ISSN:0975-9646
M.Sahithi et al, / (IJCSIT) International Journal of Computer Science and Information Technologies, Vol. 3 (1) , 2012,3287-3290

Implementation of Random Number Generator Using LFSR for High Secured Multi Purpose Applications

M.Sahithi[#1], B.MuraliKrishna[#2],M.Jyothi [#3],K.Purnima[#4],A.Jhansi Rani[#5],N.Naga Sudha[#6]

[#1]*M.Tech student, Department of ECE, K L University*
Vijayawada, INDIA

[#2, 3, 4, 5,6]*Department of ECE, K L University*
Vijayawada, INDIA

[*]*B.MuraliKrishna, Department of ECE, KL University*
Vijayawada, INDIA

***Abstract*— Random numbers are required in a wide variety of applications. As digital systems become faster and denser, it is feasible, and frequently necessary, to implement random number generators directly in hardware. In this paper we present the 8-Bit random number generation using linear feedback shift register. This is very much useful in cryptography, data encryption and Circuit testing.**

***Keywords*— Linear feedback shift register, Cryptography, Circuit testing**

I. INTRODUCTION

Random numbers are required in a wide variety of applications, including data encryption, circuit testing, system simulation and Monte Carlo method. In the past, the random number generation was mostly done by software. However, as digital systems become faster and denser, it is feasible, and frequently necessary, to implement the generator directly in hardware. Although the software-based methods are well understood [1] [2] [3] [4], they frequently require complex arithmetic operations and thus are not feasible to be constructed in hardware.

Ideally, the generated random numbers should be uncorrelated and satisfy any statistical test for randomness. A generator can be either “truly random” or “pseudo random”. The former exhibits true randomness and the value of next number is unpredictable. The later only appears to be random. The sequence is actually based on specific mathematical algorithms and thus the pattern is repetitive and predictable. However, if the cycle period is very large, the sequence appears to be non-repetitive and random. Although it is possible to implement a true random number generator in hardware, it is slow and relatively expensive. In this paper we present 8-bit random number generator using linear feedback shift register. The whole design was captured in VHDL language and synthesized for a specific XILINX device

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.

One of the two main parts of an LFSR is the shift register (the other being 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. Two uses for a shift register are 1) convert between parallel and serial data and 2) delay a serial bit stream. The conversion function can go either way -- fill the shift register positions all at once (parallel) and then shift them out (serial) or shift the contents into the register bit by bit (serial) and then read the contents after the register is full (parallel). The delay function simply shifts the bits from one end of the shift register to the other, providing a delay equal to the length of the shift register. 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, 2n-1 different non-zero bit pattern can be generated.

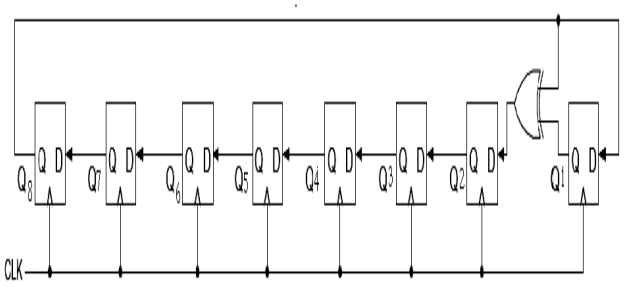

Fig .1. LFSR ARCHITECTURE

In general XORs are only ever 2-input and never connect in series. Therefore the minimum clock period for this circuit is T>T2-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 2n-1 patterns. Also can be used fast counter, if the particular sequence of count value is not important such as micro-code micro-pc.

One of the inputs to a shift register is the clock; a shift occurs in the register when this clock input changes state from one to zero. A shift register can shift its contents in either direction depending on how the device is designed. During a shift, the bit on the far right end of the shift register is moved out of the register. This end bit position is often referred to as the output bit. After a shift, the bit on the left end of the shift register is left empty unless a new bit is put into it.

One of the most frequent uses of a LFSR inside a FPGA is as a counter. Using a LFSR instead of a binary counter can increase the clock rate considerably due to the low routing resource required to produce the next state logic. In a sequential binary counter (i.e. counts 0, 1, 2, ...) the logic required for any particular bit has an input from all of the lesser significant bits and from its own register output. For example, MS bit of a 32-bit counter would require logic with a fan-in of 32. In a FPGA, with its limited fan-in for each macro-block, this would require many levels of logic hence reducing the maximum possible clock rate.

For a LFSR on the other hand, the maximum clock frequency is only limited by the propagation delay through the feedback logic which is usually not more than a couple of XOR gates. If the LFSR is floor planned correctly with each bit in adjacent macro-blocks, then a very fast counter can be realised.

The main problem with using LFSRs as counters is the pseudorandom nature of the sequence that they produce. In some applications this may not be acceptable, but for others, frequency division for example, it may not be important. Another problem is that the sequence length for a n-bit maximal LFSR is only 2^{n-1}, whereas the sequence length for a n-bit binary counter is 2^n. If we want to divide an input clock by 16, a 4-bit binary counter would be sufficient, but a 4-bit LFSR would not.

LFSRs have long been used as pseudo-random number generators for use in stream ciphers (especially in military cryptography), due to the ease of construction from simple electromechanical or electronic circuits, long periods, and very uniformly distributed output streams. However, an LFSR is a linear system, leading to fairly easy cryptanalysis. For example, given a stretch of known plaintext and corresponding cipher text, an attacker can intercept and recover a stretch of LFSR output stream used in the system described, and from that stretch of the output stream can construct an LFSR of minimal size that simulates the intended receiver by using the Berlekamp-Massey algorithm.

III.RANDOM NUMBER GENERATOR

A random number generator (often abbreviated as RNG) is a computational or physical device designed to

generate a sequence of numbers or symbols that lack 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. Thus, results would sometimes be collected and distributed as random number tables. Nowadays, after the advent of computational random number generators, a growing number of government-run lotteries, and lottery games, are using RNGs instead of more traditional drawing methods. RNGs are also used today to determine the odds of modern slot machines.

Several computational methods for random number generation exist, but often fall short of the goal of true randomness though they may meet, with varying success, some of the statistical tests for randomness intended to measure how unpredictable their results are. Random number generators have applications in gambling, statistical sampling, computer simulation, cryptography, completely randomized design, and other areas where producing an unpredictable result is desirable. Note that, in general, where unpredictability is paramount such as in security applications hardware generators are generally preferred, where feasible, over pseudo-random algorithms. Random number generators are very useful in developing Monte Carlo method simulations as debugging is facilitated by the ability to run the same sequence of random numbers again by starting from the same *random seed*. They are also used in cryptography so long as the *seed* is secret. Sender and receiver can generate the same set of numbers automatically to use as keys.

The generation of pseudo-random numbers is an important and common task in computer programming. While cryptography and certain numerical algorithms require a very high degree of apparent randomness, many other operations only need a modest amount of unpredictability. Some simple examples might be presenting a user with a "Random Quote of the Day", or determining which way a computer-controlled adversary might move in a computer game. Weaker forms of randomness are also closely associated with hash algorithms and in creating amortized searching and sorting algorithms.

TYPES OF RANDOM NUMBER GENERATORS

A. True Random Number Generator

A hardware (true) random number generator is a piece of electronics that plugs into a computer and produces genuine random numbers as opposed to the pseudo-random numbers that are produced by a computer program such as newran. The usual method is to amplify noise generated by a resistor (Johnson noise) or a semi-conductor diode and feed this to a comparator or Schmitt trigger. If you sample the output (not too quickly) you (hope to) get a series of bits which are statistically independent. These can be assembled into bytes, integers or floating point numbers and then, if necessary, into random numbers from other distributions using methods such as those in newran.

B. Pseudo Random Number Generator

These use a formula to generate numbers which behave very like genuine random numbers and are widely used for simulations of random processes and statistical methods. In most cases a good pseudo-random number generator seems to work as you would expect a genuine random generator to work. For a suite of programs for testing pseudo-random number generators and details of some pseudo-random number generators see George Marsaglia's Diehard tests. See also Taygeta Scientifics' notes on random number generators and the numerical analysis FAQ list.

Ideally, the generated random numbers should be uncorrelated and satisfy any statistical test for randomness. A generator can be either "truly random" or "pseudo random". The former exhibits true randomness and the value of next number is unpredictable. The later only appears to be random. The sequence is actually based on specific mathematical algorithms and thus the pattern is repetitive and predictable. However, if the cycle period is very large, the sequence appears to be non-repetitive and random. Although it is possible to implement a true random number generator in hardware, it is slow and relatively expensive. Security protocols and encryption algorithms are basically based on random number generators.

IV. PROPOSED MODEL

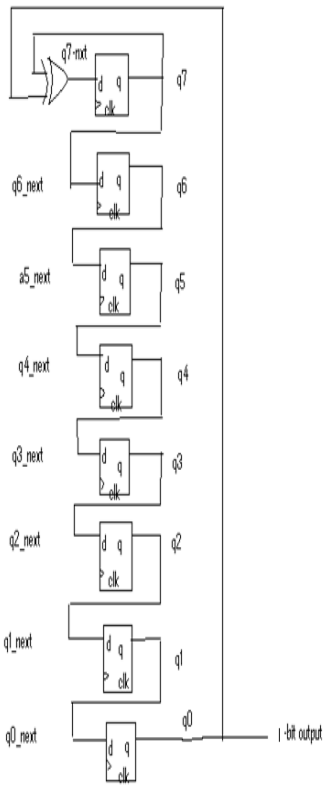

Fig.2. 8-BIT LFSR

From the above figure 8-BIT random number has been generated using "LINEAR FEEDBACK SHIFT REGISTER" .Here also we are using q7,q6,q5,q4,q3,q2,q1,q0 as outputs of the register and q0_next,q1_next,q2_next,q3_next,q4_next,q5_next,q6_next,q7_next as their next values and the boolean equation can be written as:

qo_next=q1
q1_next=q2
q2_next=q3
q3_next=q4
q4_next=q5
q5_next=q6
q6_next=q7
q7_next= q7 xor q0

This is the proposed model of generating 8-BIT random number using "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.

V.SIMULATION RESULTS

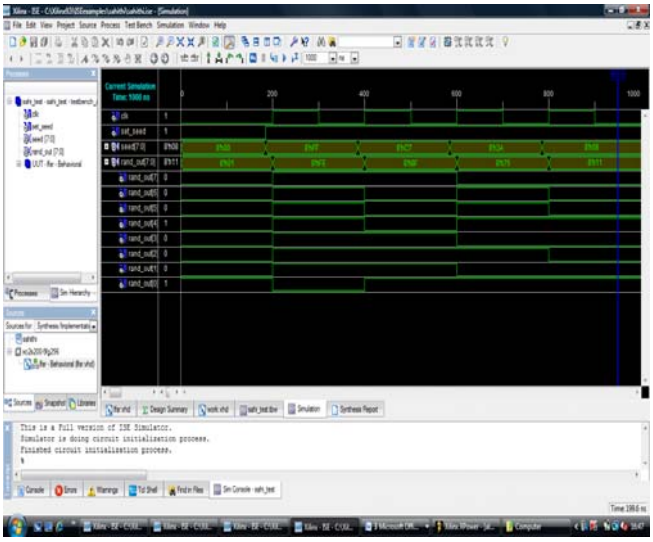

Fig.3 Simulation results for proposed model using LFSR

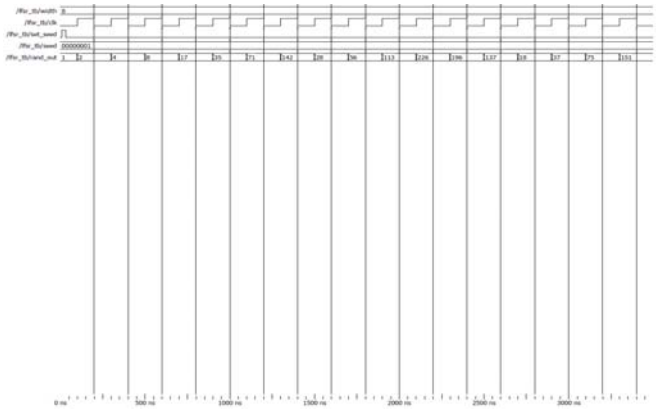

Fig.4.Simulation Results Using Modelsim

Here we present the random number generator using Linear Feedback Shift Register. From the above fig.4. it generates different patterns of random numbers. Here we get perfect randomness. By using Linear feedback shift register it can generate different patterns in less amount of time. This is mainly useful for security purpose. This is highly secure. The main applications are circuit testing, cryptography and also data encryption.Because of getting good randomness these are very much useful for security purpose. In the above figure we have taken the initial seed to generate different random numbers. From the initial seed only it starts to generate the different patterns to get the randomness and is very useful for security purpose. The lfsr can be used for counters and dividers.

VI.CONCLUSION

Several different ways have already been examined to increase the number of randomness of random number generator. We presented the random number generator using linear feedback shift register. For a single-bit random number generator, LFSR is the most effective method. When multiple bits are required, LFSR can be extended by utilizing extra time or extra circuitry. Cryptographic algorithms and communications protocols are based on random numbers generators.

vREFERENCES

[1] F. James, "A Review of Pseudo-random Number Generators, " Computer physics communication 60,1990.

[2] D.E. Knuth, *The Art of Computer Programming Vol. 2:Seminumerical* Method (2nd edition),Addision-Wesly,Reading Mass….1981.

[3] P. L'Ecuyer, "Random Numbers for Simulation," *Comm ACM* 33:10, 1990.

[4] G.A. Marsaglia, "A Current View of Random Number Generators," Computational science And Statistics:The interface,ed..L Balliard Elsevie,Amsterdam,1985.

[5] Xilinx pseudo Random Number generator. www.xilinx.com December 2001.

[6] Wikipedia, Pseudorandom Number Generators, http://wikipedia.com. Pseudorandom number generator (2003).

[7] Mustapha Abdulai, Inexpensive Parallel Random Number Generator for Configurable Hardware 2003.