" | "AND";
```

The following example shows how to create expression (NOT(OR(D1)(D2))) from above-described grammar. Such created expression can be introduced as complete derivation tree presented in fig. 2.

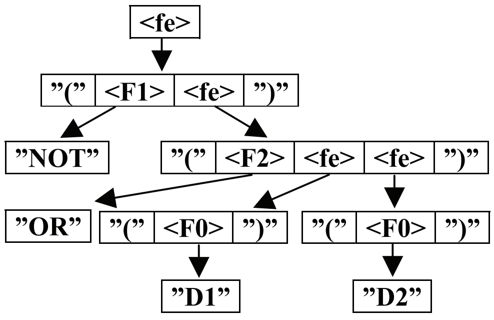

Fig. 2. Complete derivation tree for (NOT(OR(D1)(D2)))

Symbols in the corner brackets are non-terminal grammar ones. Symbols contained in quotation marks are terminal symbols. At applying of BNF rules, we can generate expression of this grammar. Rule formulated in the following way defines possible derivations for the non-terminal symbol.

5. GENETIC OPERATORS FOR DERIVATION TREES

Unlike genetic algorithms in genetic programming individuals of population are not made by the sequences of signs, but by the complete derivation trees of specific grammar. Using such operators like initialisation, cross-over or mutation on derivation trees is not reduced to simple function operating on signs, what is typical for genetic algorithms. After initial population creating the genetic operators form the next generations. This algorithm is repeated until the solution of given problem is not founded. Genetic operators, which are made in one step at the transition from old generation to new one are presented in fig. 3. In one generation-step the following operators are used:

- Fitness calculation
- Reproduction:
    - Selection
    - Sampling
    - Mating (finding a partner for every individual selected in the sampling-process).
- Cross-over
- Mutation
- Elitism-operator (copying the best individual)

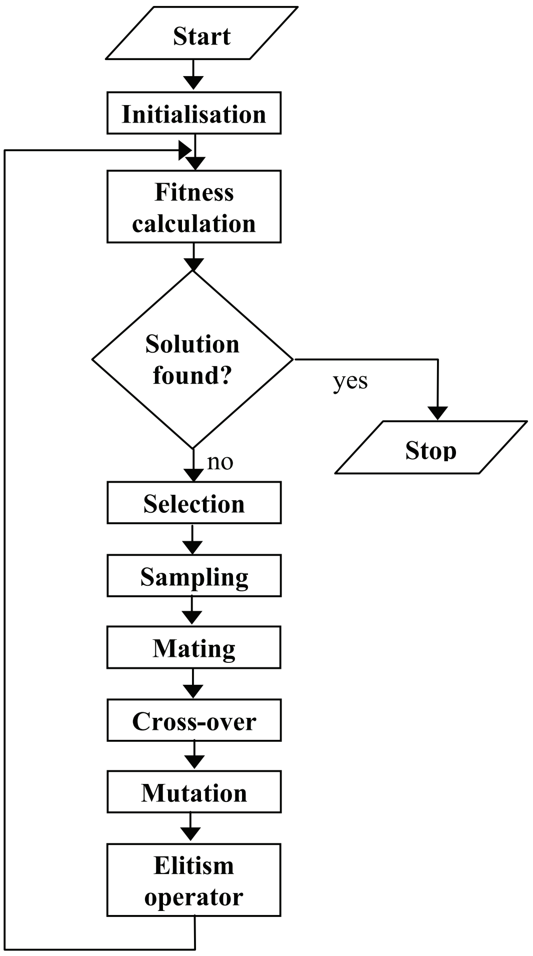

Fig. 3 Algorithm of genetic programming.

6. RESULTS OF INVESTIGATIONS

Starting the problem solving one should understand that several factors influencing favourable solution exist. Performed investigations show that well-chosen adaptation function is very important matter because of its influence on quality of solutions. Fitness functions, available in program, were tested with different parameters, what influenced the individuals building, and their number in population and diversity of population.
Ten-bite register was used in the following experiments.
The following grammars are used for function creating:

- grammar A
```
S := <fe>;
<fe> := "(" <f0> ")" |
"(" <f1> <fe> ")" |
"(" <f2> <fe> <fe> ")" ;
<f0> := "D1" | "D2" ;
<f1> := "NOT" ;
<f2> := "AND" | "OR";
```
- grammar B
```
S := <fe>;
<fe> := "(" <f0> ")" |
"(" <f1> <fe> ")" |
"(" <f2> <fe> <fe> ")" ;
<f0> := "D1" | "D2" | "D3" | "D4"  ;
<f1> := "NOT" ;
<f2> := "AND" | "OR";
```
- grammar C
```
S := <fe>;
<fe> := "(" <f0> ")" |
    "(" <f2> <fe> <fe> ")" ;
<f0> := "D1" | "D2" | "D3";
<f2> := "XOR";
```
- grammar D
```
S := <fe>;
<fe> := "(" <f0> ")" |
    "(" <f1> <fe> ")" |
      "(" <f2> <fe> <fe> ")" ;
<f0> := "D1" | "D2" | "D3" | "D4"  ;
<f1> := "NOT" ;
<f2> := "AND" | "OR" | "XOR";
```
- grammar E
```
    S := <fe>;
<fe> := "(" <f0> ")" |
      "(" <f1> <fe> ")" |
       "(" <f2> <fe> <fe> ")" ;
<f0> := "D1" | "D2" | "D3";
<f1> := "NOT";
<f2> := "XOR";
```

Investigation tool was worked out as diploma theses by Słota 2004 and Elmrych 2004

6.1 Graph of the longest cycle or the longest path

Parameters of genetic programming:
Maximum depth of derivation tree: 100;
Permissible maximum depth of derivation tree: 200
Population size: 10;
Probability of mutation: 0.03;
Probability of cross-over: 0.6;
Minimum for selection: 0.6;

Non-Linear Feedback Shift Register.
Connecting of feedback function: f0: output 0; inputs 1,2,5,6; grammar B.
```
f0->(AND(NOT(NOT(OR(NOT(D2))(AND(NOT(
NOT(D3)))(NOT(NOT(NOT(D3))))))))(OR(OR(OR(
NOT(NOT(D3)))(NOT(NOT(D1))))(NOT(D4)))(
NOT(NOT(NOT(AND(OR(D3)(AND(AND(AND(
OR(NOT(NOT(AND(D3)(NOT(AND(NOT(OR(D4)
(D1)))(NOT(D1))))))))(AND(OR(D3)(D3))(D4)))(
AND(NOT(OR(AND(NOT(NOT(D2)))(D1))(AND(
D4)(AND(D2)(D4))))))(D2)))(NOT(D4)))(NOT(D2))
))(D3))))))))
```

Linear Feedback Shift Register.
Connecting of feedback function: f0:-output 0; inputs 2,4,8; grammar 3;
```
f0->(XOR(D1)(XOR(XOR(XOR(XOR(XOR(D3)(
D3))(D2))(D1))(XOR(XOR(D2)(XD1)(D3))(XOR(
XOR(D2)(XOR(XOR(XOR(D3)(XOR(XOR(D2)(
XOR(D2)(XOR(XOR()(D3))))))(D1)))(D2))(D3)))(
D2))))))(D3)))(XOR(D2)(XOR(D2)(D3))))))
```

Hybrid Register
Connecting of feedback function: f0: output 0; inputs 1,2,5,6; grammar B.
Connecting of the feedback function: f1:-output 0; inputs 2,4,8; grammar 3;
f0->(OR(OR(OR(AND(NOT(AND(OR(AND(D4)(
NOT(D2)))(NOT(OR(D3)(D3))))(OR(AND(D1)(D4)
)(NOT(AND(NOT(AND(AND(NOT(D3))(NOT(
OR(D1)(OR(NOT(AND(D3)(D2)))(NOT(D4))))))(
D1)))(D3))))))(NOT(D2)))(D4))(OR(D1)(AND(NOT
(NOT(D3)))(D3)))))(NOT(D3)))
f1->(XOR(XOR(XOR(XOR(XOR(XOR(XOR(D1)(
XOR(D3)(D1)))(XOR(D3)(XOR(D2)(D3))))(D3))(
XOR(XOR(D2)(D2))(XOR(XOR(D3)(D2))(D2))))(
XOR(XOR(D1)(D2))(XOR(D1)(D1))))(XOR(XOR(
D1)(XOR(XOR(D2)(XOR(XOR(D1)(D3))(D3)))(
D2)))(D2)))(XOR(D3)(D2)))

6.2 Cyclical graph of transition

Non-Linear Feedback Shift Register.
Connecting of feedback function: f0: output 0; inputs 1,2,5,6; grammar B.
Connecting of feedback function: f1: output 1; inputs 4,8; grammar 1.
f0->(AND(D3)(OR(D1)(OR(D1)(OR(OR(D3)(D3))(
OR(OR(OR(D1)(AND(AND(AND(AND(NOT(NOT
(NOT(D3))))(OR(OR(D3)(NOT(D3)))(D4)))(AND(
AND(D4)(D4))(D3)))(D2))(NOT(D2))))(D3))(NOT(
NOT(NOT(NOT(OR(NOT(OR(D4)(D1)))(NOT(OR
(D2)(D3)))))))))))))
f1->(OR(D2)(AND(NOT(NOT(AND(OR(NOT(
AND(D2)(AND(NOT(NOT(OR(NOT(OR(D1)(NOT
(OR(NOT(D2))(D1))))))(AND(OR(D1)(NOT(D1)))(
OR(NOT(AND(D1)(D2)))(D2))))))(NOT(D2)))))(
OR(NOT(NOT(D1)))(OR(D2)(NOT(AND(D2)(D1))
)))))(D1))))(OR(D1)(D1))))

Hybrid Register
Connecting of feedback function: f0: output 0; inputs 1,2,5,6; grammar B.
Connecting of feedback function: f1:-output 3; inputs 2,4,8; grammar 3;
f0->(AND(D1)(AND(OR(D1)(NOT(D1)))(NOT(OR
(AND(OR(OR(NOT(OR(NOT(NOT(OR(OR(D4)(
D3))(NOT(NOT(D1))))))(D2)))(NOT(AND(NOT(
NOT(NOT(D2))))(D3))))(OR(D3)(AND(OR(NOT(
D3))(D4))(OR(NOT(D1))(D3)))))(NOT(D1)))(OR(
D4)(AND(D1)(D3)))))))
f1->(XOR(XOR(XOR(XOR(D3)(D3))(XOR(XOR(
XOR(XOR(D3)(XOR(XOR(D3)(D1))(D2)))(XOR(
XOR(XOR(D3)(XOR(D3)(D2)))(D3))(XOR(XOR(
XOR(XOR(D1)(XOR(D2)(D3)))(D2))(D3))(XOR(
D3)(D1)))))(D3))(D1)))(D1))(XOR(XOR(D3)(D2))
(D1)))

6.3 Graph with cycle of the given length

The length of cycle in graph is 100.

Non-Linear Feedback Shift Register.
Connecting of feedback function: f0: output 0; inputs 1,2,5,6; grammar B.
f0->(OR(NOT(OR(AND(AND(D3)(AND(NOT(D4
))(D4)))(D4))(NOT(D4))))(OR(OR(NOT(NOT(AND
(NOT(OR(D2)(OR(NOT(AND(AND(D2)(OR(AND
(AND(NOT(D3))(NOT(AND(D2)(D3))))(D3))(D4))
)(D1)))(NOT(D2)))))(NOT(D1)))))(D2))(OR(NOT(
D3))(AND(NOT(D1))(D2)))))

Hybrid Register
Connecting of feedback function: f0: output 0; inputs 1,2,5,6; grammar B.
Connecting of feedback function: f1:-output 3; inputs 2,4,8; grammar 3;
f0->(NOT(AND(AND(AND(NOT(OR(D2)(OR(D2
)(AND(NOT(AND(D3)(OR(AND(OR(D2)(OR(D2)
(NOT(OR(D2)(OR(D2)(AND(NOT(NOT(AND(
NOT(OR(D1)(D2)))(D1))))(OR(D2)(NOT(OR(D2)
(OR(D2)(AND(NOT(NOT(AND(NOT(OR(D1)(D2)
))(D1))))(AND(D1)(D2)))))))))))))(NOT(D2)))(D1)))
)(AND(D1)(D2))))))(NOT(D4)))(AND(NOT(AND(
NOT(OR(D1)(D2)))(D1)))(D3)))(AND(NOT(AND(
NOT(OR(D1)(D2)))(D1)))(D3))))
f1->(XOR(D3)(XOR(XOR(D2)(D2))(XOR(XOR(
XOR(D1)(D3))(XOR(D1)(XOR(XOR(D1)(D3))(
XOR(D1)(XOR(D3)(XOR(XOR(D3)(D2))(D2)))))))
(D3))))

6.3 The graph has the given sequence of states.

The graph contains the following sequence of states: 135,267,530,37,79,154,308,617;
Connecting of feedback function:f0 output 0; inputs 3,2; grammar D;
Connecting of feedback function:f1 output 1; inputs 5,8; grammar D;
Connecting of feedback function:f2 output 2; inputs 7,9,4; grammar E;

Parameters of genetic programming:
Maximum depth of derivation tree: 30;
Permissible maximum depth of derivation tree: 200
Population size: 30;
Probability of mutation: 0.03;
Probability of cross-over: 0.6;
Minimum for selection: 0.6;
f0->(NOT(D1))
f1->(AND(NOT(D1))(D1))
f2->(NOT(D3))

The graph contains the following sequence of states: 59, 113, 228, 463, 921, 817, 608, 194, 384, 774;
f0->(OR(D1)(D2))
f1->(NOT(NOT(OR(D1)(AND(AND(NOT(D1))(
OR(NOT(NOT(NOT(NOT(D2)))))(D1)))(OR(D1)
(D2))))))
f2->(NOT(D2))

6.4 Graph contains definite sequence in cycle of the given length.

Cycle in graph contains 12 states and sequence: 59,113,228,463,921,817,608,194.
Parameters of genetic programming and connecting of feedback function of register are as in point 6.3.
f0->(OR(OR(D1)(NOT(NOT(OR(D1)(AND(NOT (D2))(AND(D2)(D2)))))))(NOT(NOT(D2))))
f1->(OR(D2)(AND(OR(NOT(D2))(OR(AND(D1)(NOT(NOT(D1))))(D1)))(NOT(NOT(D1)))))
f2->(NOT(NOT(NOT(D2))))

6.5 Graph has the sequence of growing values

The graph has 7 states in growing relation, beginning from 10.
Connecting of feedback function:f0 output 0; inputs 5,8; grammar D;
Connecting of feedback function:f1 output 1; inputs 7,9,6; grammar E;
Maximum depth of derivation tree: 30;
Permissible maximum depth of derivation tree: 200
Population size: 30;
Probability of mutation: 0.02;
Probability of cross-over: 0.7;
Minimum for selection: 0.6;
f0->(OR(AND(NOT(D1))(D2))(OR(NOT(D2))(NOT (OR(D2)(NOT(D2))))))
f1->(NOT(XOR(XOR(NOT(XOR(D1)(NOT(D1)))) (D1))(NOT(D1))))

6.6 Code combinations

Two-bite register was used. To create function grammar D was used.
Connecting of feedback function:f0 output 0; inputs 0,1,0,1; grammar D;
Connecting of feedback function:f1 output 1; inputs 0,1,1,0; grammar D;
Maximum depth of derivation tree: 50;
Permissible maximum depth of derivation tree: 200
Population size: 100;
Probability of mutation: 0.02;
Probability of cross-over: 0.7;
Minimum for selection: 0.6;
Solution realising code sequences presented in point 3 was searched.
f0->(NOT(XOR(XOR(NOT(NOT(NOT(NOT(XOR (OR(D2)(D2))(D1)))))))(NOT(XOR(D2)(NOT(NOT(D4))))))(D3)))
f1->(OR(NOT(AND(AND(AND(AND(D4)(NOT(NOT(NOT(D3))))))(D4))(XOR(NOT(D3))(AND(D3)(D4))))(AND(D1)(NOT(NOT(NOT(D3)))))))(D1))

7. CONCLUSIONS

From performed investigations results that genetic programming is the proper tool to design non-linear feedback registers in the considered scale. One should pay attention to the fact, that start parameters for genetic algorithm as well as the way of connection of feedback function of register are of essential importance in searching solution. Improperly chosen start parameters can cause that solution will be never found or the process of solution searching will be very long. At initial population creating, one should take the depth of derivation trees as well as maximum depth into account because these parameters have large influence on created individual, e.g.: through cross-over. The kind of used logical function also plays the large role. Investigations show that applying XOR at feedback creating gives much better results. The special attention should be paid into the way of connecting of feedback function of register; i.e. to which register cell this function was connected. At such type of problems the conclusion is that ideal situation can be that, in which feedback, and more precisely the way of connecting feedback function to register can be also chosen by genetic algorithm.

REFERENCES

Badura D. (1992). *Techniki projektowania samotestowalnych układów i pakietów cyfrowych wykorzystujące rejestry szeregowe z nieliniowym sprzężeniem zwrotnym,* Uniwersytet Śląski.

Elmrych M. (2004). *Metody genetyczne w projektowaniu rejestrów z nieliniowym sprzężeniem zwrotnym*, Uniwersytet Śląski, Sosnowiec.

Gościniak I. (1996). *Liniowa metoda zwiększania efektywności diagnostycznej ścieżki samotestującej i pierścienia samotestującego*. Pomiary Automatyka Kontrola, nr 4, Warszawa.

Gościniak I., Chodacki M. (2003). *Genetic algorithms for the designing feedback shift registers,* 6th IEEE International Workshop on Design and Diagnostics of Electronic Circuits and Systems, Poznań, Poland, pp 301-302.

Hławiczka A. (1997). *Rejestry liniowe – analiza, synteza i zastosowania w testowaniu układów cyfrowych*, Wydawnictwo Politechniki Śląskiej, Gliwice.

Hürner H. (1996). *A C++ Class Library for Genetic Programming,* The Vienna University of Economics.

Koza J. R. (1992). *Genetic Programming: On the Programming of Computers by Means of Natural Selection*, MIT Press.

Słota T. (2004). *Metody genetyczne w projektowaniu rejestrów liczących*, Uniwersytet Śląski, Sosnowiec.

Wagner F. (1977). Projektowanie krótkich rejestrów liczących, Politechnika Śląska, ZN Nr. 526.