International Journal of Computer Sciences and Engineering Systems
Vol. 11 No. 1 (June, 2017)

A Comparison of Architecture Design and FPGA Implementation of Programmable Cellular Automata Generator and Stream Generators

Guitouni ZIED, Machhout MOHSEN, Zeghid MEDIEN and Tourki RACHED

Electronics and Micro-Electronic Laboratory (LEME), Monastir, TunisiaMonastir, 5000, Tunisia
E-mail: ziedguitouni@yahoo.fr

Abstract: *Security protocols and encryption algorithms are essentially based on algorithms and on Random Number Generators (RNG). In this paper, we described a single key cryptographic system based on Programmable Cellular Automata Generator (PCAG) and Stream Generators. We proposed reconfigurable architecture of PCAG. The hardware implementations of PCAG and two representative stream generators are compared in terms of performance and consumed area. The ciphers used for the comparison are the A5/1 and W7. The designs were coded using VHDL language. For the hardware implementation of the designs, on a reconfigurable hardware platform Virtex II 2V250FG256 FPGA device was used. The implementation results illustrate the hardware performance of each generator in terms of throughput-to-area ratio. This ratio equals to: 9.78 for the A5/1, 2.72 for the PCA and 2.35 for the W7.*

Keywords: Cellular automata, Stream ciphers, A5/1 generator, W7 generator and VLSI implementation

1. INTRODUCTION

Random number generators play an important role in several computational fields, including Monte Carlo techniques, cryptographic protocols, and stochastic optimization methods [2]. With the advent of massively parallel scientific computation, the generation of pseudorandom numbers has become essential.

Random number generators must possess a number of properties if they are for cryptographic application. The most important properties from this point of view are good results on standard statistical tests of randomness, computational efficiency, a long period, and reproducibility of the sequence.

There exist many methods for generating random numbers on a computer, the most popular one being the linear congruential generators. Linear congruential generators are based on the following recurrent formula:

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

When a, c and m are integers. With $n \geq 0$, $m > 0$ and $0 < a < m$.

The value $m > 0$ is called the modulus, a is the multiplier, and c is an additive constant. Ref. [5] describes in great detail how to pick suitable values for these parameters. The sequence clearly has a maximum possible period of m. The linear congruential generators are very popular among researchers and most mathematical software packages.

So-called lagged-Fibonacci generators are also widely used. They are of the form:

$$X_n = (X_{n-r} \ op \ X_{n-p}) \bmod m \qquad (2)$$

The integer's numbers r and p are called lags and there are several methods for choosing them appropriately (see [5]). The operator op can be one of the following binary operators: addition, subtraction, multiplication, or exclusive or.

However, it should be noted that from the point of view of hardware implementation both congruential and lagged-Fibonacci RNGs are not very suitable; they are inefficient in terms of area occupation and execution time when applied to fine-grained massively parallel machines, for built-in self-test, or for other on-board applications.

A third widespread type of generator is the so-called Linear Feedback Shift Register (LFSR) generators. A pseudo random sequence is generated by the linear recursion equation:

$$X_n = (c_1 X_{n-1} + c_2 X_{n-2} + \ldots + c_k X_{n-k}) \bmod 2. \qquad (3)$$

With $c_1, c_2, \ldots, c_k$ are in $\{0,1\}$

Linear feedback shift registers are popular generators among physicists and computer engineers. There exist forms of LFSR that are suitable for hardware implementation. M. D. Galanis [4] described a comparison of stream generator based on LFSR (A5/1 and W7). Their results suggest that A5/1 generator achieves the best hardware performance. The throughput of W7 generator implementation is much

better compared to the one that the A5/1 implementation achieves.

Pseudorandom number generation by cellular automata (CAs) has been an active field of research in the last decade [1], one of the underlying motivations stemming from the advantages offered by CAs when considered from a VLSI viewpoint: CAs are simple, regular, locally interconnected, and modular [2]. These characteristics make them easier to implement in hardware than other models, thus making CAs an attractive choice for onboard applications. CAs have tradionally been used to implement RNGs in cryptographic devices [3].

One dimensional CAs RNG has been extensively studied in the past [1, 4, 5]. These studies have shown convincingly the suitability of CA-generated pseudorandom numbers and their superiority with respect to other widely used methods.

In this paper, we described and implemented a single key cryptographic system based on 1-D CA on a reconfigurable hardware platform FPGA, and we compared the architecture design and Hardware implementation of 1-D CA generator and stream generators (A5/1 and W7).

This paper is organized as follows. In the next section we summarize works done on CAs for random number generation. Section 3 outlines description of two representative stream generators based on LFSR, in section 4 we present a VLSI comparison of CA and LFSR, the comparison of implementation results is presented in section 5. Finally in section 6, we offer some concluding remarks.

2. CELLULAR AUTOMATA AND THE GENERATION OF RANDOM NUMBER

2.1 Preliminary Cellular Automata Theory

Cellular automata (CAs) are discrete dynamical systems in that space, time and properties can have only a finite number of states. A CA can be defined as a d-dimensional Euclidean space (where d = 1, 2 or 3 is used in practice), partitioned into cells of uniform size, each one embedding an identical elementary automaton (ea). Input for each ea is given by the states of the elementary automaton in the neighboring cells, where neighbourhood conditions are determined by a pattern invariant in time and constant over the cells. At the time t = 0, eas are in arbitrary states and the CA evolves changing the state of all eas at discrete times, according to a local rule. Each cell in the regular spatial lattice can have any one of a finite number of states. As mentioned before, the states of the cells in the lattice are updated according to a local rule called the state transition function. That is, the state of a cell at a given time depends only on its own state in the previous time step and the states of its nearby neighbors at the previous time step.

In this paper, we shall concentrate on d = 1, i.e. one dimensional grids. The identical rule contained in each cell is essentially a finite state machine, usually specified in the form of a rule table, with an entry for every possible neighbourhood configuration of states.

2.2 One dimensional CAs

A 1-D binary CA is an array of cells (registers) $[q_0(t), q_1(t),\ldots, q_n(t)]$ where each cell's state q_i {0, 1} and i [0, n] is any of its permissible state [6]. At each discrete time step (clock cycle) , each cell of the CA updates its state using a transition rule based on a Boolean function. Applied to the current states of each cell's state transition neighborhood $q_i(t + 1) = f_i(q_1(t), q_2(t),\ldots)$. The conventional nearest three-cell state transition neighborhood, having a radius r = 1, consists of itself q_i and its left/right most neighbors q_{i-1}/q_{i+1} Cellular automata can be uniform, with the same set of state transition neighborhood/rules are used for each cell, or hybrid, where each cell can use a different set.

Wolfram [10] first proposed CA as Pseudo-random Number Generator (PNG). He has used uniform, 1D CAs with r = 1, and rule 30, Hortensius *et al.* [7] and Nandi *et al.* [8] used nonuniform CAs with two rules 90 and 150, and it was found that the quality of generated Pseudo Number Sequences (PNSs) was better than the quality of the Wolfram system. Recently Tomassini and Perrenoud [9] proposed to use nonuniform, 1D with r = 1 and four rules 90, 105, 150 and 165, which provide high quality PNSs and huge space of possible secret keys which is difficult for cryptanalysis.

The CA which is characterized by a rule 90 specifies an evolution from neighborhood configuration to the next state as follows:

$Q_{i-1}Q_iQ_{i+1}(t)$	111	110	101	100	011	010	001	000
$Q_i(t+1)$	0	1	0	1	1	0	1	0

Decimal 90 (Rule 90)

In fig. 1 we present a five-bit Uniform Cellular Automata (UCA) implemented in hardware with the rule 90, the next state of the ith cell depends on the present states of its left and right neighbors.

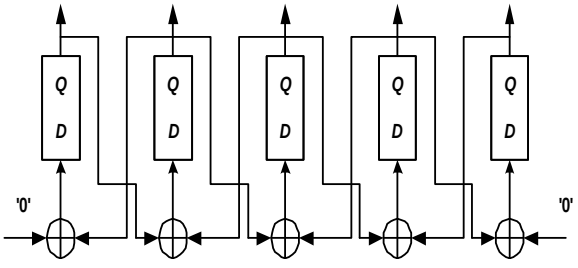


Figure 1: Five-bit UCA 90 Implemented in Hardware

The rules 90, 105, 150 and 165 are respectively characterized by these Boolean equations:

- Rule 30 : $x_i(t + 1) = x_{i-1}(t) \oplus [x_i(t)$ **OR** $x_{i+1}(t)]$.
- Rule 90 : $x_i(t + 1) = x_{i-1}(t) \oplus x_{i+1}(t)$.
- Rule 105: $x_i(t + 1) = x_i(t) \overline{\oplus} [x_{i-1}(t) \oplus x_{i+1}(t)]$.
- Rule 150: $x_i(t + 1) = x_{i-1}(t) \oplus x_i(t) \oplus x_{i+1}(t)$.
- Rule 165: $x_i(t + 1) = x_{i-1}(t) \overline{\oplus} x_{i+1}(t)$.

Where $\oplus$ and $\overline{\oplus}$ are respectively the exclusive and not exclusive OR function.

If in a CA the same rule is applied to all cells, then the CA is called uniform one (figure 1); otherwise the CA is called hybrid CA (figure 2). For rule 150 the next state of the ith cell depends on the present states of its left and right neighbors and on its own present state.

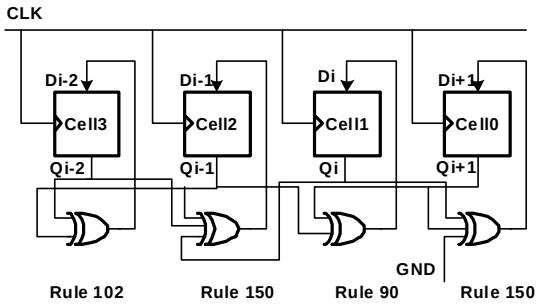
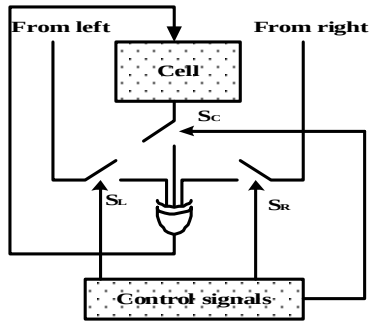


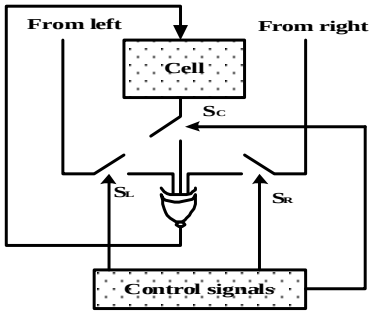
Figure 2: Hybrid Cellular Automata

The CA is characterized by XOR and/or XNOR dependence is called an additive CA. If in a CA the neighbourhood dependence is XOR, then it is called a non complemented CA and the corresponding rule is referred to as a non complemented rule [11]. For neighbourhood dependence of XNOR, the CA is called a complemented CA. The corresponding rule involving the EXNOR function is a complemented CA, single or multiple cells may employ a complemented rule with XNOR function.

Programmable CA (PCA) is a structure where the combination logic (CL) of each cell is not fixed but it's controlled by a number of control signals such that different rules can be realized on the same structure.



(A)



(B)

Figure 3: A 3-neighborhood PCA with a Non Complemented Additive Rule (A) and with a Complemented Additive Rule (B)

In figure 3 we present a standard 3-neighbrhood with complemented and non-complemented additive rules.

Table 1
PCA Rules

Control Signals Signals			*Non complted Rules*	*Complemented Rules*
S_L	S_C	S_R		
0	0	1	X_R	not (X_R)
0	1	0	X_C	not (X_C)
0	1	1	$X_C \oplus X_R$	$X_C \overline{\oplus} X_R$
1	0	0	X_L	not (X_L)
1	0	1	$X_L \oplus X_R$	$X_L \overline{\oplus} X_R$
1	1	0	$X_L \oplus X_C$	$X_R \overline{\oplus} X_C$
1	1	1	$X_L \oplus X_C \oplus X_R$	$X_L \overline{\oplus} (X_C \oplus X_R)$

Using such a cell structure like those shown in figure 3, all possible additive rules can be achieved. The combinations of the control signals of S_L, S_C, S_R and the corresponding rules are listed in Table1, where X_R is the value of the right neighbor, X_L is the value of the left neighbour and X_C represents the value of the cell.

In the following section we present the reconfigurable architecture of the RNG based on Programmable Cellular Automata (PCA).

2.3 Reconfigurable Architecture of the PCA Generator

In this Section, we present a reconfigurable architecture of RNG, which is capable of implementing all PCA rules for 3-neighborhood. Since rules selections are commonly described as 4-bit words, the selected rules refer to the local rules and types of run, where it is possible to change the rule during the evolution.

This architecture consists of:

- FIFO for storing the initial state
- A Control Unit
- A 1-D PCA with 128 cells for generating the stream sequence number.
- A 128 bits memory for storing the current state of CA (key).
- A control signals is composed by four bits (.$R(i)$, where $0 \le i \le 3$). The bit 0 of rules ($R(0)$) indicates complemented ($R(0) = 0$) from non complemented ($R(0) = 1$) additive rules. The genome of a cell is given by the 3-bit string ($R(0)$, $R(1)$ and $R(3)$) LCR. Where L (Left), C (Center) and R (Right).
- A Parallel / Serial Converter for the output sequence number.

The PCA generates different stream key. The generation of the random numbers is synchronous with the main clock signal (CLK). Figure 4 shows our proposed architecture of RNG based on PAC.

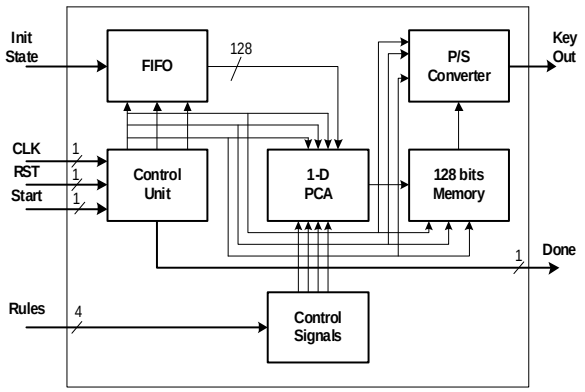


Figure 4: Our Proposed Architecture of PCA RNG

The length of a CA's state cycle is very important in determining the suitability of the CA as a generator of random number [2]. Ideally, an arbitrary n-cell CA RNG should have a maximum cycle length about 2*n*-1.

2.3 Test Results

The random number sequences produced by the CA as described above are coded in hexadecimal form. We used two tests: the DIEHARD and the NIST Test Suites values. These two tests were performed on files of 1O Mega bits. The tests are only briefly described herein since the corresponding programs and documentation are freely available on the web [14].

Table 2 shows that DIEHARD tests results of our PCA generator. In this table three CAs for the length 128 bits with different rules and the same genome are presented.

According to table 2, high statistical quality of the random sequences is generated by the PCAG. Nevertheless, the quality of the random sequences depends on the transition function (rule).

Table 2
Diehard Tests Results

Test name	*CA1*	*CA2*	*CA3*
Birthday spacing	Pass	Pass	Pass
Overlapping permu	Fail	Pass	Pass
Binary rank 31*31	Pass	Pass	Pass
Binary rank 32*32	Pass	Pass	Pass
Binary rank 6*8	Pass	Pass	Pass
COUNT-THE-1	Pass	Fail	Pass
Parking lot	Pass	Pass	Pass
Minimum distance	Pass	Pass	Pass
3D sphere	Pass	Pass	Pass
the SQUEEZE	Pass	Fail	Pass
Overlapping sum	Pass	Pass	Pass
Run up 1	Pass	Pass	Pass
Run up 2	Pass	Pass	Pass
Run down 1	Pass	Pass	Pass
Run down 2	Pass	Pass	Pass
Craps of throws	Pass	Pass	Pass
Craps of wins	Pass	Pass	Pass

In the next section we described two types of stream generators based on LFSR.

3. STREAM GENERATORS

3.1 A5/1 Generator

The A5/1 generator is composed by three linear feedback shift register (LFSR); R1, R2 and R3 of lengths 19, 22 and 23, respectively. All of these registers have primitive feedback polynomials and each register is updated according to its own feedback polynomial.

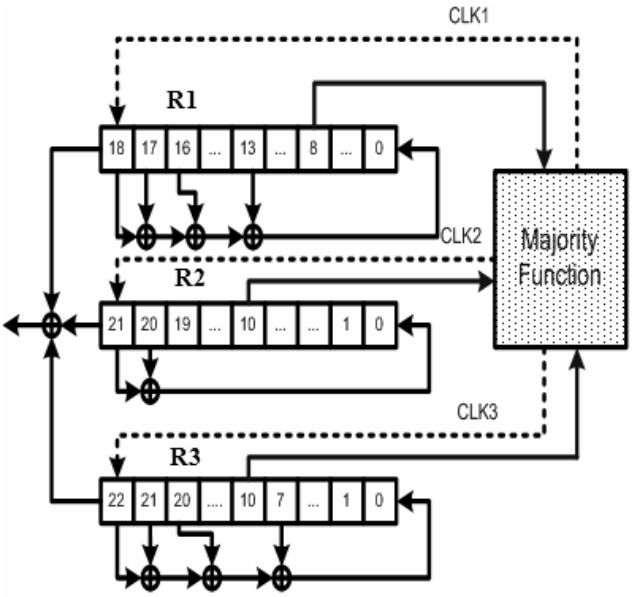


Figure 4: The A5/1 Stream Generator

The taps of R1 are at the bit positions 13, 16, 17, 18; the taps of R2 are at the bit positions 20, 21; and the taps of R3 are at bit positions 7, 20, 21, 22. These registers are maximal length LFSRs with periods 2^{19} –1, 2^{22} - 1, and 2^{23} –1 one respectively. The output of A5/1 is produced by XOR'ing the most significant bit (MSB)of each register as shown in Fig. 4.

Each LFSR has a single clocking tap in bit 8 for R1, bit 10 for R2 and R3. Clocking mechanism of each LFSR is determined according to the majority rule: Each clock cycle majority of CLK1, CLK2, and CLK3 is calculated. Two or three registers whose clocking tap value is the same as the majority bit are clocked. Since at each clock cycle at least two LFSRs are clocked, an individual LFSR moves with probability 3/4 and stops with probability 1/4.

3.2 W7 Generator

The W7 generator is a byte-wide, synchronous stream cipher optimized for efficient hardware implementation at a very high data rates. It is a symmetric key algorithm supporting key lengths of 128 bits. W7 cipher contains eight similar models or cells M1, M2,…, M8. Each cell consists of three LFSRs of lengths 38, 43 and 47, respectively. And one

majority function [4]. The three registers are maximal length LFSRs with periods $2^{38}-1$, $2^{43}-1$, and $2^{47}-1$, respectively.

The proposed architecture for the hardware implementation of one cell is presented in Fig.5. The outputs of each cell composed the key stream byte.

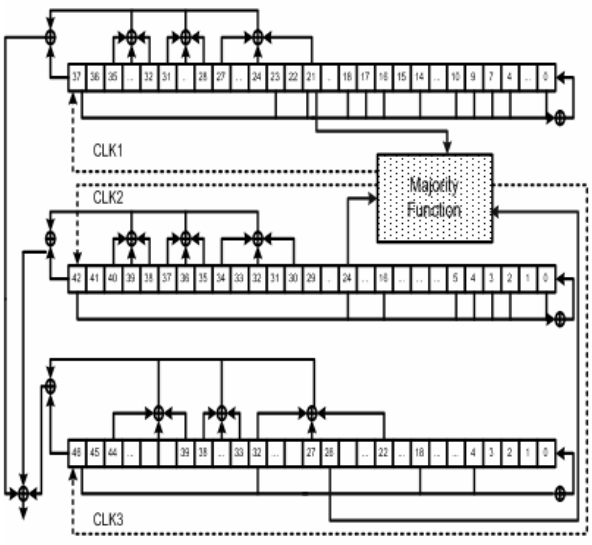


Figure 5: The W7 Stream Generator

One bit in each register is designated as the clock tap for that register, as it is shown in Fig. 5. At each clock cycle the majority value for these taps determines which LFSRs advance. Only the LFSR, whose clock taps agree with the majority, advance. The output bit arises after a non-linear function in the register which is a combination of several bits in the LFSR, as presented in Fig. 5. The non-linear function is a combination of the logical-AND functions. The actual key stream output is taken as the exclusive-OR of the three LFSRs. The key stream byte is the aggregation of each cell output.

4. THE VLSI IMPLEMENTATION OF CA AND LFSRS

In the VLSI implementation [12], a signal path can be modeled as a network which is consists of resistors and capacitors (figure 6).

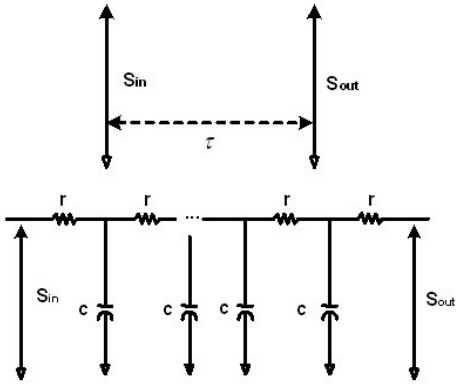


Figure 6: Distributional Resistors and Capacitors Equivalence of Signal Path in VLSI

The delay time of signal is described by $\tau = \frac{rcl^2}{2}$ (6)

Where l is the length of signal path, r is a resistance and c is a capacitance.

The delay time of signal path in VLSI means that length of signal path becomes the principal issue in VLSI implementation. We named it as the discipline of locality [13], which means that shorter signal path will bring higher speed than the longer signal path. Contrasting to the LFSR, the cellular automata have the property of locality of signal path, so the cellular automata have more potential advantages in speed than LFSR in VLSI implementation.

In figure 7 we present the structure of cellular automata and LFSR. In which the D_i is the D Flip-Flop, CL is a combinational logical, while a_i is a logical of 0 or 1.

For the LFSR the stream output is S_i, Where $0 \le i \le N-1$.

According to figure 7, the information processing speeds is determined mainly by the length of the signal path [15]. With the locality of signal path, the signal path length of the CA is related to the distance (d) of two flip-flops while the signal path length of LFSR is related to the distance of N flip-flops due to the global signal path of LFSR. So the signal path length ratio of the CA to the LFSR is given by:

$$l_{ca/lfsr} = \frac{d}{(N-1)d} = \frac{1}{N-1} \qquad (7)$$

Equation (7) denotes that the signal path length of LFSR is N-1 times of the signal length of cellular automata appropriately. From (6), we obtained that:

$$\frac{\tau_{ca}}{\tau_{lfsr}} = \left(\frac{rcd^2}{2}\right) \Big/ \left(\frac{rc((N-1)d)^2}{2}\right) = \frac{1}{(N-1)^2} \qquad (8)$$

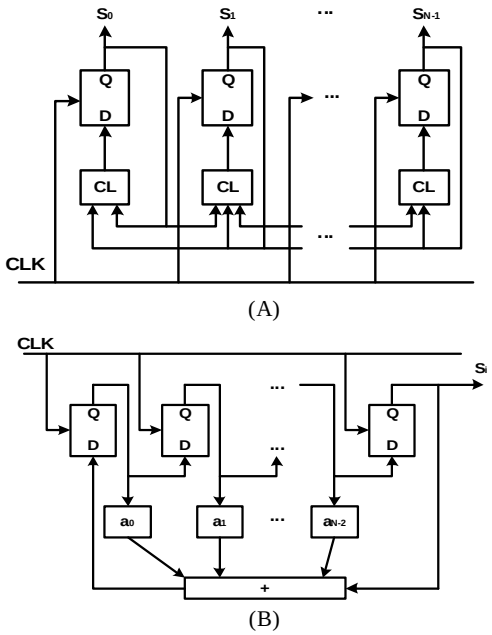


Figure 7: The Structure of a Cellular Automata (A) and LFSR (B)

Equation (8) denotes that the ratio of the information processing speeds of cellular automata to LFSR is inverse to the square of unit number as $\frac{1}{(N-1)^2}$. This means that cellular automata have more potential speeds advantages than LFSRs when considerations principally focus on the signal path length properties, especially in the large system.

5. COMPARISON OF IMPLEMENTATION RESULTS

We have prototyped this three architectures of RNGs on an FPGA device. There are implemented in VHDL language with use of the ModelSim Simulator 6.2 and synthesized using Xilinx ISE™ Tools. Results have been synthesized using a reconfigurable hardware platform Virtex II 2V250FG256 FPGA.

Table 3
RNGs Performance and Area Comparison

RNG	*Area (Slices)*	*Throughput (Mbps)*	*Throughput/ Area*
PCA	715	1947.88	2.72
A5/1	40	391.38	9.78
W7	656	1545.00	2.35

The results of performance (in terms of throughput), consumed area (CLB slices) and Throughput to Area, for the implemented RNGs, are presented in Table 3.

The results for the throughput-to-area ratio for all generators are graphically shown in Figure 8. The A5/1 achieves the best hardware performance.

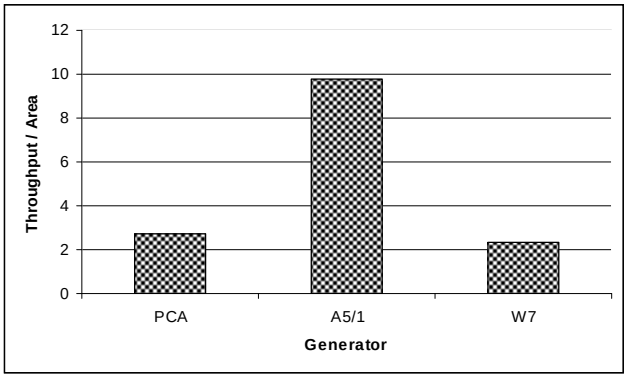


Figure 8: Throughput to Area Ratio Results

The throughput of W7 implementation has the second best throughput. It is much better compared to the A5/1 implementation, but this comes with an area cost. The comparison results for the throughput for all generators are shown in Figure 9.

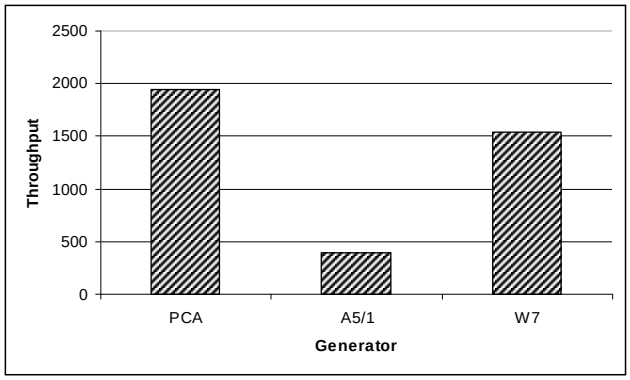


Figure 9: Throughput Results

As can be noticed, the cellular automata generator has more potential speeds advantages than the stream generators. In table 4 we present the comparison of our implementation results for stream generators (W7 and A5/1) and the implementation results published in [4].

Table 4
Implementation Results Comparison

Generator	*Criteria*	*Our Results*	*[4] Results*
	Area (Slices)	40	32
A5/1	Throughput (Mbps)	391.38	188.3
	Throughput/ Area	9.78	5.88
W7	Area (Slices)	656	608
	Throughput (Mbps)	1545	768
	Throughput/ Area	2.35	1.26

According to table 4, our proprietary implementation of stream generator is faster, it has the best throughput was by about 1545Mbps for W7 and 391.38 Mbps for A5/1, and the throughput-to-area ratio are 2.35 for W7 and 9.78 for A5/1. Also, result of [4] has the best area 40 for A5/1 and 656 for W7.

To our knowledge there are no published hardware implementations results for PCAG, which can be compared with our respective implementations.

6. CONCLUSION SUMMARY

In this paper, we described the single key cryptographic system based on PCA generator and stream generators. We proposed reconfigurable architecture of PCA generator. These generators are implemented in hardware and compared in terms of performance and consumed FPGA area. These ciphers were coded in VHDL language and synthesized in an FPGA device. The PCA generator achieves the largest throughput 1947.88 Mbps. The largest throughput-to-area ratio has been achieved by the A5/1 cipher and is equal to 978 Mbps/slice. The proposed architecture of PCA generator supports variable number of cells and variables rules. Therefore we can generate a long

number sequence with long period compared to that of the LFSRs generator, with a good statistical quality.

REFERENCES

[1] P. P. Chaudhuri, D. R. Chowdhury, S. Nandi, and S. Chattopadhyay, "Additive Cellular Automata: Theory and Application", *IEEE Computer Society*, **1**, 1997.

[2] M. Tomassini, M. Sipper, M. Perrenoud, "On the Generation of High-Quality Random Numbers by Two-Dimensional Cellular Automata", *IEEE Transactions on Computers*, **49**(10), 291-305, 2000.

[3] D. R. Chowdhury, I. S. Gupta, and P. P. Chaudhuri, "A Class of Two-Dimensional Cellular Automata and Applications in Random Pattern Testing", *J. Electronic Testing: Theory and Applications*, **5**, 65-80, 1994.

[4] M. D. Galanis, P. Kitsos, G. Kostopoulos, N. Sklavos, O. Koufopavlou, and C. E. Goutis, "Comparison of the Hardware Implementation of the Stream Ciphers" Proceedings of 11th IEEE International *Conference on Electronics, Circuits and Systems (ICECS'04),Tel-Aviv, Israel*, December 13-15, 2004.

[5] D. E. Knuth, "The Art of Computer Program-ming": **2**, *Semi numerical Algorithms*, Addison-Wesley, Reading, MA, 3rd ed., 1998.

[6] Sheng-Uei Guan and Syn Kiat Tan, "Pseudorandom Number Generation With Self-Programmable Cellular Automata", *IEEE Transactions on Computers-Aided Design of Integrated Circuits and Systems*, **23**, 1095-1101, 2004.

[7] P. D. Hortensius, R. D. McLeod, and H. C. Card, Parallel Random Numbers VLSI Systems using Cellular Automata, *IEE Transactions on Cmputers*, **38**(10), 1,466-1,473, 1989.

[8] S. Nandi, B. K. Kar, and P. P. Chaudhuri, Theory and Application of Cellular Automata in Cryptography, *IEEE Transactions on Computers*, **43**, 1,346-1,357, 1994.

[9] M. Tomassini and M. Perrenoud, "Cryptography with Cellular Automata", *Elsevier, Applied Soft Computing*, **1**, 151-160, (2001).

[10] S. Wolfram, "Cryptography with Cellular automata," in Advances in Cryptography—Crypto'25 (Springer-Verlag Lecture Notes in Computer Sciences 218), pp. 429-432, 1986.

[11] P. P. Chaudhuri, D. R. Chowdhury, S. Nandi, and S. Chattopadhyay, Additive Cellular Automata: Theory and Application, **1**. Los Alamitos, Calif.: IEEE CS Press, 1997.

[12] Xie Yongrui, Introduction to VLSI, Beijing, Tseinghua University Press, 2002.

[13] K. Charles L. Seitz, "Concurrent VLSI Architectures", *IEEE Transactions on Computers*, **C.33** (12), 1247-1265, 1984.

[14] *http://stat.fsu.edu/~geo/diehard.html.*

[15] Z. Chuanwu, L. Libin, "VLSI Characteristic of Cellular Automata and LFSR", *Proceedings of ISCIT 2005*, pp. 997-1000, 2005.