Design and Analysis of Random Number Generator for Implementation of Genetic Algorithms using FPGA

Javad Frounchi Mohammad Hossein Zarifi Sanaz Asgari Far Hamed Taghipour
jfrounchi@tabrizu.ac.ir m_zarifi@tabrizu.ac.ir sanaz.asgarifar@gmail.com ha_taghipour1@yahoo.com

Microelectronic and Microsensor Research Lab, Faculty of Electrical and Computer Engineering, University of Tabriz, Tabriz, Iran

Key words: Random Number Generator, Genetic Algorithm, Field Programmable Gate Array.

ABSTRACT

In this paper we designed a new random number generator (RNG) based on LFSR. This random number generator can be used in Bluetooth encryptions and genetic algorithm implementation. The algorithm used to generate random number realized using simple circuit and implemented on a Virtex-4 LX25 FPGA from Xilinx. This designed block indicated a good sequence random numbers which is used in the genetic algorithm implementation for solving travailing salesperson problem (TSP) on FPGA.

I. INTRODUCTION

Random number sequences can be found in a large number of applications that include cryptography, genetic algorithms, Bluetooth encryptions and neural networks. Good random number generators are required to provide true source of randomness in applications where one has to model a physical process. For example, in a simulation of a wireless communication system there are several noise sources that need to be included. The most commonly used is Additive White Gaussian Noise, but there other analog impairments like phase noise, clock jitter, etc.

A wide variety of ingenious methods have been designed to generate random numbers. In computing, a hardware random number generator is an apparatus that generates random numbers from a physical process. Such devices are often based on microscopic phenomena such as thermal noise or the photoelectric effect or other quantum phenomena. These processes are, in theory, completely unpredictable, and the theory's assertions of unpredictability are subject to experimental test. A quantum-based hardware random number generator typically contains an amplifier to bring the output of the physical process into the macroscopic realm, and a transducer to convert the output into a digital signal.
Hardware random number generators are often relatively slow, and they may produce a biased sequence. Whether a hardware random number generator is suitable for a particular application depends on the details of both the application and the generator.

Most generators are software based and generally fall into one of these three categories:

- Linear Congruential Generator: Usually of the Form $X_i = (aX_{i-1} + b) \bmod m$, where *a*, *b*, and *m* are constants. This random number generator requires integer recursion thus it is expensive in hardware. Also, this generator is efficiently predictable when the constants *a*, *b* and *m* are known [1].
- Lagged Fibonacci Generator: Generally of the form $X_i = (X_{i-r} \otimes X_{i-s}) \bmod m$, where *r, s* and *m* are constants, $r > s$, and $\otimes$ could be any of the following binary operators, +, −, ×, *xor*. This generator requires the initial data set $X_1, X_2 \ldots$ Xn and depending on the choice of the binary operator might require integer recursion [1].
- Linear Feedback Shift Register Generator: They are based on the theory of primitive polynomials in the form $X^p + X^q + 1$. Given such a primitive polynomial and p binary digits, X1, X2… $Xp_{-1,}$ then $X_k = X_{k-p}$ xor X_{k-p+q}. This generator has been shown to exhibit lattice structures in the random number sequence generated [1].

In this work we designed a new random number generator by using LFSR generator which shows acceptable results to be used in genetic algorithms implementation. We have implemented our designed system on a Virtex-4 LX25 FPGA from Xilinx.

II. LFSR DESCRIPTION

A linear feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state. The only linear functions of single bits are xor and inverse-xor; thus it is a shift register whose input bit is driven by the exclusive-or (xor) of some bits of the overall shift register value. The initial value of the LFSR is called the seed, and because the operation of the register is

deterministic, the sequence of values produced by the register is completely determined by its current (or previous) state. Likewise, because the register has a finite number of possible states, it must eventually enter a repeating cycle. However, a LFSR with a well-chosen feedback function can produce a sequence of bits which appears random and which has a very long cycle.

One of the two main parts of an LFSR is the shift register (the other is the feedback function). A shift register is a device whose identifying function is to shift its contents into adjacent positions within the register or, in the case of the position on the end, out of the register. The position on the other end is left empty unless some new content is shifted into the register. The feedback function in an LFSR has several names: XOR, odd parity, sum modulo 2. The bits contained in selected positions in the shift register are combined in some sort of function and the result is fed back into the register's input bit. By definition, the selected bit values are collected before the register is clocked and the result of the feedback function is inserted into the shift register during the shift, filling the position that is emptied as a result of the shift.

An LFSR is one of a class of devices known as state machines. The contents of the register, the bits tapped for the feedback function, and the output of the feedback function together describe the state of the LFSR. With each shift, the LFSR moves to a new state. There is one exception to this -- when the contents of the register are all zeroes, the LFSR will never change state. For any given state, there can be only one succeeding state. The reverse is also true: any given state can have only one preceding state. For the rest of this discussion, only the contents of the register will be used to describe the state of the LFSR. A state space of an LFSR is the list of all the states the LFSR can be in for a particular tap sequence and a particular starting value. Any tap sequence will yield at least two state spaces for an LFSR. (One of these spaces will be the one that contains only one state -- the all zero one.) Tap sequences that yield only two state spaces are referred to as maximal length tap sequences. The state of an LFSR that is n bits long can be any one of 2^n different values. The largest state space possible for such an LFSR will be $2^n - 1$ (all possible values minus the zero state). Because each state can have only once succeeding state, an LFSR with a maximal length tap sequence will pass through every non-zero state once and only once before repeating a state.

For our application the best random number sequence is one that is very uniform, has very little correlation effects, requires minimal hardware, easy to use, and above all the "random sequence" must be completely reproducible.
The Random number generator chosen for this study is based on a one LFSR with the following connecting rule:
D1=Q8
D2=Q1
.
.
.
Dn=Q7

Where Q1,…,Q8 are the outputs and D1,..,D8 are the inputs. As shown in Figure 1, the random number generator is implemented using XOR and Dff. One of the outputs, Q1, is XORed with the output from the leftmost Dff, Q8. Then the last output is feedback into first Dff input. This circuit counts through 2^8-1 different non-zero bit patterns. With n flip-flops, 2^n-1 different non-zero bit pattern can be generated.

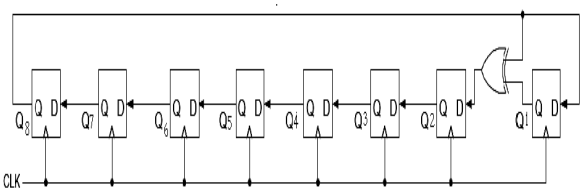

Figure 1. LFSR architecture

In general XORs are only ever 2-input and never connect in series. Therefore the minimum clock period for this circuit is $T > T_{2\text{-input XOR}}$ + clock overhead. The latency is very little and independent of n.

This design can be used as a random number generator that numbers appear in a random sequence repeats every 2^n-1 patterns. Also can be used fast counter, if the particular sequence of count value is not important such as micro-code micro-pc.

Mathematical analysis of LFSR

It is known that a Linear Feedback Shift Register LFSR associated with its characteristic polynomial G[x] of order *n* can generate a very good random like binary variable of periodicity 2^n-1 [2]. Associating *q* independent LFSRs generate a *q* bit variable *Uq* uniformly distributed over {0, 1, 2... 2*q*-1}. The LFSR design in FPGA need only *n* logic cells, each of them with its own register. Figure 2, illustrates the LFSR structure called "one to many" with the polynomial $x^5 + x^2 + 1$:

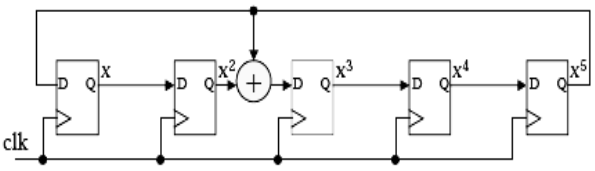

Figure 2. LFSR for $X^5 + X^2 + 1$

At every clock cycle, 4bits are used as outputs and "shifted". For instance for the LFSR of Figure 2, *t* being

the clock period, the register x^5 can be expressed as $x^5(t) = x^4(t\text{-}1) = x^2(t\text{-}3)+x^5(t\text{-}3) = x(t\text{-}4)+x4+(t\text{-}4)$.

By considering operations every 4*t*, 4 virtual shift operations are done in one clock cycle. This technique can be easily coded in VHDL and generates almost no extra FPGA logic cell.

III. RESULTS AND DISCUSSION

Figure 3, shows a schematic of the LFSR used in previous works.

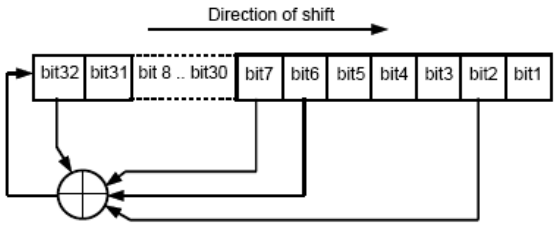

Figure 3. Logical Feedback Shift Register Random Number Generator

In previously works, the structure of LFSR RNG using the seed, have been studied. The obvious weakness of this type of RNG is that sequential values fail the serial test described by Knuth. At any time step *t* there is a 50% probability that the value at time *t* +1 can be predicted. If for an LFSR of length *n* at time *t* the value is *v*, then at time *t* +1 the value will be *v*/2 or $v=2/2^{n-1}$. This is shown in Figure 4, where pairs of values *vt* and *vt*+1 are plotted. It can be seen that for any value *vt* there are only two possible values of *vt*+1. Though the random number generator runs in parallel with the main GP machine, it is possible to access sequential values when creating an initial program, or when choosing crossover points.
There is then a possibility of a potentially degrading bias by using such an RNG [9].

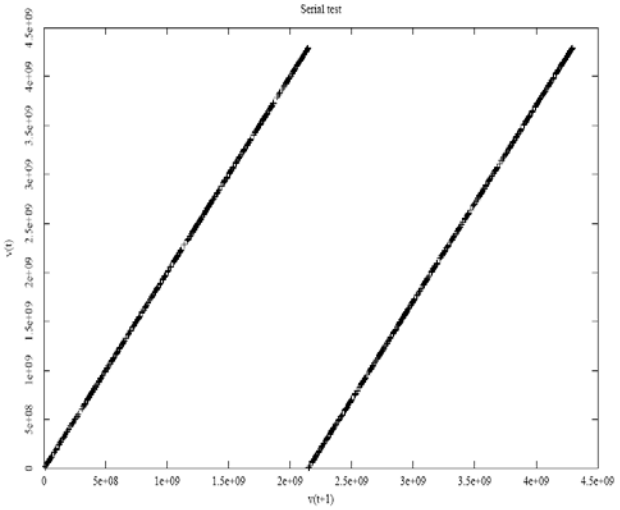

Figure 4. Serial test of a simple LFSR RNG

The proposed RNG structure has been illustrated in Figure 1, it decreases predictability of next number and generates exactly 2^n-1 various random numbers. Figure 5 shows the schematics of implemented structure on a Virtex-4 LX25 FPGA from Xilinx. The FPGA features 24,192 logic cell, 48 18*18-bit signed multipliers and 72 block Select RAMs. The Xilinx ISE 8.1i has been used to implement and synthesis the system.

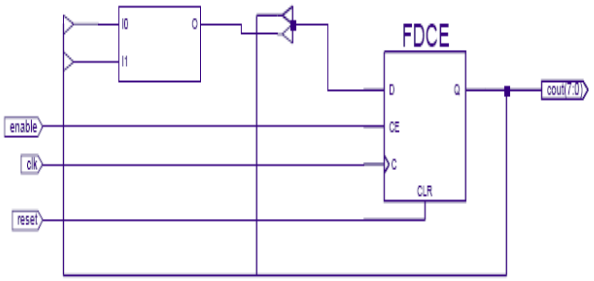

Figure 5. Implemented structure on FPGA

Table 1 shows the utilization of hardware resources on the chip. The floorplan of main system is shown in Figure 7 and RNG part is located. The remaining slices of the FPGA have been used for the implementation of next stages of genetic algorithms. The whole chip has been simulated using ModelSim 6.0 to evaluate its performance enhancement rate compared to the solving the TSP problem on MATLAB using a 2.4 GHz Pentium-4 based PC. The simulation results of the synthesized design of RNG part are illustrated in Figure 6.

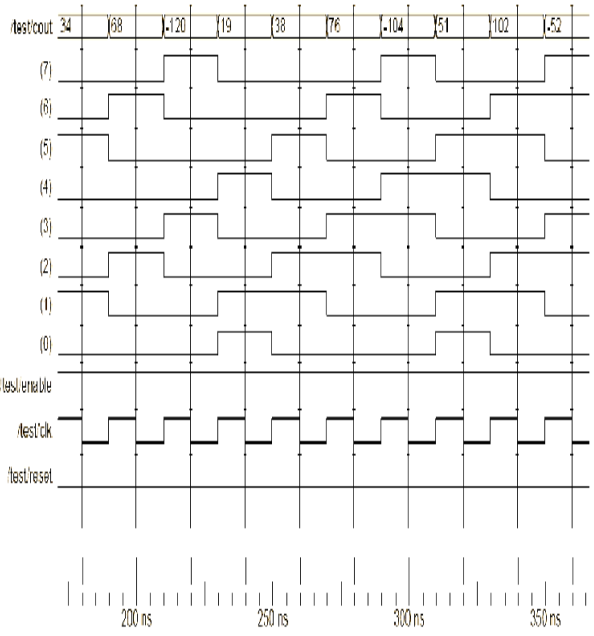

Figure 6. Simulation results of RNG part of chip

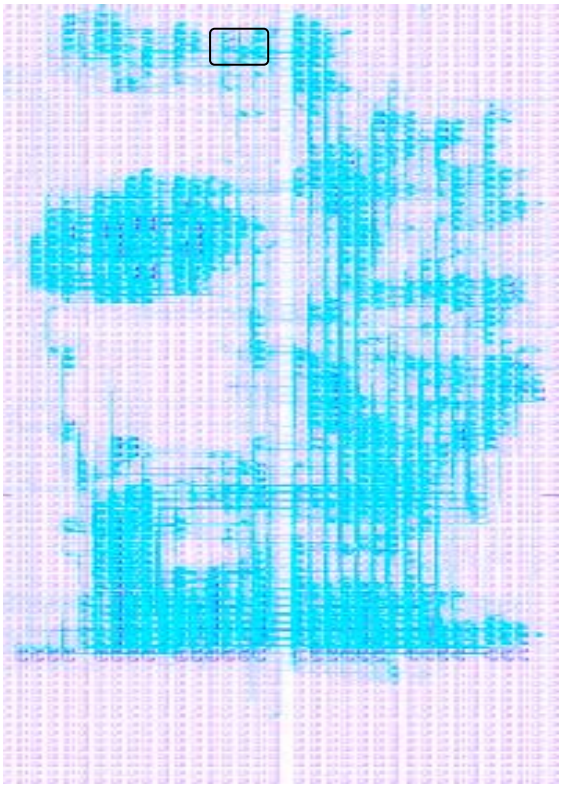

Figure 7. Floorplan of the main system, RNG part located

Table 1. Utilization of hardware resources for RNG part

Device Utilization Summary			
Logic Utilization	**Used**	**Available**	**Utilization**
Number of Slice Flip Flops	8	21,504	1%
Number of 4 input LUTs	1	21,504	1%
Logic Distribution			
Number of occupied Slices	4	10,752	1%
Number of Slices containing only related logic	4	4	100%
Number of Slices containing unrelated logic	0	4	0%
Total Number of 4 input LUTs	1	21,504	1%
Number of bonded IOBs	11	240	4%
Number of BUFG/BUFGCTRLs	1	32	3%
Number used as BUFGs	1		
Number used as BUFGCTRLs	0		
Total equivalent gate count for design	70		
Additional JTAG gate count for IOBs	528		

IV. CONCLUSION

Several different ways have already been examined to increase the number of randomness of random number generator. We proposed an RNG based on Field Programmable Gate Array which was used in genetic algorithms implementation that provides more randomness numbers than the previous works. Parallel generators of this kind can be used extensively to generate more randomness numbers with long lengths.

REFERENCES

1. I.Vattulainen, K. Kankaala, J. Saarinen, and T. Ala-Nissila, A Comparative study of pseudorandom number generators, Computer Phys. Comm. 86 (1995) 209-226
2. E.R. Berlekamp, "Algebric Coding Theory", McGraw-Hill
3. Paul Graham and Brent Nelson. Genetic algorithms in software and in hardware-a performance analysis of workstation and custom computing machine implementations. In Kenneth L. Pocek and Jeffrey Arnold, editors, *Proceedings of the Fourth IEEE Symposium of FPGAs for Custom Computing Machines.*, Pages 216–225, Napa Valley, California, April 1996. IEEE Computer Society Press.
4. Tsutomu Maruyama, Terunobu Funatsu, Minenobu Seki, Yoshiki Yamaguchi, and Tsutomu Hoshino. A Field-Programmable Gate-Array system for Evolutionary Computation. *IPSJ Journal*, 40(5), 1999.
5. J. Ackermann, U. Tangen, B. Bodekker, J. Breyer, E. Stoll, J.S. McCaskill, Parallel random number generator for inexpensive configurable hardware cells, Computer Phys. Comm 140(2001) 293-302
6. C. Aporntewan and P. Chongstitvatana. A Hardware implementation of the compact genetic algorithm. *IEEE Congress on Evolutionary Computation*, pages 624–629, May 2001.
7. Xilinx. Pseudo random number generator. www.xilinx.com/xcell/xl35/xl35_44.pdf, December 2001.
8. Wikipedia, Psuedorandom Number Generators, http://wikipedia.com/wiki/Pseudorandom_number_generator (2003)
9. Mustapha Abdulai, Inexpensive Parallel Random Number Generator for Configurable Hardware 2003.