Research Paper

Volume : 3 | Issue : 5 | May 2014 • ISSN No 2277 - 8179

A VLSI Implementation of Lfsr For Test Pattern With Bist Techniques

Engineering

KEYWORDS : Linear Feedback Shift Register, Iteration bound, Look Ahead Transformations, Loop unrolling

Sathyanathan.A — PG Scholar, Department of Electronics and Communication, Vedavyasa Institute of Technology, Karadparamba Kerala, India

ABSTRACT

This paper presents a VLSI implementation of LFSR for Test pattern with BIST techniques with Naïve, LAT and TePLAT algorithms. LFSR 10 is specified by generator polynomial over Galois Field. This paper examines a new algorithm called as Term Preserving Look Ahead Transformations (TePLAT). It converts bit serial algorithm to bit parallel algorithm with overhead. For bit parallelism the used technique is Loop Unrolling. LFSR 10is designed on various algorithms such as LAT, TePLAT. Hardware of LFSR 10 is implemented on Xilinx Spartan 3E.Simulation tool used for this LFSR 10 is Xilinx ISE simulator and this is modeled in Verilog HDL. Mainly bit level parallelism technique is used for speed up the process.

i. INTRODUCTION

In a Linear feedback shift register (LFSR) is a shift register whose input bit must be linear function of its earlier. In LFSR, linearly XORing single bits [6]. It is often a shift register input data must be driven by exclusive OR of other input bits.

LFSR are used for many applications like encryption, error correction coding, wireless communications, testing and random number generation [2],[3],[7]. A LFSR is formulated by generator polynomial over Galois field ie GF (2). For an initial condition, a linear sequence will be formed from the generator polynomial over Galois field. Several LFSR proposed in the literature [6]. Implementation of 20 LFSR's with characteristic polynomial degrees varying from 1 to 128forms a reconfigurable LFSR. In circuit implementation 128 flip flops that can be implemented one of 20 different types of LFSRs. For implementing on LFSR some XOR gates must be connected.

The flip flops are clocked for every clock cycle, so that power consumption in the serial architecture is high. The flip flops are serially connected; the output can be taken from the input of next flip flop. LFSR can be configured for a generator polynomial over Galois Field; it works like a serial filter. In every clock cycle output may be changed, so dynamic power consumption must be increased. So a new design is used to modify the LFSR to reduce the power consumption.

The vectorised compilation techniques such as loop unrolling and bit level parallelism are used to increase speed and reduce power consumption. The iteration bound means the inverse theoretical maximum throughput an algorithm can achieve. Many LFSR polynomials can be listed on Tables 3.1 and 3.2. Look Ahead Transformations (LAT) which resolves the difficulty in original LFSR implementation. In LAT, it produces additional operations and that it contains may more terms than original LFSR [8].

The evaluation and performance of all algorithms is tested and listed in Tables 3.1 and 3.2. Vectorised loop unrolling may be implemented in Xilinx Spartan 3E.

LFSR AND GENERATOR POLYNOMIALS

Implementation of LFSR can be done with characteristic polynomial $P(x) = 1+C_1x+C_2x.....+C_nx$ in Fig 1. If there is a connection exists then Ci=1, otherwise Ci=0. The serial architecture of characteristic polynomial $P(x) =1+x^3+x^{16}$ is called as LFSR 10 shown in Figure 2. In this the LFSR length which denotes flip flops as M=16 and number of tap coefficients which may be denoted as N=2.

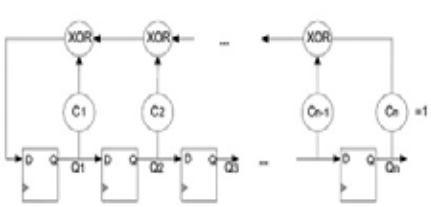

Figure 1 Serial LFSR

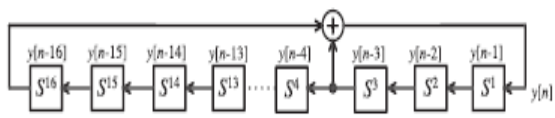

Figure 2 LFSR-10 with $P(x) =1+x^2+x^{16}$

An LFSR is formulated by Its generator polynomial GF(2)

$P(x) = 1+p_k.xk$ (1)

Where both x and $p_k \in \{0,1\}$ and also K is the order of generator polynomial p(x). Each generator polynomial forms a liner difference equation.

$y[n] + p_k.y[n-k]=0,$ (2)

Where $y[n] \in \{0,1\}$,"+" is the logical exclusive-OR operator, "." (Dot) is the logical AND operator. XORing both sides with y[n], has given

$y[n] = p_k.y[n-k]$ (3)

A LFSR described in equation (4) is termed as fsm of state as 1 x K binary vector [y[n-k]……y[n-1]] and illegal state of [0,0…….0]. Given the current state, next state can be obtained as [y[n-k+1]…….y[n]] will makes as left shifting the binary vector corresponds to the current state by 1 bit filling in the right most bit.

$y [n-K-1] = p_k.y[n-k-1]$ (4)

For generating LFSR 10, p0 = p3 = p16 =1, and

all others as zero ie pk = 0.

$P(x) =1+x^2+x^{16}$

The corresponding linear recurrent equation is

y[n]=y[n-3] ^y[n-16] (5)

TABLE I
Common LFSR polynomials

LFSR Index	Generator Polynomial
1	$1+x+x^3$
2	$1+x+x^4$
3	$1+x^2+x^5$
4	$1+x+x^6$

5	$1+x^4+x^5+x^6+x^7$
6	$1+x+x^5+x^6+x^8$
7	$1+x^4+x^9$
8	$1+x^3+x^{10}$
9	$1+x^3+x^4+x^7+x^{12}$
10	$1+x^3+x^{16}$

III. LOOP UNROLLING

unrolling or loop unfolding is a compiler optimization technique. It occurs in consecutive iterations into a single iteration through the bit level parallelism.

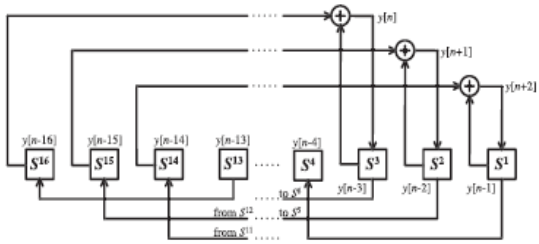

Fig 3 A Loop Unrolled version of LFSR-10

Implementation of LFSR produces bit binary values of y[n] as b bit register is called as naive LFSR implementation. If bit length is 32 or 64, it is tremendous wast because execution speed must be less. So most used approach in this case is bit level parallelism termed as loop unrolling.

When LFSR-10 as working as follows

1. for (n = 16; n < Bit Operated; n++){

2. y[n]= y[n -3] ^ y[n -16];

3. }

At each step. The output y[n] is calculated. The way of expressing when the algorithm is unrolled twice.

When LFSR-10 is unrolled two times

1. for (n = 16; n < Bit Operated; n+3){

2. y[n]= y[n -3] ^ y[n -16];

3. y[n+1]= y[n -2] ^ y[n -15];

4. y[n+2]= y[n -3] ^ y[n -14];

5. }

Now, construction of a 3-bit vector form

Y[n:n+2]= [y[[n]y[n +1] y[n +2]]

When LFSR-10 is unrolled two times, vectorized

1. for (n =16; n < Bit Operated; n = n +3){

2. Y [n : n + 2]= Y [n -3 : n -1] ^ Y [n -16 : n -14];

3. }

There are 3 output bits that are used in loop unrolling such that they are computed in single clock cycle. In LFSR 10, 3 fold speeds is performed by loop unrolling. So first of all calculate y[n], then y[n+1] to y[n+3] must be evaluated.

The inverse number of iterations that are unrolled into iterations simultaneously is called iteration bound. If the iteration bound is minimum such that throughput is maximum. In Fig. 3 iteration bound is 1/3. So the successive iterations that are formed are 3. When applying iteration bound, so that no needs to apply the loop unrolling.

IV. look Ahead transformations

Look Ahead Transformation (LAT) is used for reducing iteration bound. With k^* defined equation 3 expressed as

$$y[n]= y[n- k^*]+\sum p_k.y[n-k] \qquad (6)$$

Then substitute $n = n – k^*$ into (2)

$$0 = y[n –k^*] +\sum p_{m-k}{}^*.y[n-m] \qquad (7)$$

When substitute (7) into RHS of (6) and XOR as addition, LAT must be transformed as linear sequence equation as follows

$$z[n]=\sum q_m.z[n-m] \qquad (8)$$

V. TERM PRESERVING LOOK AHEAD TRANSFORMATION

Look Ahead Transformations is used to reduce iteration bound, also computation overhead that may affect the potential performance. So alternative LAT is used to solve the problem arises.

A TePLAT of LFSR with generator polynomial over Galois Field P(x) is a LFSR with generator polynomial $Q(x) = [P(x)]^2$, where P(x) is given as

$$[P(x)]^2 = \sum p_k x^{2k} \qquad (9)$$

TePLAT is an extension of bit level parallelism, used to produce pseudo random bit stream. When TePLAT is applied at generator polynomial over Galois Field must be doubled. The on chip storage area or memory is used to store the doubled data. When the TePLAT is used m times, such that it is called as m^{th} level TePLAT.

TePLAT LFSR-10, Unrolled, vectorised with overhead

1. lfsr_1 = Y [0]; //initialization, b0 b15 are copied to lfsr1

2. lfsr_2 = Y [1]; // b16 b31 in lfsr2

3. for (n =32; n < Bit Operated; n = n + 6) { //iteration overhead

4. tmp = lfsr 1 ^ (lfsr_2 «10); //arithmetic left shifting

5. tmp = tmp & 0xFC00; // aligning of data

6. lfsr_1 =(lfsr _1 «6)|(lfsr _2 »10);

7. lfsr_2 =(lfsr_ 2 « 6)|(tmp» 10);

8.}

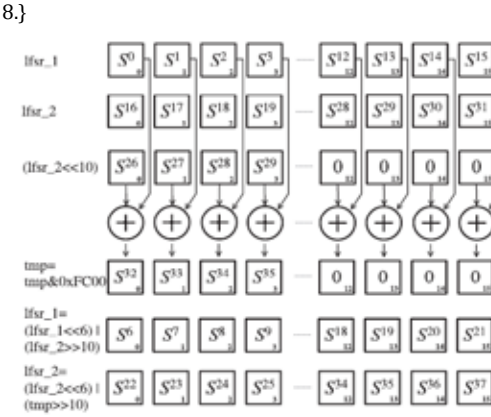

Figure 4 TePLAT, Loop Unrolled, Vectorised LFSR 10

Volume : 3 | Issue : 5 | May 2014 • ISSN No 2277 - 8179

VI. SIMULATION RESULTS

In LFSR hardware was developed to accelerate speed by bit level parallelism and was implemented on Xilinx Spartan 3E FPGA. In this paper Xilinx ISE simulator is used for simulations. In this paper generated LFSR codes with TePLAT for generator polynomial and also run the codes on the simulator correspondingly. In simulation bit level parallelism is performed by applying loop unrolling and also get TePLAT is better than any other technique.

Fig 5. Simulation of Loop unrolled vectorised version of LFSR 10

Fig 5. Simulation of Loop unrolled TePLAT version of LFSR 10

VII. CONCLUSIONS AND FUTURE WORKS

In this paper, TePLAT is used to perform bit level parallelism of a LFSR on Verilog simulators. The conventional LFSRs are similar to that applied loop unrolling technique. When existing approach is compared such that speed must be increased. LAT is used for hardware implementation of LFSR 10. Future research will focus on the areas like BIST, Wireless, Parity check and channel coding.

REFERENCE

[1] Jui-Chieh Lin, Sao-Jie Chen and Yu Hen Hu, "Cycle-Efficient LFSR Implementation On word based micro architecture," Proc. IEEE Transactions on Computers., vol. 62, no. 4, pp. 832-839, April 2013. | [2] G.C. Ahlquist, M. Rice, and B. Nelson, "Error Control Coding in Software Radios: An FPGA Approach," Proc. IEEE Personal Comm., vol. 6, no. 4, pp. 35-39, Aug. 1999. | [3] K.K. Saluja and C.-F. See, "An Efficient Signature Computation Method," IEEE Design and Test of Computers, vol. 9, no. 4, pp. 22-26, Dec. 1992. | [4] K.K. Parhi, VLSI Digital Signal Processing Systems Design and Implementation. John Wiley & Sons, Inc., 1999. | [5] Y. Tang, L. Qian, and Y. Wang, "Optimized Software Implementation of a Full-Rate IEEE 802.11a Compliant Digital Baseband Transmitter on a Digital Signal Processor," Proc. IEEE Global Comm. Conf. (GLOBECOM '05), vol. 4, pp. 2194-2198, 2005. | [6] S. Chowdhury and S. Maitra, "Efficient Software Implementation of Linear Feedback Shift Registers," Proc. Second Int'l Conf. Cryptology in India: Progress in Cryptology (INDOCRYPT '01), vol. 2247, pp. 297-307, 2001. | [7] C. Lauradoux, "From Hardware to Software Synthesis of Linear Feedback Shift Registers," Proc. IEEE 21st Int'l Parallel and Distributed Processing Symp. | (IPDPS '07), pp. 1-8, 2007. | [8] R.S. Katti, X. Ruan, and H. Khattri, "Multiple-Output Low-Power Linear Feedback Shift Register Design," IEEE Trans. Circuits and System I, vol. 53, no. 7, pp. 1487-1495, July 2006. | [9] J. Mitola III, Cognitive Radio Architecture. John Wiley & Sons, 2006. | [10] L. He,W. Liao, and M. S. Stan, "System level leakage reduction considering the interdependence of temperature and leakage," in Proc. DAC, Jun. 2004, pp. 12–17. | [11] D. A. Wood, M. D. Hill, and R. E. Kessler, "A model for estimating trace-sample miss ratios," in Proc. ACM SIGMETRICS, Jun. 1991, pp. 79–89. | |