THE SYNTHESIS OF SELF-TEST CONTROL LOGIC

Oliver F. Haberl, Hans-Joachim Wunderlich

Institute of Computer Design and Fault Tolerance
(Prof. Dr.-Ing. D. Schmid)
University of Karlsruhe
Haid-und-Neu-Straße 7, D-7500 Karlsruhe 1

Abstract: In recent years, many built-in self-test techniques have been proposed based on feedback shift-registers for pattern generation and signature analysis. But in general, these test-registers cannot test several modules of the chip concurrently, and they have to be controlled by an external automatic test equipment.

The presented paper proposes a method to integrate the additional test control logic into the chip. Based on a register transfer description of the circuit, the test control is derived and an according finite automaton is synthesized. A hardware implementation is proposed, resulting in circuits, where the entire self-test only consists in activating the test mode, clocking and evaluating the overall signature.

1) Introduction

Most self-test techniques are implemented by multi-functional registers, which generate test patterns and compress test responses during special operation modes. The so-called BILBO (built-in logic block observer) based on standard linear feedback shift-registers (LFSR) generates pseudo-random patterns according to a uniform distribution [1]. Weighted patterns can be generated by a so-called GURT (generator of unequiprobable random tests) using modular LFSRs [2], providing a higher fault coverage by shorter test lengths. Many other BIST (built-in self-test)-registers have been proposed supporting a pseudo-random, deterministic or a (pseudo-) exhaustive test.

The benefits of a self-test are retained, if the resulting test schedule is not be controlled by an external test equipment, but the test control logic is integrated into the chip. Then the entire self-test will only consist in activating the test mode, clocking, and evaluating the signature.

The control logic initializes the involved test-registers for each sub-test, selects test-registers for pattern generation and for compressing test responses and unloads the signatures. Hence the necessary external test equipment only consists in an external shift-register and a clock generator.

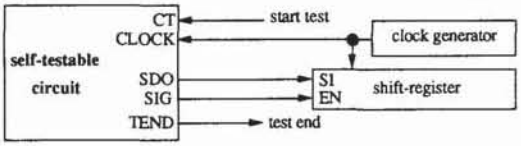

Figure 1: Self-test configuration

In this paper, we discuss the logic synthesis of the test control. We assume that the BIST-registers are already placed by methods described in [3], [4], or [5] e. g., and that a test schedule is already derived for instance by [6], [7] or [8]. Some implementations of an according test control unit have been discussed in [9]. Our approach synthesizes such a unit automatically, and it concentrates on an integrated self-test.

In section 2, the design restrictions of the proposed approach are discussed. They include the clocking scheme and the design style, and some requirements on the test-register placement are sketched.

In section 3, a method for test scheduling is discussed. It is shown, how a general test schedule can be described by a matrix.

Based on this matrix, in section 4 a microprogrammable control unit is synthesized. The test control logic generates all necessary control signals for the test-registers during the entire test phase. It will also indicate that the signature is available at a special output.

In section 5, alternatives to synthesize the hardware structure are discussed. Finally the entire approach is explained with the help of an example. For a circuit description at register-transfer level the test matrix is generated, and the control is synthesized.

2) Design restrictions

The presented approach can be applied to fully synchronous circuits. The circuit may be controlled by multiple, non-overlapping clock-phases, but we have to assume that all test-registers are clocked by the same phase.

There is no restriction on the test strategy (random [1], weighted random [2,10] or pseudo-exhaustive patterns [11,12]) implemented by the BIST-registers, but they have to work within four operation modes: In the system mode they are working as parallel registers, during the self-test either they generate test patterns or they evaluate the test responses, and they have to work like a shift-register. It is assumed that the test-registers have two control inputs b_0 and b_1 to select the operation modes given by the table 1.

b_0	b_1	mode
0	0	parallel
0	1	shift
1	0	generate
1	1	signature

Table 1: Operation modes of a test-register

At register-transfer level, these test-registers are placed dividing the design into partitions, which are tested by a set of test-registers generating the test patterns and by a disjoint set of test-registers evaluating the test responses. An obvious consequence is that every feedback loop of the data flow graph must be cut.

For board testing the approach can be augmented by a boundary scan. If no boundary scan is implemented, test-registers at the inputs and outputs are required.

3) Test scheduling

The self-test of the entire circuit is a collection of single tests, which are described by three parameters:

- a subset of registers generating test patterns;
- a disjoint subset of registers evaluating test responses;
- the number of patterns.

The overall test time is reduced, if several single tests are running concurrently. In general, the division of the global test into single tests is not unique, it is rather a complex optimizing problem to be solved during the placement of the test-registers. The single tests are collected to test sessions, consisting of compatible single tests without any inconsistent requirements on the operation modes of the involved test-registers.

Various scheduling techniques have been proposed ([3], [8] e.g.), all of them lead to a result that may be described by a matrix $A := (a_{i,j})$. It describes the operation modes of test-register $i := 1,...,t$ during the test session $j := 1,...,s$

$$a_{i,j} := \begin{cases} 0, & \text{if register } i \text{ generates patterns during test session } j \\ 1, & \text{if register } i \text{ compresses responses during test session } j \\ x, & \text{if register } i \text{ is not active during test session } j \end{cases}$$

Each column of A corresponds to a weight $\alpha(j)$, $j := 1,...,s$ denoting the number of patterns during test session j.

During the entire test all test-registers receive $b_0 = 1$, and only the signals at the b_1 inputs might differ. Table 2 describes whether two control values are compatible.

a_2 \ a_1	0	1	x
0	0	~	0
1	~	1	1
x	0	1	x

Table 2: Compatibility table; ~ denotes incompatibility

In matrix A each row corresponds to a test-register, and two rows can be merged if all pairwise components are compatible. The number q of rows in the new matrix $B := (b_{i,j})$ is the necessary number of control lines at the b_1 inputs. Of course, the objective function is a minimal number q of rows. The matrix B with according weights $\beta(j) := \alpha(j)$ completely describes the control flow during the entire test.

Each column b(j) describes a test session of length $\beta(j)$, and test-registers have a valid signature, if the corresponding control lines have a 1. These signatures can be shifted out either immediately after each session, or only once after the entire test. The latter simplifies the control logic, but it increases the aliasing probability slightly:

- If a register contains a valid signature, and if it is switched into the pattern generation mode, then no further aliasing can occur, since no further data is received;
- but if this register is switched into the data compression mode, then the data input sequence is prolonged, and the known estimations on aliasing probabilities [13], [14] are valid.

It should be noted, that this reasoning is only valid, if no register is doing pattern generation and signature analysis concurrently.

4) The control unit

The behavior described by B can be implemented by a control unit, providing a self-test architecture shown in figure 2.

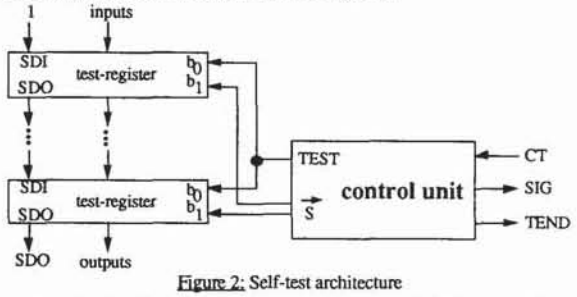

Figure 2: Self-test architecture

The signal CT (chip test) selects the self-test mode. The test-registers are connected in series by the SDI (scan data in) and SDO (scan data out) nodes. The scan data output of the last test-register feeds an additional primary pin SDO of the design, where the signature strings are shifted out. The signal SIG (signature) indicates that valid signatures are available, and the end of the entire self-test is marked by the signal TEND.

The test control unit of figure 2 communicates to the test-registers by the signal TEST connected to all b_0 inputs, and by the set of lines $\vec{S}$. The output of $\vec{S}$ during test session j is determined by the matrix B: $\vec{S}(j) = (b_{1,j},...,b_{q,j})$. This is implemented by a microprogrammable structure:

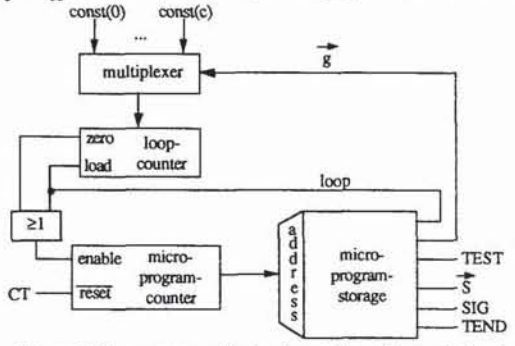

Figure 3: Microprogrammable implementation of the control unit

The loop-counter initializes the test-registers if necessary, controls the output of signatures and determines the duration of a test session. The variable c is the number of different constants, which are selected by the signals $\vec{g}$. The signal loop controls the timing of a loop: with loop = 1 the loop-counter is initialized with a constant selected by $\vec{g}$, and with loop = 0 the microprogram counter stops, until the loop-counter reaches zero.

Table 3 gives an overview of the instruction set. The parameter i is the address of a constant for initializing the loop-counter. The parameter j is the number of the test session.

instruction	loop	$\vec{g}$	TEST	$\vec{S}$	SIG	TEND
begin			0	0	0	
test(i,j)	1	i				0
	0	x	1	$\vec{S}$(j)	0	0
			1	$\vec{S}$(j)	0	
shift(i)	1	i				0
	0	x	0	1	0	0
			0	1	0	
sig(i)	1	i				0
	0	x	0	1	1	0
			0	1	1	
end	1	x				0
	x	x	x	x	x	1

Table 3: Instruction set

At the first address, the instruction *begin* controls the chip during the normal mode. The test sessions are realized by the instruction *test*. It selects the test-registers for generating test patterns or compressing test responses. The instructions *shift* and *sig* control shift operation, *sig* additionally indicates a valid signature at the pin SDO. The instruction *end* sets the signal TEND.

Each test session described by b(j) and β(j) can be realized by the instruction *test*. If a session is terminated by checking the signatures, the instructions *shift* and *sig* are used. Hence the microprogram, the necessary number c of constants const(i), and the constants can be created automatically by a straightforward algorithm GEN. The input of GEN are the matrices A and B, the number of test patterns for each test session and the length of each test-register (figure 4).

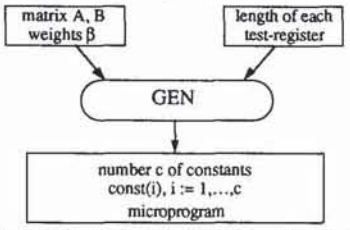

Figure 4: Generation of a microprogrammable structure

The output of GEN is now used to synthesize the control automaton. More efficient solutions are obtained by structures based on random logic or PLAs rather than by ROM-based solutions.

5) Efficient hardware solutions

In addition to the trade-off between hardware-overhead and aliasing probability, there is another trade-off between hardware-overhead and test length.

Many different constants control the shift and test operations. If we use only two integers determining upper bounds of these constants, the hardware is diminished drastically. Then the test control unloads the entire scan path for checking the signature, regardless of the operation modes of the respective test-registers, and the shifting clocks are extended until a power of 2 is reached. Moreover, we assign the same length to each test session, which should also be augmented to a power of 2.

We set υ(i) := length of test-register i, and we assume that the entire scan path at least has a length $\sum_{i=1}^{t} \upsilon(i) \geq 3$. We define $\omega := \lceil \log_2 (\sum_{i=1}^{t} \upsilon(i)-2) \rceil$ and set const(0) to the complement of 2^ω. Hence we have to count upward from const(0) in order to generate 2^ω clocks.

In a similar way, we determine $\zeta := \lceil \log_2 (\max_{1 \leq j \leq s} \beta(j)-2) \rceil$ as the general session length, and set const(1) to the complement of 2^ζ. By the reduction to a constant which is a power of 2 in a complementary representation, only an inverter is needed to create the constants, and the loop is implemented by an upward counter (figure 5).

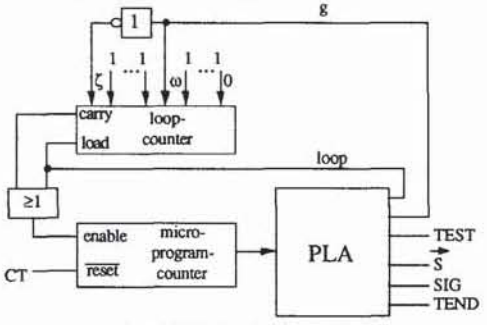

Figure 5: Efficient test control unit

It should be noted, that the program storage in figure 5 is implemented by a PLA generated by the program ESPRESSO [15]. ESPRESSO can directly use the microprogram code as an input, and produces an efficient hardware structure.

Up to now we have presented hardware solutions according to different trade-offs, as clarified by table 4.

I) Test length	II) Aliasing
1) Optimal constants	1) Signature checking after each test session
2) Two upper bounds of constants	2) Signature checking after the entire test

Table 4: Trade-offs

6) An example

The example is a circuit for matrix multiplication, accelerating some real time critical applications in robotics. It is a standard cell design, and it is described in [16] in detail. Figure 6 shows its register-transfer structure after the placement of the test-registers.

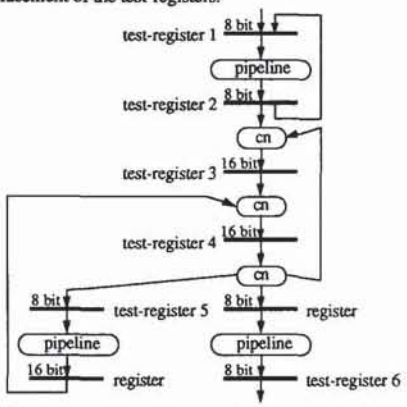

Figure 6: RT-representation of the example circuit with placed test-registers

We obtain a complete self-test by 2 test sessions:

A	test session 1	test session 2
test-register 1	0	1
test-register 2	1	0
test-register 3	0	1
test-register 4	1	0
test-register 5	0	1
test-register 6	x	1

Merging some rows of A, only q = 2 control lines are required, with $\vec{S}(1) = (0,1)$ for test session 1, and $\vec{S}(2) = (1,0)$ for test session 2:

B	test session 1	test session 2
test-register 1,3,5,6	0	1
test-register 2,4	1	0

We assume $\beta(1) = \beta(2)$ for the length of the test sessions, then the algorithm GEN generates the following sequence of microinstructions and constants for the loop-counter:

instructions	constants
begin	
shift(0)	$\upsilon(1)+\upsilon(2)+\upsilon(3)+\upsilon(4)+\upsilon(5)-2=54=const(0)$
test(1,1)	$\beta(1)-2=const(1)$
shift(2)	$\upsilon(5)+\upsilon(6)-2=14=const(2)$
sig(2)	$\upsilon(4)-2=14=const(2)$
shift(2)	$\upsilon(3)-2=14=const(2)$
sig(3)	$\upsilon(2)-2=6=const(3)$
shift(3)	$\upsilon(1)+\upsilon(2)+\upsilon(3)+\upsilon(4)-\upsilon(5)+\upsilon(6)-56-2=6=const(3)$
test(1,2)	$\beta(2)-2=const(1)$
sig(2)	$\upsilon(5)+\upsilon(6)-2=14=const(2)$
shift(2)	$\upsilon(4)-2=14=const(2)$
sig(2)	$\upsilon(3)-2=14=const(2)$
shift(3)	$\upsilon(2)-2=6=const(3)$
sig(3)	$\upsilon(1)-2=6=const(3)$
end	

Using the coding of table 3, this program can be stored at 28 addresses in a ROM. It can be minimized by ESPRESSO, in order to obtain a PLA-implementation by 20 product-terms.

We have presented the (I1, II1)-solution of table 4. Table 5 shows the necessary number of product terms for all four versions.

(I1,II1)	(I1,II2)	(I2,II1)	(I2,II2)
20	10	12	10

Table 5: Necessary number of product terms for the implementations by table 4

Especially the (I2,II2)-solution leads to a very small control unit (ζ:=10):

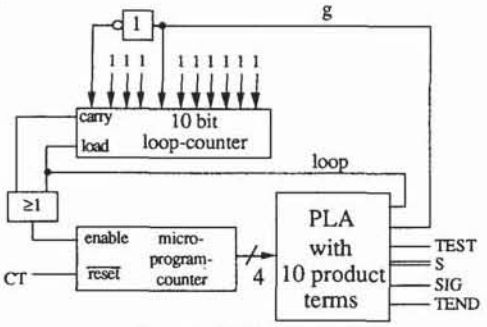

Figure 7: (I2,II2)-control unit

Conclusions

Methods have been presented to synthesize a self-test control unit after the placement of test-registers and the test scheduling.

If the test schedule is given in a straightforward matrix form, the presented algorithms automatically synthesize a finite automaton controlling the entire test. Different implementations of the control unit are derived, balancing the parameters of hardware-overhead, test-length and aliasing probability.

References

[1] B. Könemann, J. Mucha, G. Zwiehoff "Built-In Logic Block Observation Techniques" in Proc. IEEE Int. Test Conference, 1979, pp. 37-41

[2] H.-J. Wunderlich "Self Test Using Unequiprobable Random Patterns" in Dig. Int. Symposium on Fault-Tolerant Computing, FTCS-17, July 1987, pp. 258-263

[3] A. Krasniewski, A. Albicki "Self-Testing Pipelines" in Proc. IEEE Int. Conference on Computer Design, 1985, pp. 702-706

[4] A. Krasniewski, A. Albicki "Automatic Design of Exhaustively Self-Testing Chips with BILBO Modules" in Proc. IEEE Int. Test Conference, 1985, pp. 362-371

[5] H.-J. Wunderlich "The Design of Random-Testable Sequential Circuits" (submitted)

[6] J. Kalinowki, A. Albicki, J. Beausang "Test Control Signal Distribution in Self-Testing VLSI Circuits" in Proc. IEEE Int. Conference on Computer-Aided Design, 1986, pp. 60-63

[7] J. Beausang, A. Albicki "The Design for Testability Process: Definition and Exploration" in Proc. IEEE Int. Conference on Computer Design, 1987, pp. 362-365

[8] G. L. Graig, C. R. Kime, K. K. Saluja "Test Scheduling and Control for VLSI Built-In Self-Test" in IEEE Transactions on Computers, Vol. C-37, No. 9, Sept. 1988, pp. 1099-1109

[9] M. A. Breuer, R. Gupta, J.-C. Lien, J.-C "Concurrent Control of Multiple BIT Structures" in Proc. IEEE Int. Test Conference, 1988, pp. 431-442

[10] H.-J. Wunderlich "On Computing Optimized Input Probabilities for Random Tests" in Proc. 24th Design Automation Conference, 1987, pp. 392-398

[11] S. B. Akers "On The Use Of Linear Sums In Exhaustive Testing" in Dig. Int. Symposium on Fault-Tolerant Computing, FTCS-15, 1985, pp. 148-153

[12] L.-T. Wang, E. J. McCluskey "Circuits for Pseudo-Exhaustive Test Pattern Generation" in Proc. IEEE Int. Test Conference, 1986, pp. 25-37

[13] T. W. Williams, W. Daehn, M. Gruetzner, C. W. Starke "Comparison of Aliasing Errors for Primitive and Non Primitive Polynomials" in Proc. IEEE Int. Test Conference, 1986, pp. 282-288

[14] S. K. Gupta, D. K. Pradhan "A new framework for designing an analyzing BIST technique: computation of exact aliasing probability" in Proc. IEEE Int. Test Conference, 1988, pp. 329-342

[15] R. K. Brayton, G. D. Hachtel, C. T. McMullen, A. Sangiovanni-Vincentelli Logic Minimization Algorithms for VLSI Synthesis Boston: Kluwer Academic Publishers, 1984

[16] P. Gutberlet "Entwurf eines schnellen Matrizenmultiplizierers", Studienarbeit Fakultät Informatik, Universität Karlsruhe, 1988