Random Number Generator Using Various Techniques through VHDL

Jay Kumar[1], Sudhanshu Shukla[2], Dhiraj Prakash[3], Pratyush Mishra[4], Sudhir Kumar[5]

[1,2,3,4,5] *F.E.T. R.B.S. College, Agra (GBTU, Lucknow)*

[1]jaykumar_1981@yahoo.co.in
[2]sudhanshu.swastik@gmail.com
[3]engineerdhiraj@gmail.com
[4] prt.mis174@gmail.com
[5]sudhir.agra09@gmail.com

Abstract-- **Random numbers are useful for a variety of purposes, such as generating data encryption keys, simulating and modeling complex phenomena and for selecting random samples from larger data sets. They have also been used aesthetically, for example in literature and music, and are of course ever popular for games and gambling. When discussing single numbers, a random number is one that is drawn from a set of possible values, each of which is equally probable, i.e., a uniform distribution.**

Random numbers are generated by various methods. The two types of generators used for random number generation are pseudo random number generator (PRNG) and true random number generator (TRNG). In this paper we have used computational method through three techniques i.e., linear feedback shift register, linear congruental generator and blum blum shub.

It is simulated and synthesized using VHDL on the Xilinx ISE 9.1i

Keyword: **Random number generator, pseudonumber, VHDL, simulated, synthesized.**

I. Introduction

The objective of this paper is to create random number generator using various techniques. The random number generator produces a sequence of number which lacks any pattern, i.e. appear random. The many applications of randomness have led to the development of several different methods for generating random data. Many of these have existed since ancient times, including dice, coin flipping, the shuffling of playing cards, the use of yarrow stalks (by divination) in the I Ching, and many other techniques. Because of the mechanical nature of these techniques, generating large amounts of sufficiently random numbers (important in statistics) required a lot of work and/or time.

The randomizer can be created by physical method, computational method and from probability distribution method. Computational method is used in this paper for creating random number generator. Computational method produces Pseudo-random number generators (PRNGs), which are algorithms that can automatically create long runs of numbers with good random properties but eventually the sequence repeats.
Following three techniques are used in this paper named as Linear Feedback Shift register, Linear Congruential Generator and Blum Blum Shub.

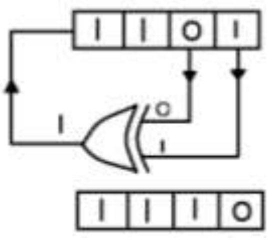
A. *Types of Pseudo Random Number Generator*

1) *Linear Feedback Shift Register:*

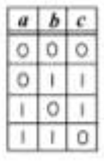
A linear feedback shift register (LFSR) is a shift register whose input bit is a linear function of its previous state.

The only linear function of single bits is 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 stream 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, an LFSR with a well-chosen feedback function can produce a sequence of bits which appears random and which has a very long cycle.



c = a XOR b



a	b	c
0	0	0
0	1	1
1	0	1
1	1	0

Fig.1 XOR operation for LFSR Fig.2 Truth table for XOR

Applications of LFSRs include generating pseudo-random numbers, pseudo-noise sequences, fast digital counters, and whitening sequences. Both hardware and software implementations of LFSRs are common.

i) 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 x5 + x2+1:

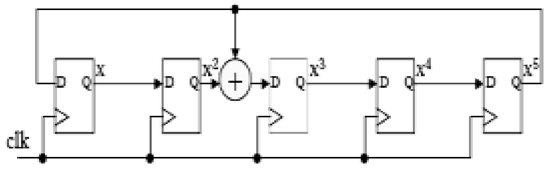


Figure 2. LFSR for X5 + X2 + 1

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 $x_5(t) = x_4(t-1) = x_2(t-3)+x_5(t-3) = x(t-4)+x4+(t-4)$. 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.

2) Linear Congruential Generator:

A Linear Congruential Generator (LCG) represents one of the oldest and best-known pseudo number generator algorithms. The theory behind them is easy to understand, and they are easily implemented and fast. However, their statistical properties are much worse than more recent generators, including generators with similar simplicity and speed.

The generator is defined by the recurrence relation:

$$X_{n+1} = (aX_n + c) \mod m$$

Where, *Xn* is the sequence of pseudorandom values, and

m, 0 < m— The "modulus"
a, 0 < a < m The "multiplier"
c, 0 ≤ c < m— The "increment" (the special case of *c* = 0 corresponds to Park-Miller RNG)
X_o*, 0 ≤* X_o *< m* The "seed" or "start value" are integer constants that specify the generator.

The period of a general LCG is at most *m*, and for some choices of *a* much less than that. Provided that *c* is nonzero, the LCG will have a full period for all seed values if and only if firstly *c* and m are relatively prime, secondly *a - 1*is divisible by all prime factors of *m*, and thirdly *a-1* is a multiple of 4 if *m* is a multiple of 4.

While LCGs are capable of producing decent pseudorandom numbers, this is extremely sensitive to the choice of the coefficients *c*, *m*, and *a*.

3) Blum Blum Shub:

Blum Blum Shub (BBS) is a pseudorandom number generator proposed in 1986 by Lenore Blum, Manuel Blum and Michael Shub (Blum et al., 1986).
Blum Blum Shub takes the form:

$$X_{n+1} = X_n^2 \bmod n$$

Where n=*p* × *q* is the product of two large primes *p* and *q*. At each step of the algorithm, some output is derived from X_{n+1}; the output is commonly the bit parity of X_{n+1} or one or more of the least significant bits of X_{n+1}. The two primes, *p* and *q*, should both be congruent to 3 (mod 4) (this guarantees that each quadratic residue has one square root which is also a quadratic residue) and gcd (φ(*p*-1), φ(*q*-1)) should be small (this makes the cycle length large).

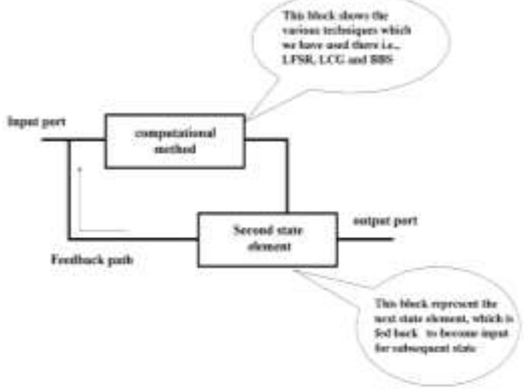

This block shows the various techniques which we have used there i.e., LFSR, LCG and BBS
Input port
computational method
Second state element
output port
Feedback path
This block represent the next state element, which is fed back to become input for subsequent state

Fig. 3 Block diagram for the program flow

II. VHDL

VHDL is an acronym for VHSlC Hardware Description Language (VHSIC is an acronym for Very High Speed Integrated Circuits). It is a hardware description language that can be used to model a digital system at many levels of abstraction ranging from the algorithmic level to the gate level. The complexity of the digital system being modelled could vary from that of a simple gate to a complete digital electronic system, or anything in between. The digital system can also be described hierarchically. Timing can also be explicitly modelled in the same description. The language not only defines the syntax but also defines very clear simulation semantics for each language construct. Therefore, models written in this language can be verified using a VHDL simulator. It is a strongly typed language and is often verbose to write. It inherits many of its features, especially the sequential language part, from the ADA programming language. Because VHDL provides an extensive range of modelling capabilities, it is often difficult to understand.

III. Result

A. LFSR

The above diagram shown is simulation diagram for LFSR which contain three input pins enable, clk &reset and one output pin, cout. When reset pin is at 'high (1)', no output is generated accept '0'. As soon as reset pin gets down to 'ground (0)', the random sequences appear to be generated but when enable and clk input pins are at 'high (1)'. Because of the finite state of shift register, these random sequence has a long repeated cycle.

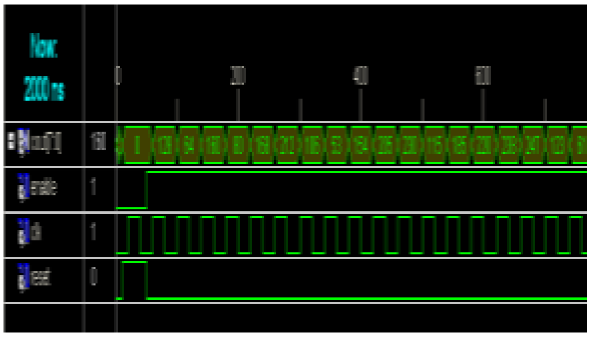


Fig.4 Simulation Diagram for LFSR

B. LCG

Figure shown below is the test bench waveform for linear feedback shift register in which clk represents the rising clock pulse given to the system; c_out is the port where we take our output. In this generator we have made two processes in which is to provide the delay and other for the the main program.

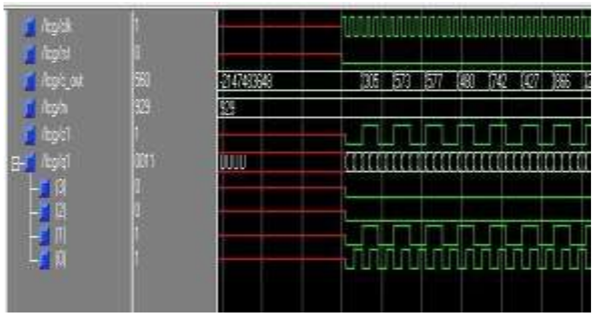


Fig.5 Simulation Diagram for LCG

C. BBS

The simulation diagram for blum blum shub is shown above. This diagram shows that the bit stream is generated which constitute a number. The output bit sequence are in random manner which is produced when least significant bit (LSB) is counted for the expression Xi = $(Xi-1)^2$ mod n for every value of i.

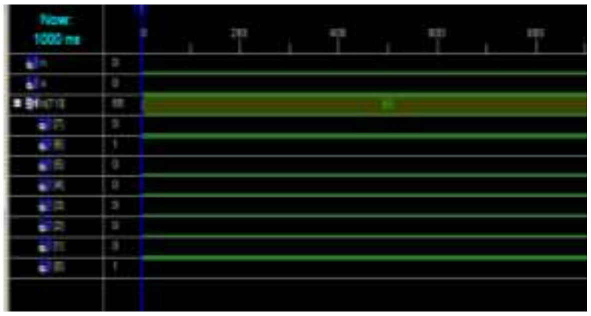


Fig.6 Simulation Diagram for BBS

IV. Conclusion

Finally the result have been displayed above in which we have found that linear feedback shift register is very low memory cost consuming, which has a seed value to initiate the random number generation. Linear feedback shift register have long cycle duration to generate random as true random number generator. It is more suitable for cryptography.

Blum blum shub generates random number generator more securely for the cryptograph but not memory cost efficient, it also uses seed value but with constraints.

Linear Congruential Generator found to be most in efficient techniques with maximum memory usage.

References

[1] Doughlas L. Perry "VHDL programming by example"
[2] E.R. Berlekamp, "Algebric Coding Theory", McGraw-Hill
[3] I.Vattulainen, K. Kankaala, J. Saarinen, and T. Ala- Nissila, A Comparative study of pseudorandom number generators, Computer Phys. Comm. 86 (1995) 209-226
[4] Mustapha Abdulai, Inexpensive Parallel Random
[5] Number Generator for Configurable Hardware 2003.
[6] 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.
[7] Peter J. Ashenden "VHDL tutorial"
[8] Tsutomu Maruyama, Terunobu Funatsu,
[9] Minenobu Seki, Yoshiki Yamaguchi, and Tsutomu Hoshino. A Field-Programmable Gate-Array system for Evolutionary Computation. IPSJ Journal, 40(5), 1999.
[10] Wikipedia, Psuedorandom Number Generators,
[11] wikipedia.com/wiki/Pseudorandom_number_generator
[12] Xilinx. Pseudo random number generator.
[13] www.xilinx.com/xcell/xl35/xl35_44.pdf, December 2001.