Journal of Engineering Science and Technology
Vol. 6, No. 4 (2011) 411 - 428
© School of Engineering, Taylor's University

IMPLEMENTATION OF NEURAL - CRYPTOGRAPHIC SYSTEM USING FPGA

KARAM M. Z. OTHMAN[1], MOHAMMED H. AL JAMMAS[2,*]

[1]College of Computer Engineering and Science, Gulf University, Bahrain
[2]Computer Engineering and Information Department,
Electronics Engineering College, Mosul University, Iraq
*Corresponding Author: dr_mohammed_al_jammas@yahoo.com

Abstract

Modern cryptography techniques are virtually unbreakable. As the Internet and other forms of electronic communication become more prevalent, electronic security is becoming increasingly important. Cryptography is used to protect e-mail messages, credit card information, and corporate data. The design of the cryptography system is a conventional cryptography that uses one key for encryption and decryption process. The chosen cryptography algorithm is stream cipher algorithm that encrypt one bit at a time. The central problem in the stream-cipher cryptography is the difficulty of generating a long unpredictable sequence of binary signals from short and random key. Pseudo random number generators (PRNG) have been widely used to construct this key sequence. The pseudo random number generator was designed using the Artificial Neural Networks (ANN). The Artificial Neural Networks (ANN) providing the required nonlinearity properties that increases the randomness statistical properties of the pseudo random generator. The learning algorithm of this neural network is backpropagation learning algorithm. The learning process was done by software program in Matlab (software implementation) to get the efficient weights. Then, the learned neural network was implemented using field programmable gate array (FPGA).

Keywords: Cryptography, Random number generator, Artificial neural network, FPGA.

1. Introduction

Cryptography is the science of securing data using mathematics to encrypt and decrypt data. It enables to store sensitive information or to transmits it across insecure networks (like the Internet), so that it cannot be read by anyone except the

Nomenclatures

bb_k	Bias of the output neuron k
b_j	Bias of the hidden neuron j
d	Factor of shifting in bits between the sequence
f	Activation function
f'	Derivative of the activation function
I	Initial value of the sequence
k	Number of blocks
m	Length of the block
N_h	Number of hidden nodes
N_p	Number of input pattern
n_0	Number of occurrence of 0's in the N-bit sequence
n_{00}	Number of occurrence of 00's in the N-bit sequence
n_{01}	Number of occurrence of 01's in the N-bit sequence
n_1	Number of occurrence of 1's in the N-bit sequence
n_{10}	Number of occurrence of 10's in the N-bit sequence
n_{11}	Number of occurrence of 11's in the N-bit sequence
Q	Equation that generate long sequence
t_k	Target output neuron k
v_{jk}	Weight between the hidden neuron j and the output neuron k
w_{ij}	Weight between input neuron i and hidden neuron j
x_i	Input signal of the input neuron i
y	Output of the neural-PRNG
y_{in}	Summation result of output node
y_k	Actual output of the neuron k
z_j	Actual output of the hidden neuron j

Greek Symbols

α	Learning rate
δ_j	Backpropagation error between the hidden and input layers
δ_k	Backpropagation error between the output and hidden layers
χ^2	Distribution value of statistical test
$\oplus$	XOR operator

Abbreviations

ANN	Artificial neural networks
CLB	Configurable logic block
CPS	Connections per second
DES	Data encryption standard
EDA	Electronic design automation
FPGA	Field programmable gate array
MCPS	Mega of connections per second
MSB	Most significant bit
PRNG	Pseudo random number generator
RTL	Register transfer logic

the intended recipient. A cryptographic algorithm, or cipher, is a mathematical function used in the encryption and decryption process. A cryptographic algorithm works in combination with a *key*. A key is used by a cipher as an input that controls the encryption in a desirable manner. The same plaintext encrypts to different cipher text with different keys. The security of encrypted data is entirely dependent on two things: the strength of the cryptographic algorithm and the secrecy of the key [1]. Cryptography found its way into many applications such as e-mail verification, identity authentication, and copyright protection. At a lower level, cryptography is used to provide security for packet cables, embedded system and many more. Owing to its wide use, it is therefore desirable to design and implement efficient cryptographic systems that are robust in various platforms. Many factors are taken into consideration when gauging efficiency, these includes gate count for hardware design, memory requirements in software implementation and cross platform performance.

2. Cryptographic Systems

A cryptographic system is a method of hiding data so that only certain people can view it. Cryptography is the practice of creating and using cryptographic systems. The original data is called plaintext. The protected data is called ciphertext. Encryption is a procedure to convert plaintext into ciphertext. Decryption is a procedure to convert ciphertext into plaintext. The Cryptography can be used to provide confidentiality, data integrity, and authentication [2].

There are two basic types of cryptographic systems: conventional cryptography and public key cryptography. In conventional cryptography, also called secret-key or symmetric-key encryption, one key is used both for encryption and decryption. The Data Encryption Standard (DES) is an example of a conventional cryptosystem that is widely employed by the Federal Government. The second type is a Public key cryptography (introduced by Martin Hellman in 1975) is an asymmetric scheme that uses a pair of keys for encryption a public key, which encrypts data, and a corresponding private, or secret key for decryption. The public key cryptography was used by the British secret service in military secret [3]. A cryptographic system typically consists of algorithms and keys. There are two types of Cryptography Systems: Block Cipher System and Stream Cipher System. In this research, the design of the cryptographic system is the stream cipher system.

Stream ciphers play an especially important role in cryptographic applications that protracts communication in very high frequency domain. The Stream cipher process data one bit at a time. The bit size of the stream cipher is typically one bit or byte. The key size is longer or equal to the message size. They are used in applications where plaintext comes in quantities of unknown length [3]. In stream cipher, the key is fed into an algorithm called the pseudo random number generator to create a long sequence of binary signals. This “key-stream“ is then mixed with the plaintext sequence, usually by XOR operation, to produce the cipher text. Figure 1 shows the stream cipher cryptography process.

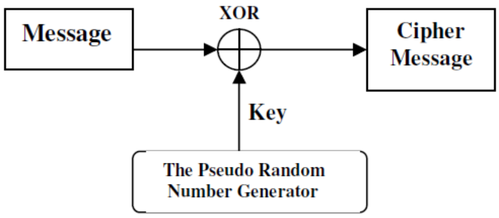


Fig. 1. Stream Cipher Cryptography Process.

3. Pseudo Random Number Generator (PRNG)

A pseudo random number generator (PRNG) is an algorithm for generating a sequence of numbers that approximates the properties of random numbers. The security of most cryptographic algorithms and protocols using PRNGs is based on the assumption that it is infeasible to distinguish use of a suitable PRNG from a random sequence. The simplest examples of this dependency are stream ciphers, which (most often) work by XOR the plaintext of a message with the output of a PRNG, producing ciphertext. Pseudo-random number generators play an important role in the use of encryption various network security applications. A sequence generator is pseudo-random if its have the following properties [4]:

- It looks random. This means that it passes all the statistical test of randomness.
- It is unpredictable. It must be computationally infeasible to predict what the next random bit will be, given complete knowledge of the algorithm or hardware generating the sequence and the entire previous bit in the stream. The output of a generator satisfying these properties will be good pseudo-random key generation, and any other cryptographic applications that required a truly random number generator.

4. The Statistical Tests

The statistical randomness tests which are usually used in checking the randomness properties of the PRNG. Some complexity measure can be applied to binary sequence to test if this sequence appears to be random sequences or not and that explained in appendix A. The basic statistical tests are: *Frequency test, serial test, poker test and autocorrelation test* [5, 6].

5.Artificial Neural Network

A neural network is an interconnected net of individual neurons. Each neuron is equipped with a specific function which is then used to evaluate the data coming in to it. The data coming into the neuron can either be a specific input, or an output from another neuron. Most types of neural network have multi-layer: an *input layer* through which data is given to the network, an *output layer* that holds the response relative to the input, and optional layer between the input and output layer called *hidden layer* where learning takes place. The number of the neuron in the input and output layers can be determined by the number of input and output variables in the physical system. The number of hidden layers and the number of neurons in this layer are arbitrary and can vary according to the application [7].

Neural network has the possibility of learning. The process of determining the weights by which the best match between the desired output and the artificial neural network output is called training process. Training will be most effective if the training data is spread throughout the input space. Learning algorithms are classified into supervised learning process and unsupervised learning process. Supervised learning is the learning with a "teacher" in the form of a function that provides continuous feedback on the quality of solutions. These tasks include pattern classification, function approximation and speech recognition, etc. Unsupervised learning refers to the learning with old knowledge as the prediction reference. These tasks include estimation problem, clustering, compression or filtering [8]. In this work, the architecture of the neural network is the backpropagation neural network and the learning algorithm is back propagation learning algorithm (see *Appendix B*) [8] (i.e., supervised training process).

The aspiration to build intelligent systems complemented with the advances in high speed computing has proved *through* simulation the capability of artificial neural networks (ANN) to map, model and classify nonlinear systems. The learning capability of the network has opened its application to various fields of engineering, science, economics, etc. Neural networks can be implemented using analog or digital systems. The digital implementation is more popular as it has the advantage of higher accuracy, better repeatability, lower noise sensitivity, better testability, higher flexibility, and compatibility with other types of preprocessors. The digital NN hardware implementations are based on FPGA (field programmable gate array) [9].

FPGA is a suitable hardware for neural network implementation as it preserves the parallel architecture of the neurons in a layer and offers flexibility in reconfiguration. FPGA-based reconfigurable computing architectures are well suited to implement ANNs as one can exploit concurrency and rapidly reconfigure to adapt the weights and topologies of an ANN [10].

6. The Artificial Neural Network Based Stream Cipher

The stream cipher cryptography need to a long unpredictable sequence of binary signals called key that used for encryption process. The Pseudo random number generators have been widely used to construct this key sequence that shown in Fig. 1. Today a common secret key based on number theory could be created over a public channel accessible to any opponent but it might not be possible to calculate the discrete logarithm of sufficiently large numbers. DES, 3DES, and Rivest, Shamir and Adleman (RSA) are the well-known and the mostly used (preferred) encryption and decryption systems. In general the cost of these systems is high and it requires more time for computation and applications. Artificial neural networks (ANNs) have been applied to solve many problems: Learning, generalization, less data requirement, fast computation, ease of implementation, and software and hardware availability features have made ANNs very attractive for the applications [11].

7. Software Implementation of Neuro-crypto System

As mentioned, the stream cipher is a symmetric cipher in which the plaintext digits are ciphered one at a time, and the transformation of successive digits varies during the ciphering algorithm. If these variations are based on a nonlinear function, the encryption algorithm becomes strongly unbreakable. Building an efficient pseudo random number generator (PRNG) that possesses randomness properties will lead to have a stream cipher algorithm with random keys.

The designing PRNG use the nonlinear characteristics of neural network. MATLAB computer simulation test has been carried to learn the neural network PRNG using backpropagation learning algorithm and check the randomness of the Neural PRNG sequence generator through the statistical tests. Moreover, this simulation is also important in finding the minimum ANN architecture since minimum FPGA hardware is an important task of this work.

8. The Proposed Neural PRNG

The proposed neural pseudo random number generator consists of two stages.

The first stage is generating a long sequence of patterns from perfect equation and initial value. So these patterns possess the randomness and unpredictable properties. The total number of equations and initial values depend on the number of bits that represent the initial value. For example, if the initial value is represented in 10 bits (N=10), the total number of possible equations is $1023=2^{10}-1$ and the total number of patterns is 1023 patterns. The following algorithm describes the generation of patterns from any equation:

1- Select the initial value (*I*) of 1st pattern from 1 to 1023.
2- Select an equation (*Q*) from 1 to 1023 and convert the number of equations to binary code.
3- Determine the locations of 1's from the binary code of the equation.
4- Make an XOR operation of the initial value between the locations selected from step 3.
5- The result of XOR operation is the most significant bit (MSB) of the pattern.
6- Find the new pattern from the following
New pattern= shift right the old pattern by the MSB
7- For the next step, the new pattern = the initial value.
Repeat from step 4 to step 7.

The algorithm above was applied to all possible values and equations to determine the efficient equations that used to generate long random patterns. Figure 2 shows the generating random patterns for the neural-PRNG.

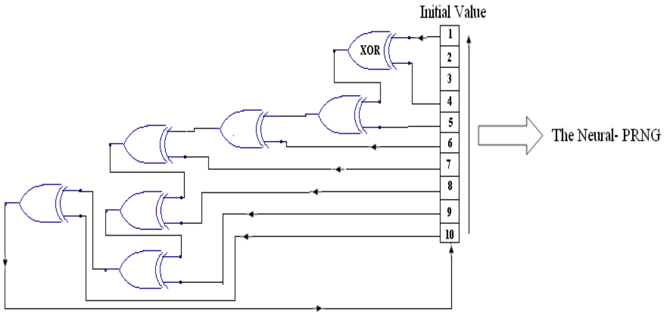


Fig. 2. Generating Input Patterns for the Neural-PRNG.

The second stage is an artificial neural network (ANN) that gets the outputs of the previous stage and set it as input to the NN. The generation process of the key using backpropagation neural network [8] consists of three phases. The first phase is the feedforward, the second phase is a backpropagation of the associated error and the third phase is related with weights adjustments. The proposed ANN for PN sequence generator is a (N_p: N_h: 1) backpropagation network with initial weight randomly set where N_p is the number of input patterns and N_h is the number of the hidden nodes. Each input neuron delivered a binary bit randomly generated. The target is "1" if the net input MSB (the bit of node number N_p) is "1", otherwise the target is "0". NN output is compared with the target and the error is backpropagated to update the weights. The output is entered a binary step function to produce the first bit of key stream.

To enter 2nd pattern, the bits (1,4,5,6,7,8,9,10) of the previous pattern are XOR-ing to generate the MSB of the 2nd NN input with shifting other bits upwards so that: $(MSB)_{old\ pattern} = (MSB\text{-}1)_{new\ pattern}$

This process is repeated until the required key length which is equal to the number of inputs (N_p) is obtained. Figure 3 shows the Neural PRNG structure.

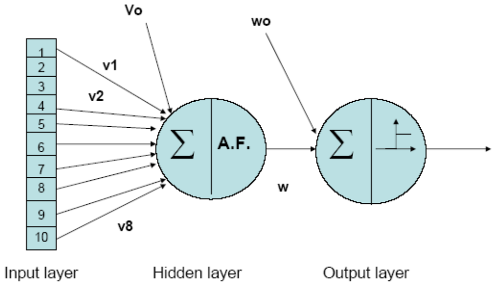


Fig. 3. Neural PRNG Structure.

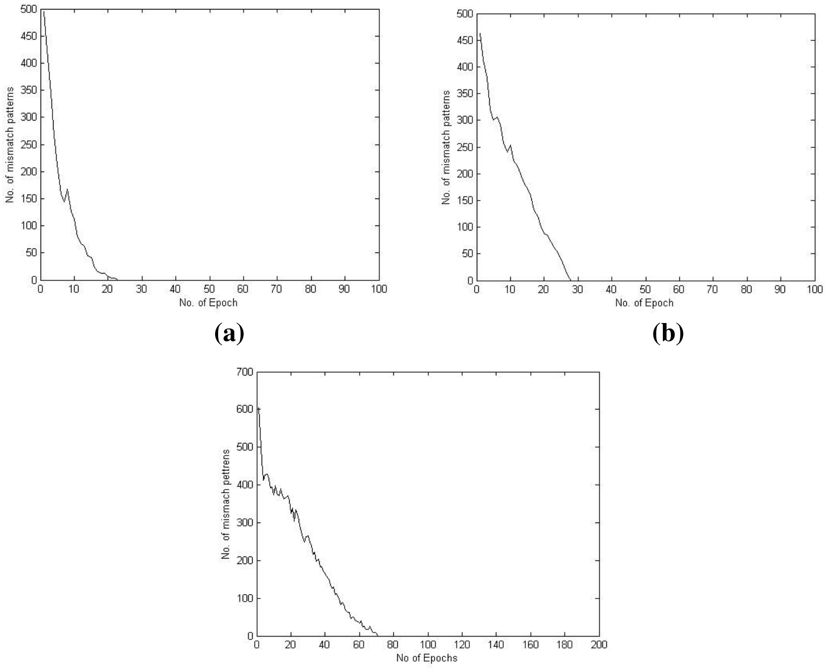
The randomness of the generated key from the backpropagation neural network can be justified using statistical tests. If the justification passes, the key can be used for data encryption.

9.Matlab Results

The matlab results contain two states. The first state is the training process that shows the efficient weights and learning the neural network using back-propagation learning algorithm. The second state is the key generation process that used in the encryption process, then check this key if it passes the statistical tests or not.

9.1. Training process

Neural- PRNG would be designed for a key length of 1024 bits. Several attempts had been performed to reach to a minimum architecture with input nodes equal 8. It was found that N_h=1 is quite sufficient to have the required Neural-PRNG. Since it is required to implement such designed Neural-PRNG generator in an FPGA environment, it is important to have minimum hardware (minimizing the architecture). Therefore, a backpropagation learning algorithm is simulated in MATLAB using a binary data representation until minimum mismatch cases between the output and the target patterns are reached (error =0). Then, the obtained weights are saved and they are very important to design a hardware leaned neural network. Figure 4(a) represents a Neural-PRNG of [8:8:1] NN that is learned to obtain efficient weights. It runs for 23 epochs to get complete matching between the output and the target patterns (zero mismatch cases). Figures 4(b) and 4(c) show the training process of [8:2:1] and [8:1:1] neural network that are run for (28 and 72) respectively to get complete matching between the output and the target patterns (error=0).



(a)

(b)

(c)

Fig. 4. The learning Process of
(a) ANN [8:8:1], (b) ANN [8:2:1], (c) ANN [8:1:1].

In the training process, the elapsed time to get the efficient weights in the training process of [8:1:1] neural network is longer than the elapsed time of the training process in [8:8:1] and [8:2:1] neural networks. Choosing the [8:1:1] neural network architecture in the design of the pseudo random number generator is came from the following reasons:

- The elapse time of training process is not important because this work deals with the learning-off chip process (Designing leaned neural network).
- In the hardware implementation, the [8:1:1] NN decreased the hardware recourses like multipliers (for each neuron) and memory (to store the efficient weights) that is limited in FPGA card.

9.2. Key generation process

When the neural network was learned and the weights were saved, the next stage is the key generation process. In this process, the weights are very important to run the neural network and generate random binary keys. The representation of the weights has two ways: the floating- point representation and the integer representation.

9.2.1. Floating point representation

The weights that resultant from the learned neural network are represented using the floating point representation. This type of representation is a favourite in software because it has precision that is always maintained with a wide dynamic range. Then, the trained neural network was applied with saving the efficient weights in order to generate randomness binary keys that used in the encryption process. The generated random keys must be checked to ensure that the binary keys have high randomness properties. The disadvantage of the floating point representation is the floating point takes up almost three times the hardware resources of integer representation. Consequently, the efficient weights are represented using the integer representation to get minimum hardware resources and efficient implementation of the design [10, 12].

9.2.2. Integer representation

The integer representation has less precision than the floating point representation. The learned neural network was applied with fixing the efficient weights in order to generate randomness binary keys. And the statistical tests are applied to check the randomness of these keys.

10. Field Programmable Gate Array (FPGA)

A Field Programmable Gate Array (FPGA) is completely reconfigurable logic chip. This logic chip consists of millions of configurable logic block (CLB) which can be linked together to form complex digital logic implementation. The individual units are interconnected by a matrix of wires and programmable switches. A user’s design is implemented specifying the simple logic function for each logic block and selectively closing the switches in the interconnected matrix [13]. A typical computer aided design (CAD) for an FPGA would include software for certain tasks like the behavioral design, functional simulation, verification, placing and routing. Using Xilinx ISE for simulation environment and VHDL (VHSIC Hardware Description Language) also provides a consistent and portable design instrument pointing FPGAs [14].

- The advantages of FPGA over a microprocessor chip for neural computing can be listed as in [10]:
- Developing hardware systems using design tools for FPGA is as easy as developing a software system.
- FPGA can be reprogrammed on the field.
- The new FPGA support hardware that required more than one million gates.
- A custom circuit built on an FPGA operates faster than microprocessor chip.

These advantages make FPGA now viable alternatives to other technology implementations for high speed neural applications.

11. FPGA Design Flow

The design flow broadly refers to the sequence of activities encompassing various design tools that begin with some abstract specification of a design and ends with a configured FPGA [14]. The design flow was described in Fig. 5.

The core generation utilities are divided in three categories: HDL editor, machine editor and schematic capture. Number of EDA (Electronic Design Automation) tools provides these functionalities, such as Xilinx ISE and ModelSim. In VHDL environment, behavioural design and synthesis are necessary and help to check the correctness and logicality of the program. After functional simulation, the verification and place and routing are start on the FPGA. Once the verification is completed, the program can be downloaded onto the FPGA [14].

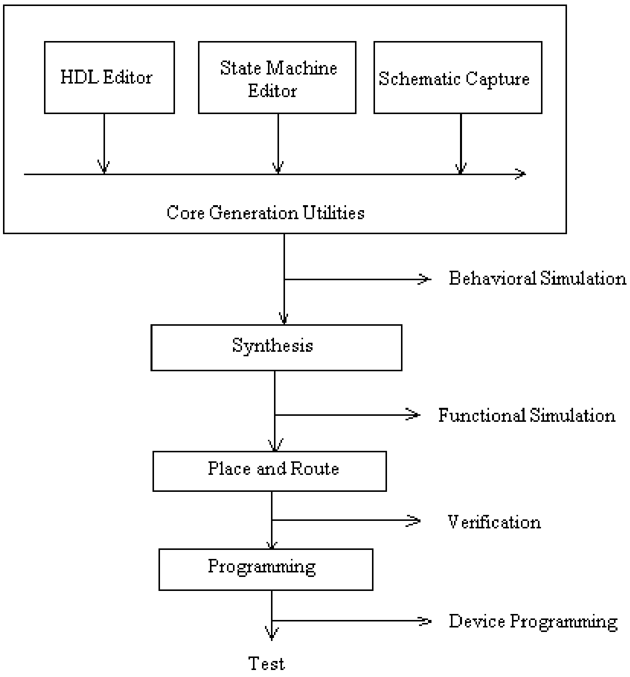


Fig. 5. FPGA Design Flow.

12. Hardware Implementation of Neuro-crypto System

Implementing neuro-crypto system and especially the neural PRNG onto an FPGA is a relatively hard process. However, due to the limited space available on an FPGA, some restrictions have to be made:

- Once the training is completed and the correct network weights determined these weights will have to be coded onto the FPGA. The accuracy in which these weights can be coded will depend upon the number of bits available to implement the weights. The weights will then have to be scaled to values that can be coded within this restriction. The method used to convert the weights from floating-point representation to binary code representation is multiplying the floating-point number by the wanted resolution ($8192=2^{13}$) that gets minimum error. For example: let weight = 0.2385 and the resolution is 8192 (2^0-2^{13}) to make the representation of the weight is 14 bit.

  0.2385 * 8192= 1953.792 = 1954
  1954 = (00011110100010) 14 bit

- To calculate the neuron outputs on an FPGA, it will be necessary to implement some combination of adders and multipliers. These will be used according to Eq. (1) to determine the neuron outputs.

$$y_{in} = \sum_{i=1}^{n} x_i w_i + b \tag{1}$$

In FPGA, the number of hardware multipliers is 18 multiplier resources. So the multiplication method must to be simplified in to a form like adders [15]. Special attention must be paid to an efficient approximation of the sigmoid function in implementing FPGA-based reprogrammable hardware-based artificial neural networks. The equation that expresses the sigmoid activation function is:

$$y = \frac{1}{1+e^{-y_{in}}} \tag{2}$$

- The sigmoid activation function has several ways of implementation in VHDL. The method that re-expresses the sigmoid Eq. (2) into simple equations is piecewise second-order approximation [16, 17] that shown in Eq. (3).

$$\begin{aligned} y &= 1 && \text{when } y_{in} > 4 \\ y &= 1 - \frac{1}{2}\left(1 - \frac{1}{4} y_{in}\right)^2 && \text{when } 0 < y_{in} < 4 \\ y &= 1 - \frac{1}{2}\left(1 - \frac{1}{4}\left|y_{in}\right|\right)^2 && \text{when } -4 < y_{in} < 0 \\ y &= 0 && \text{when } y_{in} < -4 \end{aligned} \tag{3}$$

13. The Neuro-Crypto System Design based FPGA

The design of neuro-crypto system consists of two parts. The first part is designing an input generator logic circuit to prepare input patterns to ANN stream cipher. Figure 6 illustrates the circuit diagram of the input generator logic circuit in RTL (Register Transfer logic) level. The logic circuit was designed and implemented in Spartan3e FPGA (XC3S500efg320-4).

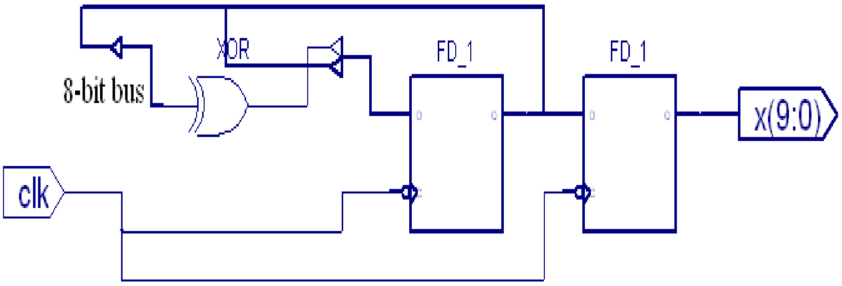


Fig. 6. Circuit Diagram of the Input Generator.

Figure 7 shows the timing simulation design results of the input generator logic circuit using Xilinx ModelSim XE II 5.8c program.

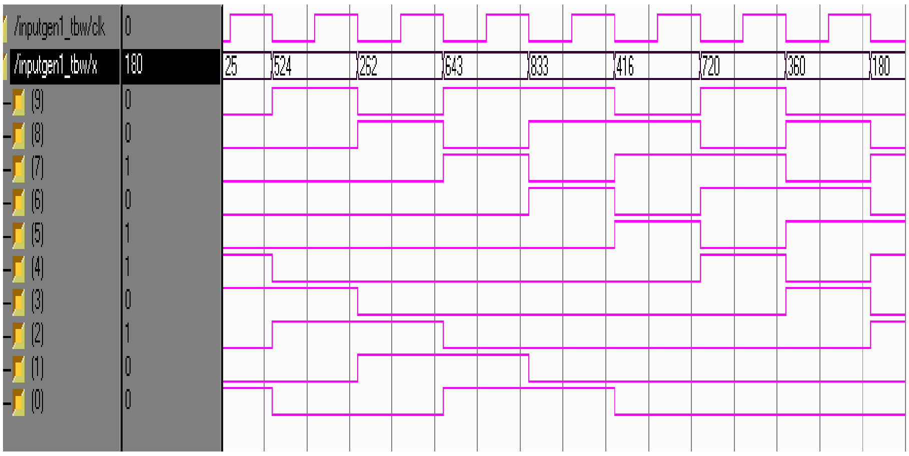


Fig. 7. The Simulation Design Results of the Input Generator Logic Circuit.

The second part of the design is the Pseudo Random Number Generator (PRNG) based Artificial Neural Network (ANN). This PRNG is simulated and synthesized in ModelSim XE II 5.8c and implemented in Spartan3E FPGA. Figure 8 shows the diagram circuit of the PRNG based ANN in RTL Schematic viewer.

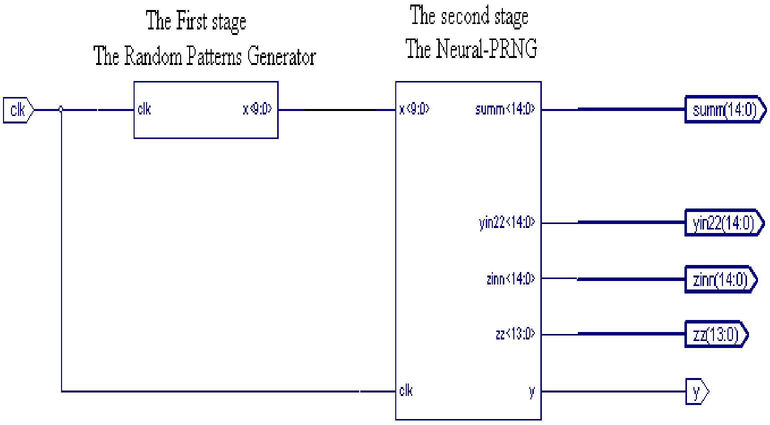


Fig. 8. The Circuit Diagram of PRNG Based ANN.

The timing diagram of the implemented pseudo random generator is shown in Fig. 9. Figure 10 shows the internal structure of the Neural – PRNG .

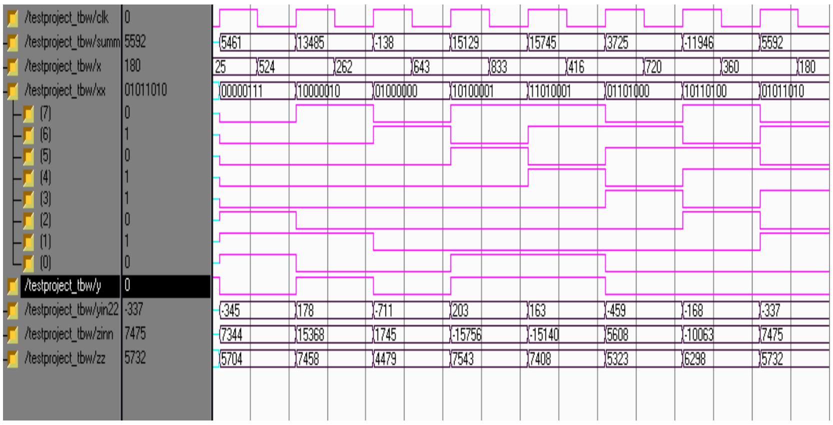

/testproject_tbw/clk 0
/testproject_tbw/summ 5592 5461 13485 -138 15129 15745 3725 -11946 5592
/testproject_tbw/x 180 25 524 262 643 833 416 720 360 180
/testproject_tbw/xx 01011010 00000111 10000010 01000000 10100001 11010001 01101000 10110100 01011010
(7) 0
(6) 1
(5) 0
(4) 1
(3) 1
(2) 0
(1) 1
(0) 0
/testproject_tbw/y 0
/testproject_tbw/yin22 -337 -345 178 -711 203 163 -459 -168 -337
/testproject_tbw/zinn 7475 7344 15368 1745 -15756 -15140 5608 -10063 7475
/testproject_tbw/zz 5732 5704 7458 4479 7543 7408 5323 6298 5732

Fig. 9. The Timing Diagram of the System.

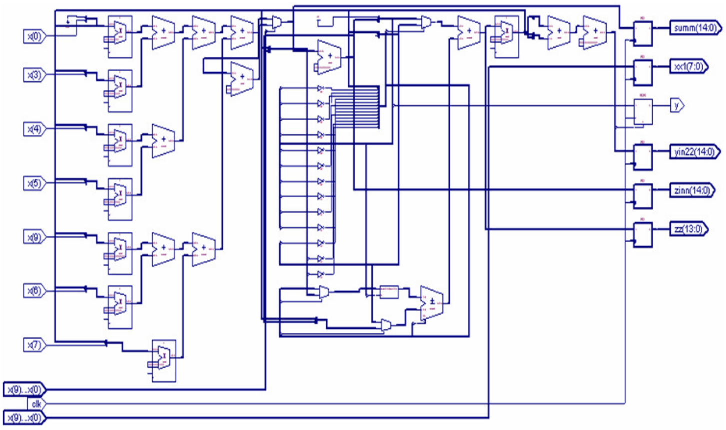


Fig. 10. The Internal Structure of the Neural – PRNG.

14. Hardware Performance Evaluation

The traditional approach for quantifying the hardware design performance is to measure the number of accumulate operations performed in the unit time that measured in MCPS (Mega of Connections Per Second) [17]. The calculation of CPS is shown below:

$$CPS = frequency \times number\ of\ bits\ of\ the\ weight \times number\ of\ inputs$$

Then, the CPS = 50MHz × 15 × 8 = 6000 MCPS.

The execution time is very important in the calculation of the design performance. In this work, the processing time of software simulation is equal to 94 msec that is simulated in personal computer has the following properties: 1.7GHz CPU Celeron and 512MB RAM. The processing time of hardware implementation is equal to 0.0736 msec that is simulated in spartan3e FPGA card and has the following properties:

Frequency =50 MHz, Total CLB=1164 and (18×18) multipliers = 20.

In order to ensure the parallel processing performance in comparison to the sequential processing one, Fig. 11 shows the processing time (in ms) of parallel (FPGA) and sequential (PC) processors.

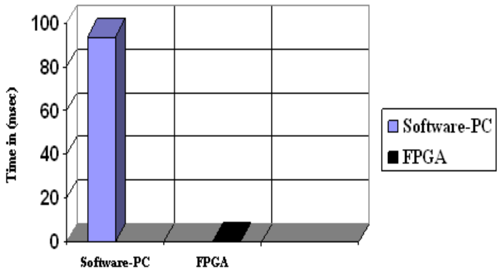


Fig. 11. The Processing Time of Parallel (FPGA) and Sequential (PC) Processors.

In hardware evaluation, the term *speedup* refers to how much a parallel procedure is faster than a corresponding sequential procedure. The speedup can be calculated by the following formula, Eq. (4) [17]:

$$\text{speedup} = \frac{\text{software time}}{\text{hardtware time}} \tag{4}$$

Then, the speedup = 94 ms/0.07336 ms=1281. This value means that the hardware performance is faster than the software performance in 1281 times. So this is taking an advantage for hardware parallelism.

15. Conclusions

In this work, a Pseudo Random Number Generator (PRNG) based on artificial Neural Networks (ANN) has been designed. This PRNG has been used to design stream cipher system with high statistical randomness properties of its key sequence using ANN. Software simulation has been build using MATLAB to firstly, ensure passing four well-known statistical tests that guaranteed randomness characteristics. Secondly, such stream cipher system is required to be implemented using FPGA technology, therefore, minimum hardware requirements has to be considered. In order to ensure the FPGA-based PRNG performance in comparison to the MATLAB-based one, Table 1 indicates the statistical properties of both software and hardware PRNG versions.

Table 1. The Statistical Properties of both Software and Hardware PRNG Versions.

Test Name	Obtained Value		Average value	Comments
	Matlab Results	FPGA Result		
Frequency Test	0.009	2.347	<=3.841	Pass
Serial Test	0.0108	3.422	<= 5.99	Pass
Poker Test	4.3607	5.299	<= 14.6	Pass
Autocorrelation Test	0.0628	0.062	1.96	Pass

Taking advantage of hardware parallelism, the hardware performance is faster than the software performance in 1281 times. So FPGAs exceed the computing power of personal computers by breaking the paradigm of sequential execution.

References

1. Alder, M.; and Gailly, J.L. (2004). *An introduction to cryptography.* PGP Corporation, USA.
2. Konheim, A.G. (2007). *Computer security and cryptography.* Wiley Inc.
3. Daswani, N.; Kern, C.; and Kesavan, A. (2007). *Foundation of security: What every programmer needs to know.* (1st Ed.) Apress Inc.
4. Shackleford, B.; Tanaka, M.; Carter, R.J.; and Snider, G. (2002). FPGA implementation of neighbourhood-of-four cellular automata random number generators. *Technical Report*, HP Laboratories Palo Alto, HPL-2001-290.
5. Menezes, A.J.; van Oorschot, P.C.; and Vanstone, S.A. (1996). *Handbook of applied cryptography*. CRC Press Inc.
6. Rukhin, A.; Soto, Juan; Nechvatal, J.; Smid, M.; and Barker, E. (2001). A statistical test suite for random and pseudorandom number generators for cryptographic applications. *NIST* special publication.
7. Negnevitsky, M. (2005). *Artificial intelligence: a guide to intelligent system.* (2005). (2nd Ed.), Pearson Education.
8. Laurene Fausett (1993). *Fundamentals of* neural *network architecture, algorithm and applications.* Available via internet at: http://www.scribd.com/doc/4471440/Fundamentals of Neural-Networks Laurene Fausett .
9. Omondi, A.R.; and Rajapakse, J.C. (2006). *FPGA implementations of neural networks.* (1st Ed.) Springer.
10. Khalifa, Y.M.A.; and Fan Y.J. (2005). An FPGA-based general purpose neural network chip with on-chip learning. *Proceedings of Fourteenth Genetic and Evolutionary Computation Conference..*
11. Seref Sagisroglu; and Necla Ozkaya (2007). Neural solutions for information security. *Journal of Polytechnic*, 10(1), 21-25.
12. Brown, S.; and Vranesic, Z. (2005). *Fundamentals of digital logic design using VHDL design*. (2nd Ed.), McGraw-Hill Inc.
13. Wolf, W. (2004). *FPGA-based system design.* PTR Prentice Hall.
14. Pedroni, V.A. (2004). *Circuit design with VHDL.* MIT press, London.
15. A. Muthuramalingam; S. Himavathi; and E. Srinivasan (2007). Neural network implementation using FPGA: issues and application. *International Journal of Information Technology*, 4(2), 86-92.
16. Tommiska, M.T. (2003). Efficient digital implementation of the sigmoid Function for reprogrammable logic. *IEE Proceedings on Computers and Digital Techniques*, 150(6), 403-411.
17. Rajah, A. (2005). *VLSI design of a neurohardware processor implementing the Kohonen neural network algorithm.* M.Sc. Thesis, Faculty of Electrical, Engineering, University of Technology Malaysia.

Appendix A

Statistical Tests

The statistical randomness tests are usually used in checking the randomness properties of the PRNG. Some complexity measure can be applied to binary sequence to test if this sequence appears to be random sequences or not. The Table A-1 shows the values of the χ^2-distribution of the statistical tests [5]:

Table A-1 χ^2- Distribution Values of the Statistical Tests.

Test Name	The χ^2 value	Degree of Freedom
Frequency Test	3.841	1
Serial Test	5.99	2
Poker Test	14.6 when m =3	7
Autocorrelation Test	1.96	10

A-1 Frequency Test

In a randomly generated N-bit sequence, we would expect approximately half the bits in the sequence to be ones and approximately half to be zeroes. The frequency test involves the calculation of χ^2 using [5]:

$$\chi^2 = (n_0 - n_1)^2 / N \tag{A.1}$$

where n_0 is the number of occurrences of 0's in the N-bit sequence.

n_1 is the number of occurrences of 1's in the N-bit sequence.

Table A-1 shows the values of the χ^2-distribution of the statistical tests. From this table it is found that the value of χ^2 for a 5% significant level is 3.841. The random property of the key sequence generation is accepted if the value of the χ^2 is less than 3.841, otherwise it fails. If n_o=n_1 then χ^2=0. To decide if the value obtained from the sequence is good enough for the sequence to pass the test, we need to compare the value of χ^2 with values of a table of χ^2-distribution for one degree of freedom [5, 6].

A-2 Serial Test

The serial test checks that the frequencies of the different transitions in a binary sequence (00, 01, 10 and 11) are approximately equal. Let n_{00} , n_{01}, n_{10} and n_{11} be the number of occurrence of 00,01,10 and 11 respectively in the N-bit sequence. Ideally n_{00} =n_{01}=n_{10} = n_{11}= (N-1)/4. The serial test involves the calculation of χ^2 using [5]:

$$\chi^2 = \frac{4}{N-1}(n_{00}^2 + n_{01}^2 + n_{10}^2 + n_{11}^2) - \frac{2}{N} - (n_0^2 + n_1^2) + 1 \tag{A.2}$$

From Table A-1 it is found that the value of χ^2 for a 5% significant level is 5.99. The random property of the key sequence generation is accepted if the value of the χ^2 is less than 5.99, otherwise it fails [5]. To decide if the value obtained from the sequence is good enough for the sequence to pass the test, we need to compare the value of χ^2 with values of a table of χ^2-distribution for two degrees of freedom.

A-3 Poker Test

The poker test is a generalization of the frequency test and the serial test. The frequency test studies the number of occurrences of both of the 1-bis patterns, and the serial test studies the number of each of the four 2-bit patterns. The poker test studies the number of occurrences of each 2^m, *m*-bit patterns, for some integer *m*. The poker test involves the calculation of χ^2 using [5]:

$$\chi^2 = \frac{2^m}{k}\left(\sum_{i=1}^{2^m} n_i^2\right)^2 - k \tag{A.3}$$

where *k* is the number of blocks

m is the length of block.

In this test, the calculated value should be compared with Table A-1 for χ^2 with 2^m-1 degrees of freedom. This test can be applied many times for different values of *m*. The random property of the key sequence generation is accepted if the value of the χ^2 is less than 14.6 with *m*=3, otherwise it fails [5].

A-4 Autocorrelation Test

The autocorrelation test equation is [5, 6]:

$$A(d) = \sum_{i=1}^{N-d} U_i \oplus U_{i+d} \tag{A.4}$$

where *N*-bit sequence is $U = U_1 \quad U_2 \ldots\ldots U_n$. The factor *d* is shift in bits between the sequence U_i with U_{i+d}, where ⊕ denotes the XOR operator. The calculation of χ^2 in this test is:

$$\chi^2 = 2\left\{A(d) - \frac{N-d}{2}\right\} / \sqrt{N-d} \tag{A.5}$$

The random property of the key sequence generation is accepted if the value of the χ^2 is less than 1.96, otherwise it fails.

Appendix B

The Backpropagation Algorithm

The algorithm below shows the procedure to achieve the process of PN sequence generator based on NN [8]:

1- Network weight values initialization (set to small random values).

2- Sum weights input and apply activation function to compute the output of the hidden layer using:

$$z_j = f(\sum_{i=1}^{n} x_i w_{ij} + b_j) \tag{B.1}$$

where: z_j is the actual output of the hidden neuron *j*.
x_j is the input signal of the input neuron *i*.
w_{ij} is the weight between input neuron *i* and hidden neuron *j*.
b_j is the bias of the hidden neuron *j*.
f is the activation function.

3- Sum the weight output of the hidden layer and apply the activation function to compute the result of the output layer neuron using:

$$y_k = f(\sum_{j=1}^{n} h_j v_{jk} + bb_k) \tag{B.2}$$

where y_k is the actual output of the neuron *k*.
v_{jk} is the weight between the hidden neuron *j* and the output neuron *k*.
bb_k is the bias of the output neuron *k*.

4- Compute the back propagation error using:

$$\delta_k = (t_k - y_k) f'(\sum_{k=1}^{n} z_j v_{jk} + bb_k) \tag{B.3}$$

where: f' is the derivative of the activation function.
t_k is the target output neuron *k*.

5- calculate weight and bias correction output layer using :

$$\Delta v_{jk} = \alpha \delta_k z_j \tag{B.4}$$

$$\Delta bb_k = \alpha\ \delta_k \tag{B.5}$$

where α is the learning rate.

6- Add the delta input for each hidden unit and calculate the error term using:

$$\delta_j = \sum_{j=1}^{n} \delta_k v_{jk} f'\left(\sum_{j=1}^{n} x_i w_{ij} + b_j\right) \tag{B.6}$$

7- Calculate the weight and the bias correction for the hidden layer using:

$$\Delta w_{ij} = \alpha \delta_j x_i \tag{B.7}$$

$$\Delta b_j = \alpha \delta_j \tag{B.8}$$

8- Update weights and biases using the following equations:

$$(v_{jk})_{new} = (v_{jk})_{old} + \Delta v_{jk} \tag{B.9}$$

$$(bb_k)_{new} = (bb_k)_{old} + \Delta bb_k \tag{B.10}$$

$$(w_{ij})_{new} = (w_{ij})_{old} + \Delta w_{ij} \tag{B.11}$$

$$(b_j)_{new} = (b_j)_{old} + \Delta b_j \tag{B.12}$$

Repeat steps 2 to 8 until output converge from the target (i.e., the error rate < 0.0005).