Implementation of Parallel LFSR for BIST

M. K. Stojčev, I. Ž. Milovanović, E. I. Milovanović, T. R. Nikolić*

Abstract – **. Built in self test (BIST) is popular approach for VLSI testing. It uses a linear feedback shift register (LFSR) as test pattern generator since LFSR generates all possible test vectors which means it can achieve high fault coverage in a relatively short run test vectors. This paper shows the implementation of parallel LFSR (PLFSR) intended for testing intellectual property (IP) blocks within a VLSI IC. The PLFSR is connected to n IP's inputs in order to apply in-test mod test vectors which, possible, detect faults. The proposed PLFSR is implemented on Xilinx FPGA device, runs at 200 MHz clock frequency and generates two random numbers per clock period. the described design is reconfigurable and is capable of operating with different primitive polynomials of degree up to n=32. In respect to the standard LFSR, the proposed design shows that it can achieve an appealing trade off between performance ($2\times$ higher system throughput, from one hand, and less then 94% hardware overhead and dynamic power consumption, from the other hand.)**

Keywords – **Built-In Self-Test, Linear Feedback Shift Register, Random Number Generator, FPGA Design**

I. INTRODUCTION

Random numbers, RNs, are used today in numerous applications including electronic circuit testing, cryptography, simulations of wireless communication systems, Monte Carlo simulations, etc.[1]. Two basic approaches are used to generate RNs. The first one is based on measurement phenomenon of some physical process which is completely unpredictable such as thermal noise in electronic circuits, or noise-power level in radio-frequency receivers. Random number generators, RNGs, which use this principle of operation can be implemented using analogue and digital electronics, but these design solutions tend to be expensive and slow. The second approach uses computational algorithms that generate long sequences of apparently RNs. In this case RNs can be generated both by using software algorithms that involve complex mathematical operations and relatively slow RN sequences generation, and by using hardware which can implement less complex methods but fast RNs generation [2]. Up-to-date complex VLSI CMOS ICs run in the range from several hundreds MHz up to several GHz, so implementation of low-price, high-speed and simple RNG becomes an ultimate design goal. The RNG as electronic device is designed to generate a sequence of numbers that lack any pattern. But in practice it is very difficult, or almost impossible, to generate a series of logical steps that produce numbers that do not follow some definite sequence. These RNs are called pseudo random numbers, PRNs.

M. K. Stojčev, I. Ž. Milovanović, E. I. Milovanović, T. R. Nikolić are with the faculty of Electronic Engineering, Niš, Serbia.

There are two different hardware implementations of pseudo random number generators, PRNGs, that are widely used for logic built-in-self-test, BIST, applications [3]. The first one is based on usage of linear feedback shift register, LFSR. Its structure is simple, suitable for implementation as IP core within a complex VLSI ICs, and therefore is most commonly used to generate test patterns or test sequences [5-8]. The second design uses cellular automata, CA. The CA based PRNGs are more complex devices but provide patterns that look more random [2-3]. In this paper we have presented an efficient parallel pseudo random number generator based on LFSR (PLFSR) which is used for fast testing the constituents (IP cores) within a complex VLSI circuits. In respect to the parallel implementations of LFSR, described in [9-10], the proposed design characterizes higher system throughput and reconfigurability.

The rest of the paper is organized as follows. In section two a brief description of standard LFSR generators is given. Section 3 deals with principle of operation and structure of parallel LFSR generator. We start with mathematical background. After that we describe PLFSR hardware structure at block diagram level. Experimental results are given in Section 4. Concluding remarks are given in Section 5.

II. STANDARD LFSR GENERATORS

LFSR belong to the devices known as finite state machines. Its input state is a linear function of the pervious state. By using feedback it modifies itself on each rising edge of the clock. The L-bit initial value of LFSR is called seed, where L is called its length, and the bit position that affects next state is called tap. Two different LFSR implementations exist. The Fibonacci configuration (also known as external-XOR LFSR) consists of a simple shift register in which a binary-weighted modulo-2 sum of taps is fed back to the input , Fig. 1. The Galois implementation (alternatively called internal-XOR LFSR) consists of shift register, the contents of which are modified at every step by a binary-weighted value of the output stage, Fig. 2.

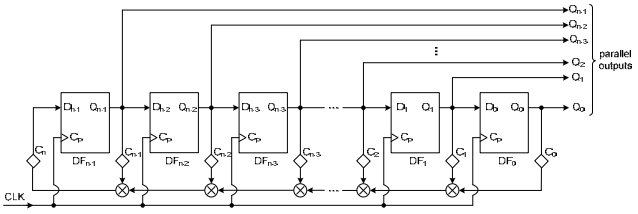

Figure 1: Fibonacci implementation of LFSR

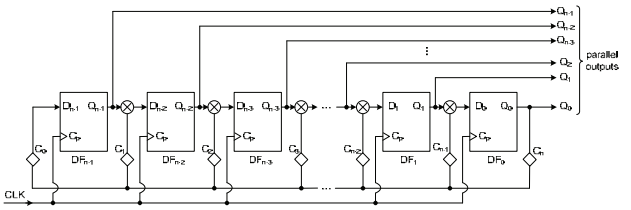

Figure 2: Galois implementation of LFSR

Codes generated by any of the two aforementioned types of LFSR are actually pseudo-random sequences because the sequence, known as the period of the PRNG, repeats after a certain number of clock cycles. Once it reaches its final state, it will traverse the sequence exactly as before. The advantage of serial LFSR architecture is small amount of hardware it requires.

With aim to increase the throughput of the LFSR generator, we propose a LFSR that generate k consecutive pseudo random numbers in parallel (PLFSR).

III. PARALLEL LFSR GENERATOR

Mathematical background

Let

$$P_n(x) = c_0x^0 + c_1x^1 + \cdots + c_{k-1}x^{k-1} + c_kx^k + \cdots + c_nx_n \quad (1)$$

be a feedback polynomial of degree n. The polynomial (1) has the following property

$$c_0 = c_n = 1,\ c_i = \{0,1\},\quad \text{and}\quad c_1 = c_2 = \cdots = c_{k-1} = 0.$$

In order to generate pseudo random number sequence of length n, polynomial (1) has to be a primitive one (see for example [4]). Let

$$\vec{Q}(0) = [q_1^{(0)} \quad q_2^{(0)} \quad \ldots \quad q_n^{(0)}]^T$$

Be a vector that corresponds to the initial state of the LFSR characterized by the polynomial (1). The next state of the LFSR can be obtained from

$$\vec{Q}(1) = A \oplus \vec{Q}(0)$$

where

$$A = \begin{bmatrix} c_1 & c_2 & \ldots & c_{n-1} & 1 \\ 1 & 0 & \ldots & 0 & 0 \\ 0 & 1 & \ldots & 0 & 0 \\ \vdots & & & & \\ 0 & 0\ldots & 0 & 1 & 0 \end{bmatrix}$$

is an $n \times n$ matrix joined with polynomial (1), and $\oplus$ a logical exclusive-or operation. In general, i-th state of the LFSR as a function of the initial state can be obtained according to

$$\vec{Q}(i) = A^i \oplus \vec{Q}(0),\quad where\quad A^i = A \oplus A^{i-1}.$$

Since $c_1 = c_2 = \ldots = c_{k-1}$, we can generate k consecutive pseudo random numbers in parallel, while keeping the same computational complexity for each of k feed-back results.

We will explain the idea on the example of the polynomial

$$P_5(x) = 1 + x^3 + x^5 \quad (2)$$

Matrix that corresponds to polynomial (2) is

$$A = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}$$

Let $\vec{Q}(0) = \begin{bmatrix} q_1^{(0)} & q_2^{(0)} & q_3^{(0)} & q_4^{(0)} & q_5^{(0)} \end{bmatrix}^T$ the initial state of the LFSR characterized by (2). The next four states can be obtained according to the following

$$\vec{Q}(1) = A \oplus \vec{Q}(0) = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix} \oplus \begin{bmatrix} q_1^{(0)} \\ q_2^{(0)} \\ q_3^{(0)} \\ q_4^{(0)} \\ q_5^{(0)} \end{bmatrix} = \begin{bmatrix} q_3^{(0)} \oplus q_5^{(0)} \\ q_1^{(0)} \\ q_2^{(0)} \\ q_3^{(0)} \\ q_4^{(0)} \end{bmatrix} = \begin{bmatrix} q_1^{(1)} \\ q_2^{(1)} \\ q_3^{(1)} \\ q_4^{(1)} \\ q_5^{(1)} \end{bmatrix}$$

$$\vec{Q}(2) = A^2 \oplus \vec{Q}(0) = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}^2 \oplus \begin{bmatrix} q_1^{(0)} \\ q_2^{(0)} \\ q_3^{(0)} \\ q_4^{(0)} \\ q_5^{(0)} \end{bmatrix} = \begin{bmatrix} q_2^{(0)} \oplus q_4^{(0)} \\ q_3^{(0)} \oplus q_5^{(0)} \\ q_1^{(0)} \\ q_2^{(0)} \\ q_3^{(0)} \end{bmatrix} = \begin{bmatrix} q_1^{(2)} \\ q_2^{(2)} \\ q_3^{(2)} \\ q_4^{(2)} \\ q_5^{(2)} \end{bmatrix}$$

$$\vec{Q}(3) = A^3 \oplus \vec{Q}(0) = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}^3 \oplus \begin{bmatrix} q_1^{(0)} \\ q_2^{(0)} \\ q_3^{(0)} \\ q_4^{(0)} \\ q_5^{(0)} \end{bmatrix} = \begin{bmatrix} q_1^{(0)} \oplus q_3^{(0)} \\ q_2^{(0)} \oplus q_4^{(0)} \\ q_3^{(0)} \oplus q_5^{(0)} \\ q_2^{(0)} \\ q_3^{(0)} \end{bmatrix} = \begin{bmatrix} q_1^{(3)} \\ q_2^{(3)} \\ q_3^{(3)} \\ q_4^{(3)} \\ q_5^{(3)} \end{bmatrix}$$

$$\vec{Q}(4) = A^4 \oplus \vec{Q}(0) = \begin{bmatrix} 0 & 0 & 1 & 0 & 1 \\ 1 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 \end{bmatrix}^4 \oplus \begin{bmatrix} q_1^{(0)} \\ q_2^{(0)} \\ q_3^{(0)} \\ q_4^{(0)} \\ q_5^{(0)} \end{bmatrix} = \begin{bmatrix} q_3^{(0)} \oplus q_5^{(0)} \oplus q_2^{(0)} \\ q_1^{(0)} \oplus q_3^{(0)} \\ q_2^{(0)} \oplus q_4^{(0)} \\ q_3^{(0)} \oplus q_5^{(0)} \\ q_1^{(0)} \end{bmatrix} = \begin{bmatrix} q_1^{(4)} \\ q_2^{(4)} \\ q_3^{(4)} \\ q_4^{(4)} \\ q_5^{(4)} \end{bmatrix}$$

According to the above equations we can conclude that computational complexity of determining states $\vec{Q}(1), \vec{Q}(2)$ and $\vec{Q}(3)$ is the same. On the other hand, determining state $\vec{Q}(4)$ requires two XOR operations for computing element $q_1^{(4)}$, leading to computational imbalance.

Parallel LFSR implementation

For the given feedback polynomial we propose LFSR for parallel generation of k consecutive pseudo random numbers, called parallel LFSR (abbreviated as PLFSR), shown in Fig. 3.

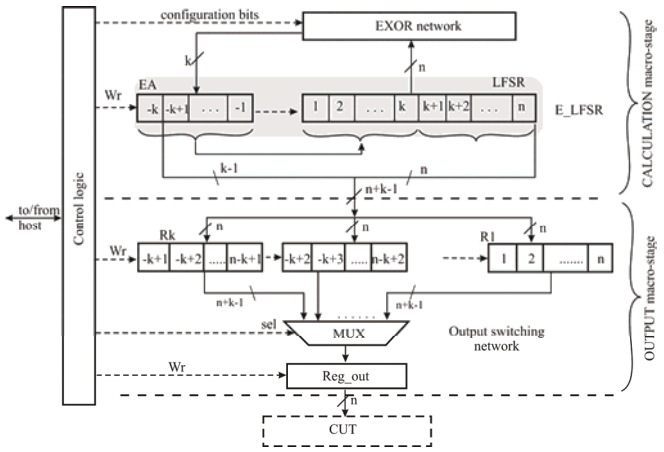

Figure 3: The structure of PLFSR

PLFSR consists of the following building blocks:

- Extended parallel shift register (E_LFSR) with $n+k$ cells (flip-flops). The right most n cells correspond to the standard LFSR, while the k leftmost cells represent linear array of k individual cells, called extension array (EA)
- The EXOR network is a combinatorial network of EXOR circuits which generate k product terms in parallel that feed in EA. Constituents of EXOR network are configuration registers that are used for selecting the primitive polynomial and the length of E_LFSR.
- k registers, R1 to Rk, composed of n flip-flops, used for temporal storing of k consecutive states of PLFSR
- Output switching network (MUX), which operates as k-input multiplexer.
- Output register, Reg_out, output stage used for driving Circuit Under Test (CUT)

Control logic (CL) used to generate control signals for driving the constituents of PLFSR.

The PLFSR is implemented as two-macro-stage pipeline. Within the first macro-stage, called CALCULATION, k consecutive pseudo-random sequences are calculated in parallel. The CALCULATION stage operates as two-cycle logic. During the first cycle, the content of E_LFSR is shifted for k positions right. In the second cycle, k consecutive resultant bits are calculated by EXOR network and written into EA. The second macro-stage, called OUTPUT stage, is implemented as k-cycle (poly phase) logic. During the first cycle registers R1 to Rk are loaded in parallel. In the next k cycles, k consecutive pseudorandom numbers are selected by MUX block and written into Reg_out.

During system initialization configuration bits are loaded into EXOR network. This provides that the proposed PLFSR can implement any feedback polynomial of degree n. In a concrete case, primarily limited by the amount of available logic blocks and input-output capacity of FPGA chips, we can implement, using reconfiguration, any polynomial of degree $n \le 32$.

The most complex part of PLFSR is the EXOR network, shown in Fig. 4. It consists of k reconfigurable EXOR blocks, each implemented as binary three of EXOR circuits. Multiplexers, M_1 to M_n, are used to switch on/off a corresponding tap in the feedback loop. The outputs of a configuration register are used for driving the select signals of multiplexers M_1 to M_n. The propagation delay of EXOR network is equal to the propagation delay through one multiplexer, plus $\log_2 n$ delay through EXOR circuits. It is independent of the chosen feedback polynomial for the given n.

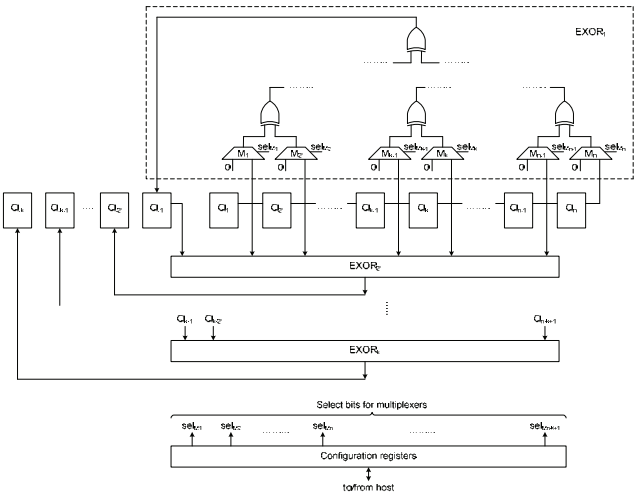

Figure 4: The structure of EXOR network

IV. EXPERIMENTAL RESULTS

In order to verify our design, we have implemented both PLFSR and standard LFSR in FPGA technology. For the sake of verification we have implemented PLFSR that generates two consecutive pseudo random sequences in parallel. The PLFSR and logics were described at register transfer level using VHDL. For FPGA implementation of PLFSR an LFSR we have used Xilinx development CAD tool ISE WebPack 13.1. Design verification was performed using test benches intended for excitation of PLFSR and LFSR. PLFSR and LFSR were implemented on FPGA devices from Virtex-6 LP series (circuit xc6vlx75tl-1lf484). The obtained results are given in Table 1, for standard LFSR, and in Table 2 for PLFSR.

Table 1: Implementation results for standard LFSR

	No of flip-flops in LFSR	No of occu-pied Slices	Best case achie-vable (ns)	Dynamic Power (mW)	Quiescent Power (mW)	Total Power (mW)
Virtex6 LP xc6vlx75tl-1Lff484	32	71	1.701	6.65	781.19	787.84
	24	52	1.885	5.99	781.18	787.17
	16	38	1.888	5.62	781.17	786.80
	8	18	1.722	4.51	781.16	785.67

Table 2: Implementation results PLFSR

	No of flip-flops in LFSR	No of occu-pied slices	Best case achie-vable (ns)	Dynamic Power (mW)	Quiescent Power (mW)	Total Power (mW)
Virtex6 LP xc6vlx75tl-1Lff484	32	113	1.883	11.17	781.26	792.43
	24	90	1.924	11.65	781.26	792.91
	16	57	1.768	9.19	781.23	790.42
	8	35	1.735	7.69	781.21	788.89

According to the results given in Tables 1 and 2 we can conclude the following:

- Hardware overhead of PLFSR compared to the standard LFSR is from 50% (for the polynomial of degree 16) up to 94% (for the polynomial of degree 8). For the given polynomial degree n, hardware overhead is independent of the chosen polynomial, i.e. active taps.
- Dynamic power consumption depends of the chosen primitive polynomial. For both PLFSR and standard LFSR power consumption was estimated using the same polynomial. The dynamic consumption ratio between PLFSR and standard LFSR, varies from 1.63 (for the polynomial of degree 16) up to 1.94 (for the polynomial of degree 24).
- Contribution of dynamic power consumption to the total power consumption is approximately 1.5% which implies that the impact of the PLFSR hardware is very low with respect to the total hardware of the FPGA chip.

Note that the system throughput of the PLFSR is two times higher compared to the standard LFSR under the same operating conditions, e.g. system clock.

V. CONCLUSION

Random number generation is an important application area that is met in BIST devices for complex VLSI circuits. In this case it is essential that the random numbers generator be amenable to hardware implementation in terms of area, high throughput rate, low-power dissipation, and low complexity. LFSR are commonly used as pseudo test pattern number generators in low overhead BIST schemes. In this paper we have presented an efficient parallel pseudo random number generator based on LFSR which is used for fast testing the constituents (IP cores) within a complex VLSI circuits. The proposed scheme was implemented on VIrtex6 LP FPGA device (circuit xc6vlx75tl-1lf484), running at clock speed of 200MHz, while delivering two 32-bit random numbers per clock. The reconfigurable hardware allows to implement any polynomial of degree 32. In comparison to standard pseudo random number generator based on LFSR, the proposed solution characterizes 2x higher throughput rate, at cost penalty of 94% of higher dynamic consumption and hardware overhead in the worst case.

ACKNOWLEDGEMENT

This work was supported by the Serbian Ministry of Education and Science, Project N0 TR-32009 –"Low power reconfigurable fault-tolerant platforms"

REFERENCES

[1] Random number generation, http://en.wikipedia.org/wiki/Random_number_generation, available Mart 2013

[2] Wijesinghe W.A.S., Jayananda M.K., Sonnadara D.U.J., Hardware Implementation of Random Number Generators, Proceedings of the Technical Sessions, Vol. 22, (2006), pp. 25-36, Institute of Physics-Sri Lanka, available at http://www.ip-sl.org/procs/ipsl063.pdf , Mart, 2013

[3] Design for testability, Laung-Terng (L.-T.) Wang, chapter 3, pp. 97- 172, in Electronic Design Automation: Synthesis, Verification, and Test, Eds. Laung-Terng Wang,Yao-Wen Chang, and Kwang-Ting (Tim) Cheng, Morgan Kaufmann Pub., Burlington, MA, USA

[4] N. Jha, S. Gupta, Testing of Digital Systems, Cambridge University Press, Cambridge UK, 2003.

[5] R. J. M. Nas, C. H. van Berkel, High Throughput, Low Set-up Time, Reconfigurable Linear Feedback Shift Registers, *IEEE International Conference on Computer Design* (*ICCD*), 3-6 October 2010, pp. 31-37

[6] N. Ahmed, M. H. Tehranipour, M. Nourani, Low Power Pattern Generation for BIST Architecture, *Proceedings of the 2004 International Symposium on Circuits and Systems* (*ISCAS'04*), Vol. 2, 2004, pp. 689-692

[7] E. A. Bezerra, F. Vargas, M. P. Gough, Improving Reconfigurable Systems Reliability by Combining Periodical Test and Redundancy Techniques: A Case Study, *Journal of Electronic Testing: Theory and Applications*, **17**(2) (2001), pp. 163-174

[8] A. Jhansirani, K. Harikishore, F. N. Basha, J. Poornima, M. Jyothil, M. Sahithi, P. Srinivas, Fault Tolerance in Bit Swapping LFSR Using FPGA Architecture, *International Journal of Engineering Research and Applications*, **2**(1), (2012), pp. 1080-1087

[9] M. Lowy, Parallel implementation of Linera Feedback Shift Registers for low power applications, IEEE Trans.on Circuits and Systems-II: Analog and Digital Signal Processing, Vol. 43, No. 6, 1966, pp. 458-466

[10] M.E. Hamid, and C.-I.H. Chen, A Note to Low-Power Linear Feedback Shift Registers, IEEE Trans.on Circuits and Systems-II: Analogue and Digital Signal Processing, Vol. 45, No. 9, 1998, pp. 1304-1307