RESEARCH ARTICLE OPEN ACCESS

FPGA Implementation of Park-Miller Algorithm to Generate Sequence of 32-Bit Pseudo Random Key for Encryption and Decryption of Text Message

Mr. Rohith. S*, Mr. Bharatesh. N**
Department of Electronics and Communication Engineering Nagarjuna College of Engineering and Technology, Bangalore, Karnataka, India-562110

ABSTRACT
In today's world, all the e-commerce transaction happens over the internet. The data that is transferred over the internet needs to be secured before the transmission process happens. The proposed scheme is a FPGA implementation of Park-Miller algorithm for generating sequence of Pseudo-Random keys. The generated pseudo random sequences are comparatively less complex in computation, power efficient and flexible so that it makes the digital circuit faster and smaller. The operation of Park-Miller algorithm to generate pseudo random number uses 32-bit Booth multiplier for multiplication, 32-bit floating point divider for division and a finite state machine for the flow of Park-Miller algorithm operation. After generating a sequence of 32-bit pseudo-random numbers, it is used as a key to encrypt the text message which is known as cipher text. Using the same key to decrypt the encrypted data to get back original text message. The programming is done in Verilog-HDL and MATLAB to verify the result. The design is simulated in Modelsim (version 10.5). The proposed module is successfully synthesized using Cadence RTL compiler. Area and power of the design is computed for 180nm technology. Finally design is implemented on Spartan 6E FPGA.
Keywords **–** Cryptography security, Encryption, Decryption, Park-Miller Algorithm, Pseudo random bit generator

I. INTRODUCTION

Encryption is used to securely transmit the data in a open networks. The transmitted data may be text, image, audio, video form. Each type of data has its own features. Therefore different techniques should be used to protect confidential data from unauthorized access. The Pseudo-Random numbers are plays very important role in the communication system for protection of information from unauthorized access. The efficiency of the Pseudo Random Number Generator (PRNG) based cryptosystem is mainly based on the random key generated by its PRNG [1-5].

Depending on the application, three types of number sequences might prove as the "random numbers". A numbers generated by a truly random process is most desirable. This type of sequence is called a random-number sequence and the key problem is deciding whether or not the generating process is random. A more practical sequence is the pseudo-random sequence, a series of numbers generated by a deterministic process that is intended merely to imitate a random sequence but which, of course, does not correctly obey such things as the laws of large numbers. A quasi-random sequence is a series of numbers that makes no guaranty at being random but that has important predefine statistical properties of random sequences.

PRNGs are efficient periodic and deterministic. Efficient meaning that in a short time, they can produce many numbers. Deterministic, means that if the starting point in the sequence is known a given sequence of numbers can be regenerate at a later time [1]. Periodic, means that the sequence will sequentially repeated after specified period. While periodicity is hardly ever a desirable characteristic, modern PRNGs have a period that is so long that it can be ignored for most practical purposes.

Pseudorandom sequences [1] have been widely used in various fields like, navigation, including communications, radar technology, cipher technologies, remote control, measurements, and industrial automation etc. [2]. Pseudorandom sequences have been used in error-correcting codes [3], spread spectrum communication [4], [5] and system identification and parameter measurements [6], [7]. Other applications are found in surface characterization and 3D scene modeling [8]. The design of a general purpose pseudorandom sequence

generator has matured and has already been commercialized [9]-[11].

Pseudorandom sequences are series of 1's and 0's that lack any definite pattern and look statistically independent and uniformly distributed. The sequences are deterministic, but existence of noise properties similar to randomness [12]. In general, a pseudorandom sequence generator is usually made up of shift registers with feedback. By linearly combining elements from the shift register and feeding them back to the input of the generator, we can obtain a sequence of much longer repeat length using the same number of delay elements in the shift register. Therefore, these blocks are also referred to as a linear feedback shift register (LFSR) [13], [14]. The length of the shift register, the number of taps and their positions in the LFSR are important to generate pseudorandom sequences with desirable auto-correlation properties [15].

The outputs from above said PRNGs suffer from some problems they include; periods are shorter than expected for some seed states. Generated numbers have lack of uniformity of distribution, correlation of successive values, output sequence is poor dimensional distribution, The distances between where some values occur are distributed differently from those in a random sequence distribution.

Park-Miller algorithm is used to generate 32-bit sequences of key[16]. The Park-Miller algorithm satisfies the following three criteria to generate good pseudo random numbers: sequence is sufficiently Random, Sequence full period, efficient implementation with 32-bit arithmetic.

The generated sequence of 32-bit sequence keys are plays a important role in the case of stream cipher to encrypt and decrypt plain text. In cryptography, a stream cipher is asymmetric key cipher, where plaintext digits are combined with a pseudo random cipher digit stream (key stream). In a stream cipher each plain text digits is encrypted one at a time with the corresponding digit of the key stream, to give a digit of the cipher text stream. Finally the generated sequence of 32-bit pseudo random keys and by same sequence of keys used to encrypt and decrypt plain text. Programming of proposed scheme is done using verilog-HDL and simulated using Xilinx ISE simulator. Finally the scheme is implemented on Spartan 3E FPGA.

The organization of this paper is as follows. Section II describes algorithm design and architecture of the proposed PRNG, Section III describe the design of Encryption and Decryption block The implementation results and synthesis report is discussed in section IV. Finally, the conclusion is presented in the last section.

II. PARK-MILLER ALGORITHM

Park-Miller algorithm [16] is based on the linear congruential form:

$X_{n+1} = a\ X_n \bmod (2^{31}-1)$

Where 'n' sequence number. Pseudo code of the Park miller algorithm is given below.

```
a=168007;        (2 < a < M-1)
m=2147483647;    (2^31-1)
q=127773;        (m div a)
r=2836;          (m mod a)
Var
hi: =in_seed div q;
lo: =in_seed mod q;
test: =a*lo –r*hi;
if test > 0 then
Out_seed:= test;
random: =out_seed / m;
else
Out_seed:= test +m;
random: =out_seed / m;
end;
```

Steps of the Park miller algorithm are as follows:

1. Initialize the input in_seed and parameters a = 16807 m= 2,147,483,647, q = 127773, r = 2836.
2. Compute the value of hi = in_seed div q and lo = in_seed mod q.
3. Then compute the corresponding test value.
4. If test > 0, save test as new out_seed, otherwise save test + m.
5. Output the new out_seed.
6. Iterate, and let the out_seed be the new in_seed.

As this algorithm is designed for computer system, the limitation of the precision of the operation system is considered. It is designed for any system whose precision is 32 bits or larger. In order to avoid overflow error, the seed adds a maximum allowable number m in the 32 bits system at before the output in the event that the value of the seed is not positive, as is illustrated in above algorithm[16].

The Figure1 shows the flow chart of Park Miller algorithm. Here first 'm' (2^{31}-1), 'a' (16807), 'q' (127773), 'r' (2836), En=0 are initialized and In_seed is a user choice with length of 32-bit value. After initializing these values for first random generation En = 0 at that time, the variables hi and lo have to be calculated as In_seed div q and In_seed mod q respectively. Then another variable Test as

a*lo – r*hi, if this value is greater than zero then out_seed becomes Test, else out_seed becomes Test + m. Finally random output will calculated as out_seed divided by m. At this stage first random output is generated. For the next pseudorandom number generation, En = 1 now out_seed becomes In_seed or hi and lo will calculated as out_seed div q and out_seed mod q there after same steps will continue to generate sequence of pseudorandom number generation.

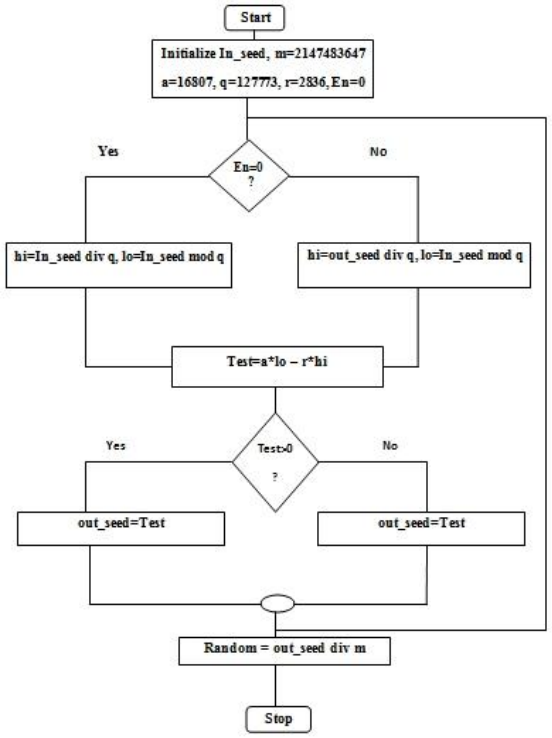

Fig 1: Flow Chart of Park Miller Algorithm

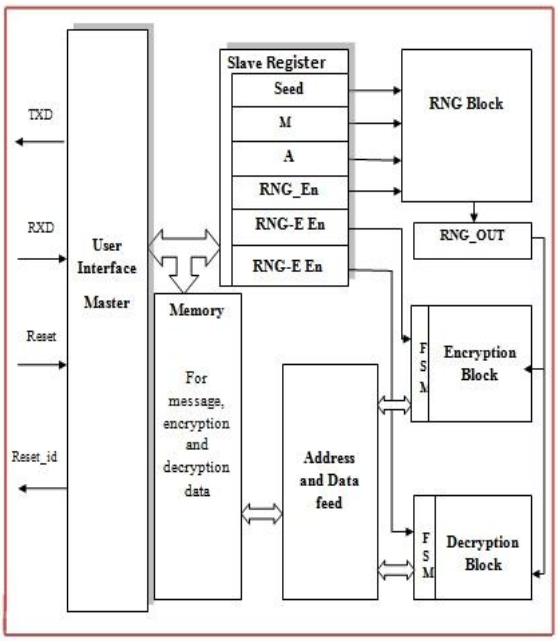

Fig 2: General Block Diagram of Proposed Design

The entire general block diagram is shown in figure 2. It consists of PRNG module, Encryption Block, Decryption Block and slave register. These internal modules described in figure 3 and figure 5.PRNG block generates sequence of 32 bit pseudo random keys with the help of state machine, divider and multiplier. Here we used an APB bus protocol for FPGA Implementation of proposed PRNG.

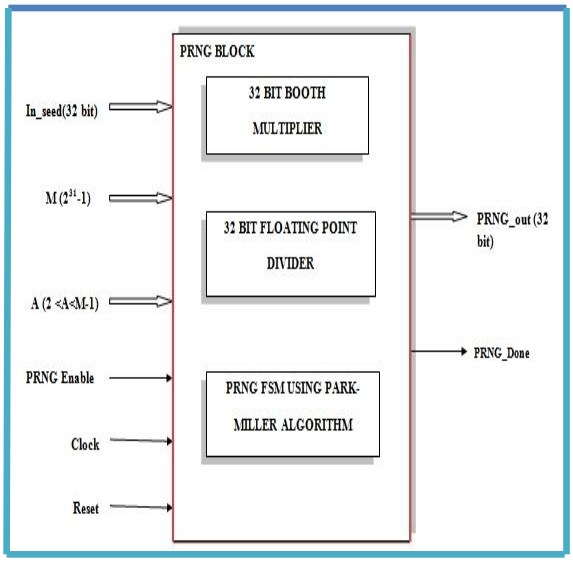

Fig 3: PRNG Block using Park Miller Algorithm

The PRNG module will use this 32-bit input in_seed number to generate its first 32-bit output out_seed when 'Reset' is activated. The second random number generation is different from the first in the input data. The second output out_seed is generated using the first output out_seed as its in_seed number and the subsequent outputs are generated using their immediate previous outputs as in_seed number.

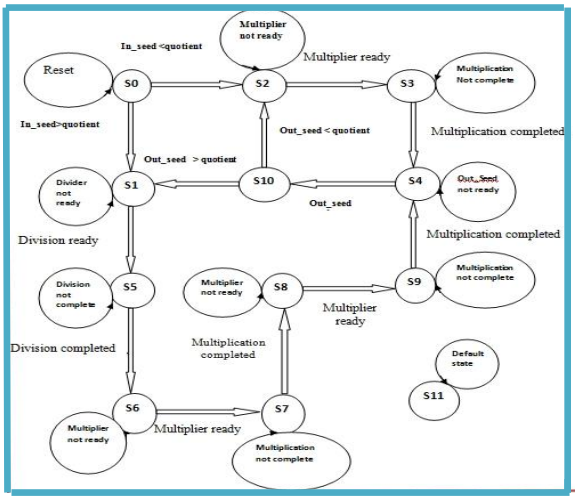

Fig 4: State Diagram of PRNG Sequence

The PRNG module shown in figure 3 is composed mainly of a state machine, 32 bit booth multiplier and 32 bit floating point divider. The state diagram of the PRNG is presented in Figure4.

The default state is 11 and the PRNG will be activated when the 'Reset' signal is high, which changes the state to 0. From state 0, the input in_seed is compared against quotient 127773. If the input in_seed is less than 127773, hi =in_ seed div q will be 0 and lo = in_seed mod q will be the input in_ seed. So from Figure 2, only a multiplication out_seed = a * lo is needed and a shorter period of producing the random number can be achieved. The PRNG then goes to state 1 and the multiplication is taken from state 3 to state 4. On the other hand, if the input in_seed is greater than 127773, one division in_seed/q and two multiplications a*lo and r*hi are needed. So it goes to state 2 and follows the state path. From state 5 to state 9, these computations are taken. As soon as the operation completes, the status register is updated. When they converge at state 10, a random number is generated and this number is taken as the input to restart the flow at either state 1 or 2. As part of the PRNG module, the booth multiplier is implemented using repeated shift and addition algorithm. That is, shift the second multiplicand to correspond with each 1 in the first multiplicand and add to the result.

The floating point divider is designed using the repeated shift and subtraction algorithm which is the reverse of the multiplication by shifting and adding. Basically, division process can be summarized as follows: shift the divisor to correspond with each portion of the dividend, subtract divisor from that portion of the dividend and concatenate 0 or 1 to the result based on the result of the subtraction.

One good feature of the Park-Miller algorithm is that "m**"** is more than 2 billion and is therefore beneficial for series simulations. This algorithm this algorithm has passed a wide range of statistical tests and is known as one of the best [16].

III. ENCRYPTION AND DECRYPTION

The generated sequence of 32-bit sequence keys are plays an important role in the case of stream cipher to encrypt and decrypt plain text. In cryptography, a stream cipher is asymmetric key cipher where plaintext digits are combined with a pseudorandom cipher digit stream (key stream). In a stream cipher each plain text digit is encrypted one at a time with the corresponding digit of the key stream, to give a digit of the cipher text stream. An alternative name is a state cipher, as the encryption of each digit is dependent on the current state.

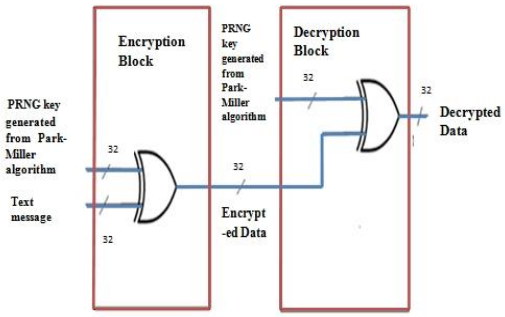

Fig 5:Encryption and Decryption of Plain Text

The design Block diagram of encryption and decryption is shown in Figure 5. Here used a xor gates to encrypt and decrypt the plain data. The pseudo random keys generated from the designed PRNG are sequence of 32 – bits. So after generating the sequence of keys used sequence of 32 bit key for encryption and decryption of plain text.

IV. IMPLEMENTATION AND RESULT

The Programming is done in Verilog-Hardware description language with behavioral level. To validate randomness of the PRNG sequence MATLAB Software used. The proposed module is successfully synthesized and implemented on XILINX Spartan 6E FPGA kit.

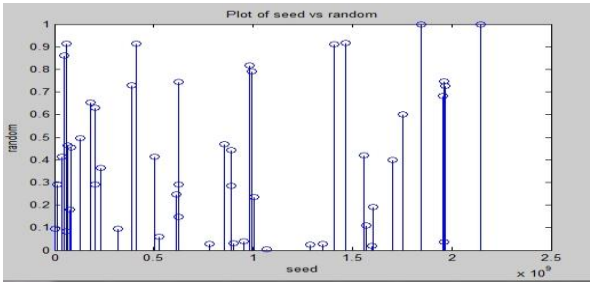

Fig 6: Plot of first 50 PRN sequence with seed input 140000.

The first 50 and 20000 Pseudo random numbers generated as discussed in section II for an arbitrarily chosen inputs in_seed 140000, 'a' is 16807, and 'm' is 2147483647 resulting plots are shown in figure 6 and in figure 7 respectively. The plot shown in Fig 6 random in nature. To verify the functionality of behavioral description of the algorithm simulation is carried out for same inputs 'in_seed', 'm' and 'a' in Modelsim using Verilog-HDL language, the results obtained is perfectly matching with MATLAB results.

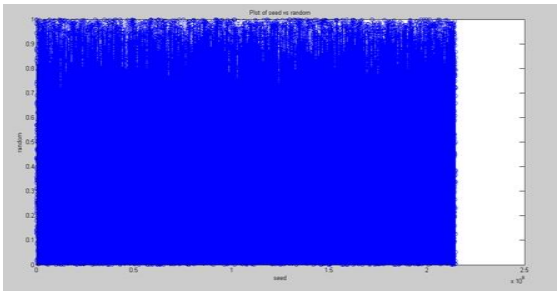

Fig 7: Plot of first 20000 PRN sequence with input seed 140000

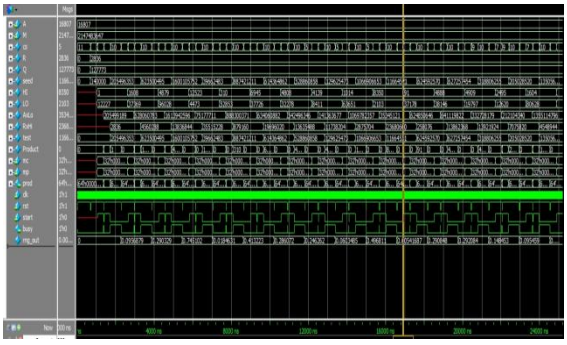

Fig 8:Simulation results of generated pseudorandom numbers with seed input 140000.

The simulation result of generated pseudo random numbers is shown in Figure 8. If the signal 'en' is high then the operation of PRNG module initiates and it takes inputs 'm' (2^{31}-1), 'a'(16807), 'In_seed' 140000. After completion of PRNG operation 'rng_done' signal will high, then the first 32 bit pseudo random number 'rng_out' 0.0956879 produce. For second pseudo random number generation out_seed becomes In_seed and same operation repeats. It gives 0.290329 as second PRN and 0.745102, 0.0184531, 0.413223, 0.286072, 0.246262, 0.603485 and so on.

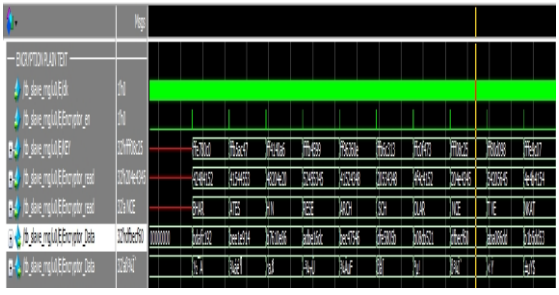

Fig 9: Simulation results of Encryption of Text Message with the help of generated sequence of pseudorandom keys.

Each block of 32 bit data bitwise XOR operation performed with 32 bit PRNG key results encrypted data. The first block of plain text data is 'BHAR' (42484152 in hexadecimal) is bitwise xor with generated 32 bit Pseudo random key 'ffe780c0' the resulted encrypted data is (½¯Á') 'bdafc192' in hexadecimal. Similar procedure is carried out for other blocks of ASCII characters.

The plain text (BHARATESH N RESEARCH SCHOLAR NCET VENKATAGIRI KOTE POST DEVANAHALLI TALUQ BANGALURU 562110)is xor with a Pseudo random keys which results in to encrypted data as (½¯Á¾³Å"·ÇÎà-
¢Ó…¾µÃˆß´Ã°«Ì'ß©Ã…±¬Á"¾ É'¶¬Ï"ºÇÐÐ«Ã „º ±ÁŽ¾¯ÁŒ³¬ "¼«Õ'ß¥ÁŽ¸¦Ì•² õÉÕ±ñÏÇ à).

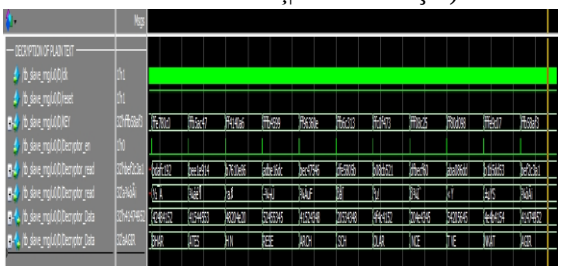

Fig 10: Simulation results of Decryption of Encrypted data with the help of generated sequence of pseudorandom keys.

Each block of 32 bit encrypted data bitwise XOR operation performed with 32 bit PRNG key results decrypted data. The first block of encrypted data (½¯Á') 'bdafc192' in hexadecimal is bitwise xor with generated 32 bit Pseudo random key 'ffe780c0' the resulted decrypted data is 'BHAR' (42484152 in hexadecimal). The same procedure is carried out for others.

The encrypted data which we have obtained from encryption is:
(½¯Á¾³Å"·ÇÎà¢Ó…¾µÃˆß´Ã°«Ì'ß©Ã…±¬Á"¾ É'¶¬Ï"ºÇÐÐ«Ã „º±ÁŽ¾¯ÁŒ³¬ "¼«Õ'ß¥ÁŽ¸¦Ì•-
² õÉÕ±ñÏÇ à). Then the resulted decrypted data is (BHARATESH N RESEARCH SCHOLAR NCET VENKATAGIRI KOTE POST DEVANAHALLI TALUQ BANGALURU 562110).

Table 1: Device utilization report of PRNG

Device Utilization Summary (estimated values)			
Logic Utilization	Used	Available	Utilization
Number of Slice Registers	1161	18224	6%
Number of Slice LUTs	1497	9112	16%
Number of fully used LUT-FF pairs	796	1862	42%
Number of bonded IOBs	148	232	63%
Number of BUFG/BUFGCTRLs	5	16	31%

```
Timing Summary:
---------------
Speed Grade: -3

   Minimum period: 5.232ns (Maximum Frequency: 191.134MHz)
   Minimum input arrival time before clock: 5.640ns
   Maximum output required time after clock: 5.622ns
```

Fig 11:Timing Sumary of proposed Design

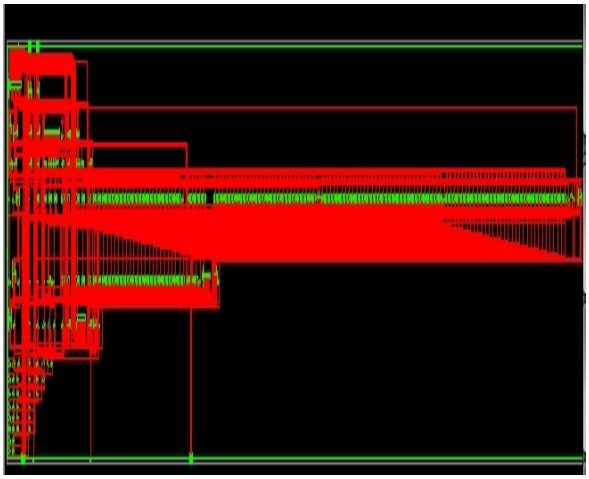

Fig 12:RTL schematic of Proposed design

The design of pseudo random number generator based on Park – Miller algorithm and encryption decryption modules synthesized using Xilinx ISE design Suit 14.3 and the RTL schematic obtained from that is shown in Figure 12. The device utilization and timing summary is given Table 1.The proposed design uses maximum operating frequency of '191.13'MHz, 5.6 ns of setup time and 5.62 ns of hold time as shown in Figure 11.

The proposed pseudo random number generator based on Park-Miller algorithm is synthesized in cadence RTL compiler 180nm technology. From the report the power consumed by the proposed pseudo random number generator is '4.38mW' which is very less compared to other random number generator. It consumes a Static power of '0.0134mW' and Dynamic power of '4.37mW'.So it can be use for any low power cryptographic applications.

The area required for whole design is only '0.447'mm^2 so the proposed pseudo random number generator is area efficient. The comparison of area and power of this design with other random number generators is shown in Table 2.

Table 2: Comparison of Area and Power of proposed design with other random number generators

RNG	[17]	[18]	[19]	[20]	Proposed design
Process (um)	0.18	1.2	0.35	0.25	0.18
Average Power Consumption (mW)	37	120	27	30	4.38
Area (mm^2)	0.767	NA	0.234	0.259	0.447

Fig 13:Spartan 6E FPGA Board

The over all proposed design of generation of pseudo random numbers using Park-Miller algorithm, Encryption and Decryption modules are deployed in spartan 6E FPGA board as shown in Figure 13.

V. CONCLUSION

In this pager the implementation of Park-Miller algorithm has been carried out using the Verilog-HDL programming language and in MATLAB to verify the result. In this work Pseudo random number generator is based on simple Park-miller algorithm design is implemented completely in digital circuit and deployed on general FPGA development board. The design produces a pseudo random number on each clock cycle. Here the text is considered as information and is encrypted using generated pseudo random key. Simulation result shows that text message is completely encrypted and cannot be recognized. The design is synthesized in cadence RTL compiler with 180nm technology. It consumes 4.37mW power and area of 0.447mm^2. Compared to [17][18][19][20] the power and area consumed by proposed PRNG is less. So it has low energy consumption and good statistical quality.

REFERENCES

[1] Z.G. Xiao, *Pseudo-Random Sequence and Its Applications*, Beijing, China: Nat. Defence Ind., 1985.

[2] L.Xu and X. Li, *"Dual-Channel Pseudorandom Sequence Generator with Precise Time Delay Between Its Two Channels," IEEE Trans. Instrum. Meas., vol. 57,* no. 12, Dec. 2008, pp. 2880-2884.

[3] C.H. Yen and B.F. Wu, *"An Error-Correcting Stream Cipher Design with State-Hopping Architecture," J. Chin. Inst. Eng., vol. 28, no.* 1, 2005, pp. 9-16.

[4] X.G.Wangetal.,*"Spread-Spectrum Communication Using Binary Spatiotemporal Chaotic Codes," Phys. Lett. A, vol. 334,no.* 1, Jan. 2005, pp. 30-36.

[5] H.J. Kim et al., *"PN Sequence Generation from 2-D Array of Shift Registers," ETRI J., vol. 27, no.* 3, June 2005, pp. 273-279.

[6] T. Johnsen et al., *"Simultaneous Use of Multiple Pseudo Random Noise Codes in Multistatic CW Radar,"* Proc. IEEE Nat. Radar Conf., 2004, pp. 266-270.

[7] D.K. Rollins et al., *"A Quantitative Measure to Evaluate Competing Designs for Non-linear Dynamic Process Identification," Can. J. Chem. Eng., vol. 84, no*. 4, 2006, pp. 459-468.

[8] H.J.W. Spoelder et al., *"Some Aspects of Pseudo Random Binary Array-Based Surface Characterization," IEEE Trans. Instrum. Meas., vol. 49, no.* 6, Dec. 2000, pp. 1331-1336.

[9] R. Shaltiel and C. Umans, *"Simple Extractors for All Minentropies and a New Pseudorandom Generator,"* Proc. Annu. Symp. Found. Compute. Sci., 2001, pp. 648-657.

[10] A.H. Tan and K.R. Godfrey, *"The Generation of Binary and Near-Binary Pseudorandom Signals: An Overview," Proc. IEEE Instrum. Meas. Technol. Conf., vol. 2,* 2001, pp. 766-771.

[11] J. Szczepanski et al., *"Biometric Random Number Generators," Comput. Secur., vol. 23, no*. 1, Feb. 2004, pp. 77-84.

[12] P. Alfke, *"Efficient Shift Registers, LFSR Counters, and Long Pseudo-Random Sequence Generators,"* Application Note, Xilinx Corp., Aug. 1995.

[13] "*Gold Code Generator Reference Design, Altera Application"* Note 295, Mar. 2003.

[14] F. Principe et al., *"Rapid Acquisition of Gold Codes and Related Sequences Using Iterative Message Passing on Redundant Graphical Models,"* Proc. Int. Conf. Military Commun, 2006.

[15] X.D. Lin and K.H. Chang, *"Optimal PN Sequence Design for Quasi synchronous CDMA Communication Systems," IEEE Trans. Comm., vol. 45, no.* 2, Feb. 1997, pp. 221-226.

[16] S. K. Park and K. W. Miller, *"Random number generators: Good ones are hard to find," Communications of the ACM, vol. 32*, Issue 10.1192-1201.

[17] H. Yang, J. Huang, T. Chang, *"A chaos-based fully digital 120 MHz pseudo random number generator,"* Circuits and Systems, 2004, the 2004 IEEE Asia-Pacific Conference on, pp.357-360, 6-9 December2004

[18] T. Sato, K. Kikuchi, M. Fukase, *"Chip Design of a Wave-Pipelined PRNG,"* Communications and Information Technologies, 2006, ISCIT 2006, International Symposium on, pp. 978-983, 18-20 October2006.

[19] C. Wang, Y. Tseng, H. Cheng, R. Hu, *"Switched-current 3-bit CMOS wideband random signal generator,"* Mixed-Signal Design, 2003,Southwest Symposium on, pp. 186-189, 23-25 February 2003.

[20] F. Pareschi, G. Setti, R. Rovatti, *"A Fast Chaos-based True Random Number Generator for Cryptographic Applications,"* Solid-State Circuits Conference, 2006, ESSCIRC 2006, 32nd European, pp.130-133, 19-21 September 2006