FPGA based hardware and device-independent implementation of chaotic generators

Abdelkader Senouci, Hamza Bouhedjeur, Kamal Tourche, Abdelkrim Boukabou

PII: S1434-8411(17)31078-6
DOI: http://dx.doi.org/10.1016/j.aeue.2017.08.011
Reference: AEUE 52007

To appear in: *International Journal of Electronics and Communications*

Received Date: 3 May 2017
Accepted Date: 7 August 2017

Please cite this article as: A. Senouci, H. Bouhedjeur, K. Tourche, A. Boukabou, FPGA based hardware and device-independent implementation of chaotic generators, *International Journal of Electronics and Communications* (2017), doi: http://dx.doi.org/10.1016/j.aeue.2017.08.011

FPGA based hardware and device-independent implementation of chaotic generators

Abdelkader Senouci[a,c,*], Hamza Bouhedjeur[b], Kamal Tourche[b], Abdelkrim Boukabou[c]

[a] *Department of Computer and Information Sciences, Northumbria University, Pandon Building Newcastle upon Tyne NE4 9XL United Kingdom*
[b] *Department of Electronics, ESACH, 16000, Algiers, Algeria*
[c] *Department of Electronics, Jijel University, 18000, Jijel, Algeria*

Abstract

In this paper we will describe the use of Matlab/Simulink without third party toolboxes as a single integrated environment to study and generate the HDL description of chaotic systems for rapid prototyping purposes. We will provide a review of different approaches to implement chaos generators in both analogue and digital forms with a focus on digital one. We will present a rapid prototyping method of digital chaos generators on FPGAs that is of great use for a seamless study, simulation and implementation of those systems in one environment. This will ease the design phase of the chaotic systems and will provide more flexibility for their study. Multiple chaos generators, such as Lorenz, Rössler, Chua, Linz-Sprott, and Sprott types have been designed using this method to showcase its effectiveness. In addition, for a better investigation of the chaotic dynamics, this work makes use of an implementation scheme that permits the visualization of the 3D chaotic attractor on the oscilloscope. The main focus of this paper is to provide a common implementation framework of chaos generators to target various applications, mainly cryptographic one. Newcomers to the field will easily dive in and active researchers will be able to speak the same language and build upon one anothers works more efficiently by adopting the presented guidelines.

Keywords: Chaos, Euler method, Fixed-point, FPGA, HDL Coder, ODE, Runge-Kutta

*Corresponding author
Email addresses: abdelkader.senouci@northumbria.ac.uk (Abdelkader Senouci), zizoumail@gmail.com (Hamza Bouhedjeur), tkamalmail@gmail.com (Kamal Tourche), aboukabou@univ-jijel.dz (Abdelkrim Boukabou)

1. Introduction

Deterministic non periodic flow: what a concise and self-contained description Edward Lorenz gave [1] for the phenomenon known nowadays as deterministic chaos, or chaos for short. Since its discovery by Edward Lorenz in 1959 [2], deterministic chaos has been extensively investigated and potential applications exploiting chaotic dynamics in different fields have emerged. For control theory, it is sometimes sought to control or suppress the chaotic behavior that dynamical systems exhibit [3, 4]; some other times, chaos itself is used for control purposes [5].

For communications, chaos theory has been extensively investigated and many brand new chaos-based communication schemes have been reported : optical links using chaotic pulse positioning [6], wireless links transmitting chaotic signals [7], spread-spectrum communication systems using chaotic signals [8], secured communication systems and links encrypted with chaos [9, 10, 11, 12, 13]. All these systems and approaches revolve around one core element which is the source of chaotic dynamics: the chaos generator. Many chaos generators have been discovered whether by modeling real physical systems [14] and phenomena [1], or by tweaking abstract mathematical equations to generate chaos [15]. The chaotic dynamics can be improved by finding the appropriate values that provide high unpredictability. This can be performed by maximizing the Lyapunov exponent as shown in [16].

During the last decades, chaos generators have been implemented in both analogue and digital forms to serve the above-mentioned applications. On the analogue side, continuous-time and discrete-time chaos generators have analogue realizations like the recent work in [17]. Analogue implementations consist in converting the set of ODEs representing the chaotic generator into an electronic circuit description [18] and, with appropriate parameters, the circuit oscillates in a chaotic regime. Other circuit implementations are constructed around operational amplifiers [19] or discrete transistors [20, 21]. However, chaos generators in the form of integrated circuits (IC) have seen the light too [22, 23, 24, 25]. However, analogue devices have several drawbacks like frequency limitations as demonstrated in [26]. Digital implementations, on the other hand, consist in approximating the chaotic response of the system through the resolution of the ODEs.

Real-time digital implementations have been performed on many different target devices: microprocessors (μP) [27], digital signal processors (DSP) [28], microcontrollers (μC) [29, 30] and Field Programmable Gate Arrays (FPGA) [31, 32, 33, 34]. The difference being that in processor-based implementations, (software) numerical resolution is made through an algorithm using some programming language, usually C/C++. For the FPGA case [35], the numerical solution is hard-wired in the device and its description is made through a Hardware Description Language (HDL), generally being VHDL [36] or Verilog [37].

The digital implementations mentioned above use different approaches to implement chaos generators. Dmitriev et al. [28] proposed the use of algorithms for the generation of chaotic signals that are implemented on Digital Signal

Processors (DSP) through the use of Fixed-point arithmetic and Runge-Kutta resolution procedure. The hardware used is 16 bits ADSP-2181 DSP and AD-1843 DAC. Bocheng et al. [29] implemented a hyperchaotic generator on a microcontroller using the Euler resolution procedure. The hardware used is an 8 bits ATmega128 μC and AD420 DAC. The two methods are based on manually coding the lines describing the chaos generators.

Makkey [27] proposed a quite automated method to implement chaos generators in real time through MATLAB/Simulink environment. First, Simulink blocks are used to obtain a model representing the chaotic system. Then the system model is simulated and adjusted and then converted to a real time model using the Real-time Workshop and a C++ compiler. Finally, the Real Time Windows Target is used to link the I/O card (Lab-PC-1200) and the real time program to visualize the implementation results.

For hardware implementations, the most straightforward way to implement chaos generators is to manually write the solution to the chaotic equations in HDL as done in [33, 38]. This method offers less flexibility when changing parameters and initial conditions, since they are represented in either binary or hexadecimal format. It is therefore not suitable for rapid prototyping.

Castro and Taborda [39] proposed the use of Labview-FPGA for the rapid prototyping of chaos generators on a PCI-7831R board containing a Virtex II FPGA. Whilst being a suitable solution for rapid prototyping, this method necessitates special hardware and software not accessible to everyone.

For the sake of rapid prototyping of digital systems, both Xilinx and Altera developed Matlab/Simulink Libraries for their target devices [40, 41].

Aseeri and Sobhy [31] and others [4, 42] used Xilinx System Generator to convert a Simulink model, representing a chaos generator, to a synthesizable model for implementation on Xilinx devices. Xunzhang et al. [43] and others [44] proceed the same way using DSP Builder for Altera devices.

As stated in [45], Xilinx System Generator method does not allow certain specific changes and presents some limitations. The same can be said on Altera DSP Builder. Furthermore, these prototyping methods are device-dependent, not interchangeable, and costly.

The aim of this paper is to present a rapid prototyping method that is device-independent and widely accessible for the study, simulation and implementation of chaos generators in one single environment. The method relies on MATLAB HDL Coder and Fixed Point toolbox to obtain a synthesizable description of chaotic models constructed under Simulink. The Simulink environment allows full control of all system parameters from the integration time step, initial conditions and parameters of chaotic system, to the number representation format. This permits to perform extensive study on all the aspects affecting the dynamics of the chaotic system with great flexibility. Furthermore, the studies and simulations performed in Simulink are fully descriptive of the implemented chaotic system.

The rest of the paper is organized as follows. Section 2 provides a mathematical description of continuous-time chaos generators. Section 3 describes in details the step-by-step prototyping procedure. Section 4 presents the results of

multiple chaos generators implemented using the proposed method, and finally, concluding remarks are drawn in Section 5.

2. Chaos Generators Description

Chaos generators are a particular class of dynamical systems. A dynamical system is one whose state changes in time. If the changes are determined by specific rules, rather than being random, the system is said to be deterministic [2]. The changes can occur at discrete time steps or continuously. Describing a dynamical system mathematically leads to either difference or differential equations [46]. Since the implementation of discrete-time chaos generators is straightforward; we will be mainly concerned with continuous-time ones that literature cannot seem to provide a unified way of implementing them in digital form, knowing that most remarks and procedures are applicable to discrete-time systems too.

In order to implement a chaos generator in any form, a mathematical model is required. One of the mathematical descriptions of dynamical systems, most naturally adopted for behavioral modeling, is the state-space representation [18]. In the state-space approach, the differential equations are of first order in the time-derivative, so that the initial values of the variables will suffice to determine the solution [18]. In general, the state-space description is given in the form:

$$\dot{s} = f(s, u, t). \tag{1}$$

$$o = h(s, u, t). \tag{2}$$

where s is the vector of state variables; u is an input vector (in forced systems); t denotes time; f and h are functions, generally nonlinear; and o is the output vector.

The set of all trajectories, the solution of the equations form, provide a complete geometrical representation of the dynamical behavior of the system, under the specified conditions. This is called the phase portrait. Generally, the solution of the equations cannot be obtained analytically, so that, in order to construct the trajectories, it is necessary to either translate the system description into an electronic circuit or use numerical methods to solve the equations [18]. The latter will be of interest through the rest of the paper.

3. Prototyping Procedure

In order to construct the phase portrait of the chaos generator, we will describe a rapid prototyping method that will allow us to implement any system on any FPGA, and visualize the chaotic attractor on an oscilloscope; as depicted on Fig.1. The chaos generator entity will be an HDL file that is generated in the Matlab/Simulink environment and integrated in the target device synthesis environment: ISE for Xilinx devices, Quartus for Altera devices or others. The state variables will be then routed to the inputs of two Digital to Analogue

Converters interfaced with the FPGA. Since we can only visualize 2D phase portraits on commercially available oscilloscopes through the X-Y display mode, the plane selector will project the high dimension attractor on a 2D plane: for a generator of dimension 3, the plane could be x-y, x-z, y-z, or any plane in between. Next, we will describe the step-by-step procedure to study and sim-

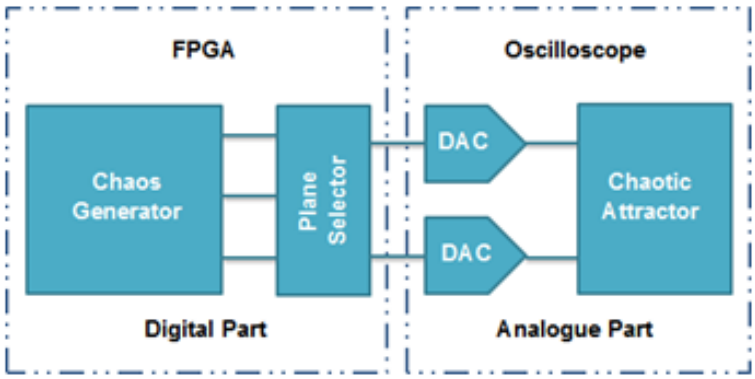


Figure 1: Hardware prototyping architecture.

ulate any chaotic system in the Matlab/Simulink environment and obtain its HDL description file. We will use the classical Lorenz system to illustrate the procedure.

The state vector s of the Lorenz system contains three variables: x, y and z:

$$s = \begin{bmatrix} x \\ y \\ z \end{bmatrix} \tag{3}$$

The output vector o equals the state vector s:

$$o = s. \tag{4}$$

The set of differential equations is as follows:

$$\begin{cases} \dot{x} = \sigma(y - x) \\ \dot{y} = rx - xz - y \\ \dot{z} = xy - bz \end{cases} \tag{5}$$

where $\sigma = 10$, $r = 28$ and $b = 8/3$ are one set of system parameters for which values chaotic behavior is manifested.

3.1. System Discretization

Since the implementation of the chaotic system will be in digital form, the solution of the equations could only be discrete and could be obtained through the choice of a numerical integration method and a finite precision number representation.

3.1.1. Numerical resolution method

In order to integrate the set of differential equations describing the chaotic systems, one could choose from many integration methods [47]. Forward Euler and Fourth order Runge-Kutta are two methods widely used in the literature [45, 48, 49].

We would like to note that, in the case of chaotic systems, we do not seek the numerical method that gives the most accurate solution, but we seek the method that gives the best chaotic behavior. Many articles discussing numerical methods used to resolve chaotic equations have been published [48, 50, 51].

Through a comparative study [48] of different numerical integration methods, based on implementation performance and the chaotic response of the systems, it was found that less complicated numerical solutions have better chaotic response, occupy less silicon area and have higher throughput. Therefore, it was concluded that the Euler technique is best suited for chaotic systems.

The authors believe that higher order integration methods are equivalent to the basic Euler method with smaller integration step. Furthermore, we believe that the exact solution cannot be obtained due to the chaotic nature of the systems to be resolved. Therefore, there is practically no need to use, higher order, more complicated numerical resolution methods in this case. Besides, as mentioned in [45, 48], the computational inaccuracy that the Euler technique suffers actually enforces the chaotic behavior of the systems.

The discrete solution in the case of Euler method applied to Eq.(1) is:

$$s_{t+\Delta t} = s_t + \Delta t f(s_t, u, t). \tag{6}$$

For the Lorenz chaotic system described by Eq.(5), the solution is:

$$\begin{cases} x_{t+\Delta t} = x_t + \Delta t * \sigma(y_t - x_t) \\ y_{t+\Delta t} = y_t + \Delta t * (r x_t - x_t * z_t - y_t) \\ z_{t+\Delta t} = z_t + \Delta t * (x_t y_{t-b z_t}) \end{cases} \tag{7}$$

3.1.2. Finite precision number representation

From a digital point of view, there is no way to provide infinite precision for real numbers. A finite precision representation has to be adopted in order to perform the computations involved in equation (6). Thus, one should opt for whether fixed point representation or floating point representation. Both representations are used in the literature [45, 51].

In this work, two's complement signed fixed point representation is adopted, which consists of an integer part coded on m bits and a fractional part coded on n bits, separated by the binary point. The total word length is then $m + n$, as depicted in Fig. 2.

3.2. System Block Description

After obtaining the set of difference equations representing the chaos generator, we move to the Simulink environment to construct a synthesizable model of the system. In order to do so, valid Fixed Point Toolbox and HDL Coder are

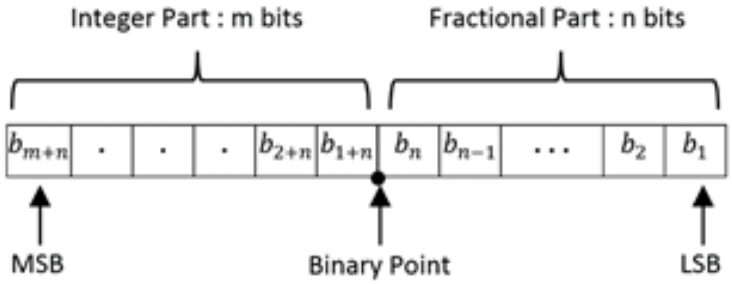


Figure 2: Fixed Point representation.

needed.

First of all, all model parameters must be defined and initialized in the Model Explorer starting with the data type. Then, under Simulation => configuration parameters, we set the solver type to fixed step and the solver to discrete-no continuous states and the fixed step (fundamental sample time) to step defined in the Model Explorer. After that, we drag and drop the Simulink building blocks needed (multiplier, adder, constant, gain, unit delay etc.) into the model and configure them by setting their data types in all data type fields. In order to make sure all data types are consistent, we choose to enable Port Data Type under Format => Ports/Signal Display.

Now, we start wiring the system according to its discrete representation. We should not forget to set the initial conditions for each unit delay block according to each variable. Fig. 3 shows the x variable integrator. For the case of Lorenz,

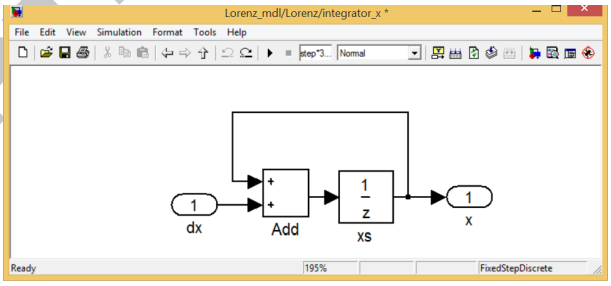


Figure 3: x variable integrator.

we obtain the following system block diagram, as depicted in Fig. 4.

After running the simulation, we obtain the following results for the case of signed fixed point representation with 32 bits word length and 24 bits fraction length. Lorenz chaotic signals using fixed point arithmetic (32Q24) are shown in Fig. 5.

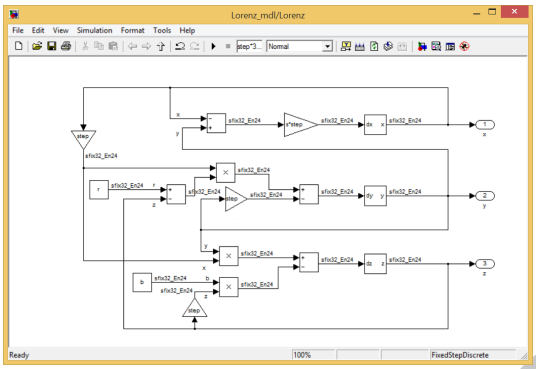


Figure 4: Lorenz Simulink model.

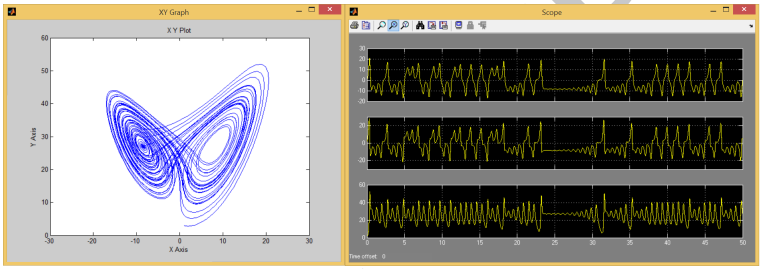


Figure 5: Lorenz chaotic signals using fixed point arithmetic (32Q24).

Note that these are the exact time series we have to obtain when we implement this model on FPGA. If we want to perform the simulation using double precision numbers, we change the type defined in Model Explorer to double precision arithmetic and check that the type has effectively changed. The obtained simulation results are shown in Fig. 6. We have also performed a simulation using

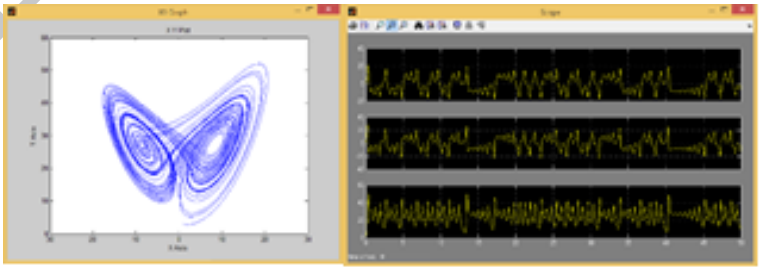


Figure 6: Lorenz chaotic signals using double precision arithmetic (64 bits).

fixed point 16Q8, and using single precision, as depicted in Fig. 7 and Fig. 8, respectively.

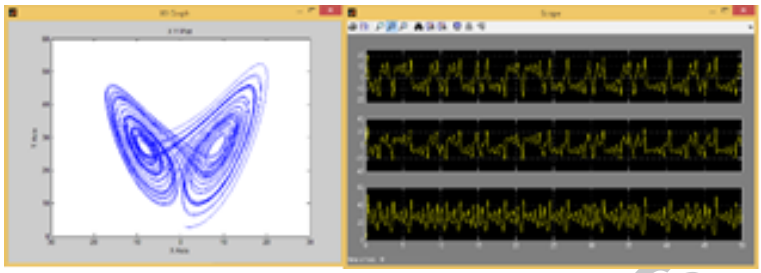


Figure 7: Lorenz chaotic signals using fixed point arithmetic (16Q8).

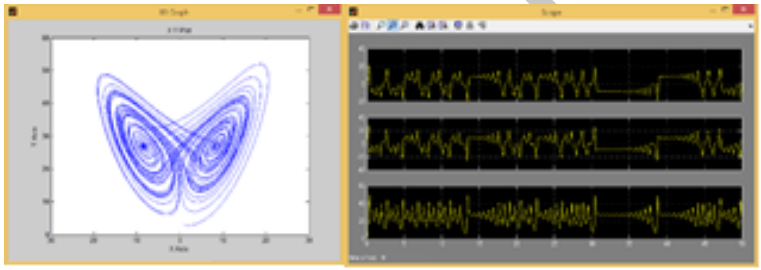


Figure 8: Lorenz chaotic signals using single precision arithmetic (32 bits).

The results are clearly different, as one should expect. This brings into attention the fact that in some cases, simulation and study are performed using a given number representation (double precision in most cases) for bifurcation diagrams and calculation of Lyaponov Exponents and the implementation are performed in another number representation (fixed point or single precision) and it is claimed that the study reflects the chaotic behavior of the implemented system. In a broad sense, this could be correct, but strictly speaking both the study and the implementation of a chaotic system should be done in the same number representation.

3.3. HDL Code Synthesis

After having the simulated and validated chaotic behavior of the system, we now proceed to obtain an HDL description of the model. To do so, with a valid Fixed Point toolbox and HDL Coder, we choose HDL Coder => Generate HDL under Tools menu.

This will generate a set of files that we need to take to our target device synthesis environment for implementation. The inspection of the code shows that it is minimal, clear and comprehensible. One can perform any modifications that

seem to be needed into the generated HDL files.
Next, implementation of VHDL code representing the Lorenz chaos generator on a Xilinx device will be shown.

3.4. Hardware Implementation

After having the simulated and validated generators in Simulink environment, we proceed to the generation of the FPGA programming file for implementation. This step is straightforward and depends on the synthesis environment being used. We create a project configured to the target device, import the HDL files generated in Matlab, set the pin constraints to a DAC, add a virtual logic analyzer, and then we synthesize the design. The overall system schematic is shown in Fig. 9. For a better investigation of the chaotic dynamics, we have implemented a number of chaos generators simultaneously. In addition, we implemented digital means that allow a clear 3D visualization of each attractor. The system manager block (SysMan) generates all the control and synchronization signals for all other blocks. It also allows changing the clock frequency of the chaos generator for better visualization on the oscilloscope. Four chaos generators are implemented at once (Lorenz, Rössler, Chua and Chen). One attractor is visualized at a time through a multiplexer block (MuxGen). Once a generator is selected, its 3D attractor is rotated through a block that implements Euler angles (Rotation). Another block then permits to zoom into the attractor (Zoom) for better visualization of the trajectories. Before feeding the signals to the DAC and to the oscilloscope, just two signals are selected through another multiplexer (MuxSig) that represents the projection of the attractor to be visualized. The signals are also fed to Chipscope. The 3D attractor is visualized on the oscilloscope through continuous sweeping of the rotation angle.

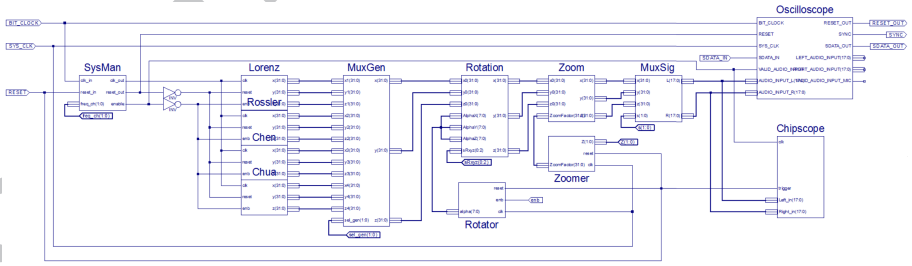


Figure 9: Implementation block diagram.

The following section will present the implementation results of different chaotic systems.

4. Real-time Implementation of Multiple Chaos Generators

The last step in the rapid prototyping approach of chaos generators is its real-time implementation. In this regard, we have implemented many chaos generators using ML507 evaluation board, and visualized the results in Chipscope virtual logic analyzer and on an oscilloscope. The test bed used for this purpose is shown in Fig. 10.

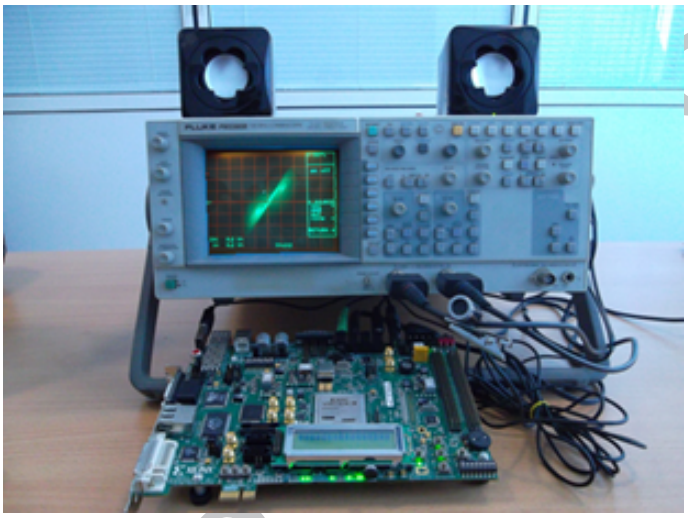


Figure 10: Implementation test bed.

Following is the implementation results of the different implemented systems. We will show both the time series and phase portraits of the implemented generators. All implementations are 32Q24 signed fixed point format. The systems contain different non-linearities that are the origin of the chaotic behavior. The initial conditions are generally $x = 1$, $y = 2$, $z = 3$ unless otherwise stated.

4.1. Lorenz chaos generator

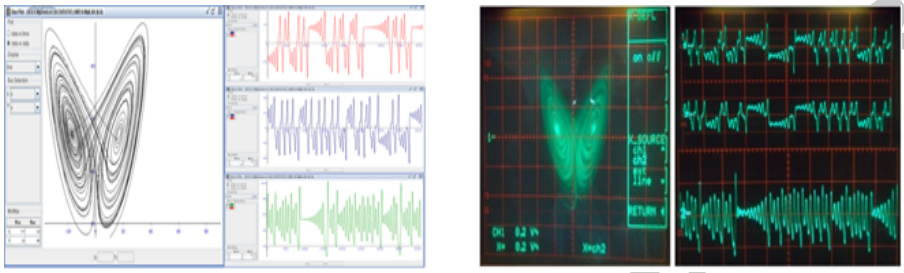
This system is found by Lorenz while modeling meteorological phenomena [1].

$$\begin{cases} \dot{x} = \sigma(y - x) \\ \dot{y} = rx - xz - y \\ \dot{z} = xy - bz \end{cases} \tag{8}$$

This set of parameters ($\sigma = 10, r = 28, b = 8/3$) yields chaotic behavior. The implementation resources used are shown in Table. 1, and real-time implementation signals are depicted in Fig. 11.

Table 1: Lorenz chaos generator

Nbre of Slice Registers	Nbre of Slice LUTs	Nbre of LUT Flip Flop pairs used	Nbre of DSP48Es	Max Frequency (MHz)
96 (0%)	500 (1%)	554	24 (18%)	44



(a) using Chipscope (b) using Oscilloscope

Figure 11: Lorenz chaotic phase portrait and time series.

4.2. Rössler chaos generator

Rössler [52] designed this system based on the Lorenz system.

$$\begin{cases} \dot{x} = -(y + x). \\ \dot{y} = -x - ay. \\ \dot{z} = b + z(x - c). \end{cases} \tag{9}$$

This set of parameters ($a = 0.34, b = 0.2, c = 5.7$) yields chaotic behavior. The implementation resources used are shown in Table. 2, and real-time implementation signals are depicted in Fig. 12.

Table 2: Rössler chaos generator

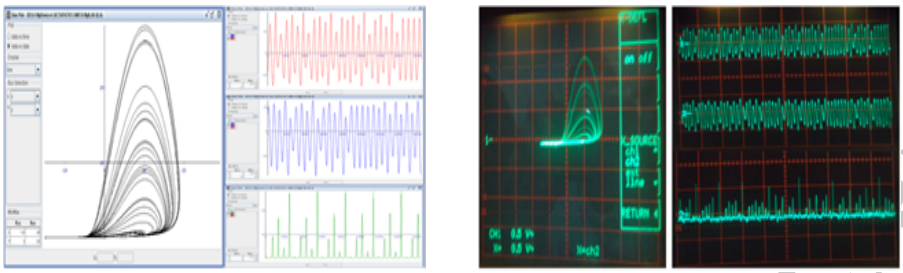
Nbre of Slice Registers	Nbre of Slice LUTs	Nbre of LUT Flip Flop pairs used	Nbre of DSP48Es	Max Frequency (MHz)
96 (0%)	364 (0%)	418	18 (14%)	43

4.3. Chua chaos generator

Chua [53] has proposed this system that represents an electronic circuit that exhibits chaos.

$$\begin{cases} \dot{x} = \alpha(y - x - g(x)) \\ \dot{y} = x - y + z \\ \dot{z} = -\beta y \end{cases} \tag{10}$$

$$g(x) = \begin{cases} m_1 x + (m_0 - m_1), & x > 1 \\ m_0 x, & |x| \leq 1 \\ m_1 x - (m_0 - m_1), & x < -1 \end{cases} \tag{11}$$



(a) using Chipscope (b) using Oscilloscope

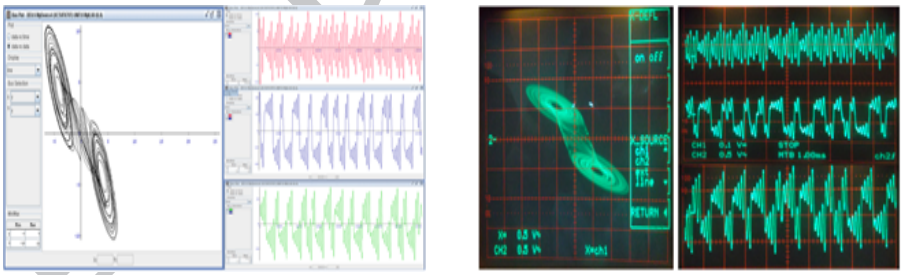
Figure 12: Rössler chaotic phase portrait and time series.

This set of parameters ($\alpha = 9, \beta = 15, m_0 = -1.3, m_1 = -0.7$) yields chaotic behavior. The implementation resources used are shown in Table. 3. Real-time

Table 3: Chua chaos generator

Nbre of Slice Registers	Nbre of Slice LUTs	Nbre of LUT Flip Flop pairs used	Nbre of DSP48Es	Max Frequency (MHz)
96 (0%)	447 (0%)	500	24 (18%)	27

implementation signals are depicted in Fig. 13.



(a) using Chipscope (b) using Oscilloscope

Figure 13: Chua chaotic phase portrait and time series.

4.4. Chen chaos generator

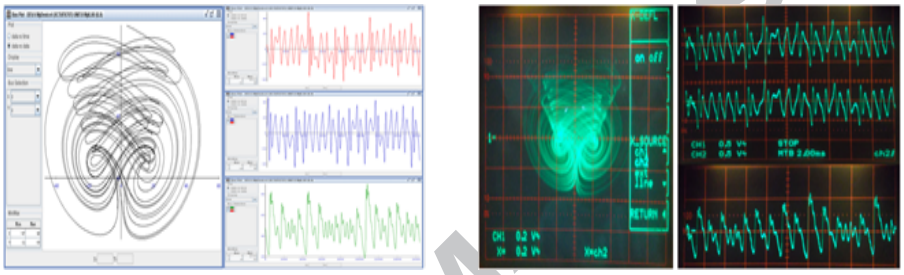
285 In the pursuit of chaos anticontrol, or choatification, Chen and Ueta [54] have found another Lorenz-based wing attractor.

$$\begin{cases} \dot{x} = a(y - x). \\ \dot{y} = (c - a)x - zx + cy. \\ \dot{z} = xy - bz. \end{cases} \tag{12}$$

This set of parameters ($a = 35, b = 8/3, c = 28$) yields chaotic behavior. The implementation resources used are shown in Table. 4, and real-time implementation signals are depicted in Fig. 14.

Table 4: Chen chaos generator

Nbre of Slice Registers	Nbre of Slice LUTs	Nbre of LUT Flip Flop pairs used	Nbre of DSP48Es	Max Frequency (MHz)
96 (0%)	349 (0%)	382	24 (18%)	43



(a) using Chipscope (b) using Oscilloscope

Figure 14: Chen chaotic phase portrait and time series.

4.5. Linz-Sprott chaos generator

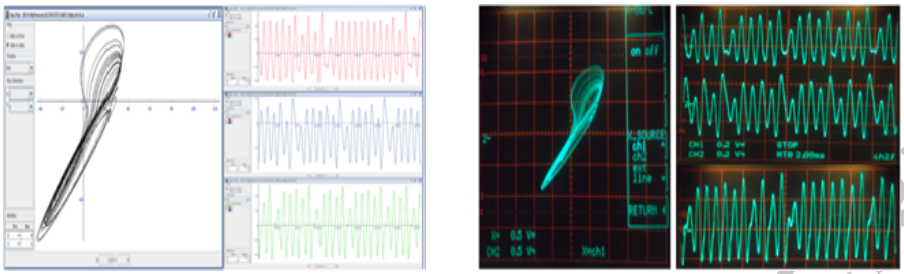
Linz and Sprott [55] worked together to find a chaotic system that is algebraically simpler. This system has only one non-linearity of type magnitude.

$$\begin{cases} \dot{x} = y \\ \dot{y} = z \\ \dot{z} = -\alpha z - y + |x| - 1 \end{cases} \tag{13}$$

This parameter ($\alpha = 0.6$) yields chaotic behavior. The implementation resources used are shown in Table.5, and real-time implementation signals are depicted in Fig. 15.

Table 5: Linz Sprott chaos generator

Nbre of Slice Registers	Nbre of Slice LUTs	Nbre of LUT Flip Flop pairs used	Nbre of DSP48Es	Max Frequency (MHz)
96 (0%)	252 (0%)	313	4 (3%)	72



(a) using Chipscope (b) using Oscilloscope

Figure 15: Linz-Sprott chaotic phase portrait and time series.

4.6. Sprott type B chaos generator

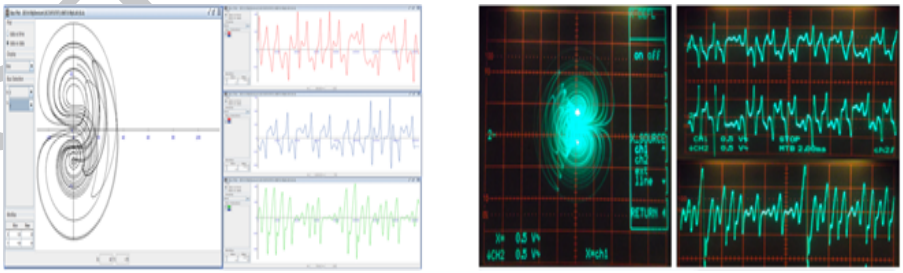
Sprott [2] has continued his research towards algebraically simple chaotic flows and has found multiple systems. Case B is one of those.

$$\begin{cases} \dot{x} = yz \\ \dot{y} = x - y \\ \dot{z} = 1 - xy \end{cases} \tag{14}$$

Table 6: Sprott type B chaos generator

Nbre of Slice Registers	Nbre of Slice LUTs	Nbre of LUT Flip Flop pairs used	Nbre of DSP48Es	Max Frequency (MHz)
96 (0%)	234 (0%)	288	20 (15%)	39

The implementation resources used are shown in Table.6, and real-time implementation signals are depicted in Fig. 16.



(a) using Chipscope (b) using Oscilloscope

Figure 16: Sprott type-B chaotic phase portrait and time series.

4.7. Sprott type E chaos generator

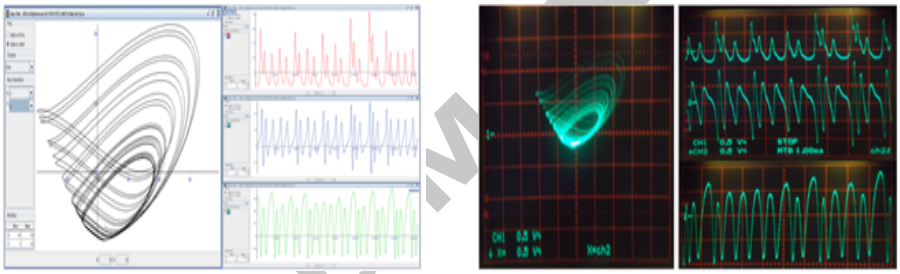
Sprott Case E [2] is another variant with slightly modified equations.

$$\begin{cases} \dot{x} = yz \\ \dot{y} = x_2 - y \\ \dot{z} = 1 - 4x \end{cases} \tag{15}$$

The implementation resources used are shown in Table.7, and real-time implementation signals are depicted in Fig. 17.

Table 7: Sprott type E chaos generator

Nbre of Slice Registers	Nbre of Slice LUTs	Nbre of LUT Flip Flop pairs used	Nbre of DSP48Es	Max Frequency (MHz)
96 (0%)	232 (0%)	275	20 (15%)	39



(a) using Chipscope (b) using Oscilloscope

Figure 17: Sprott type-E chaotic phase portrait and time series.

4.8. Sprott type H chaos generator

Sprott Case H [2] contains only one non-linearity of type power of two.

$$\begin{cases} \dot{x} = -y + z^2 \\ \dot{y} = x + ay \\ \dot{z} = x - z \end{cases} \tag{16}$$

This parameter ($\alpha = 0.5$) yields chaotic behavior. The implementation resources used are shown in Table. 8, and real-time implementation signals are depicted in Fig. 18.

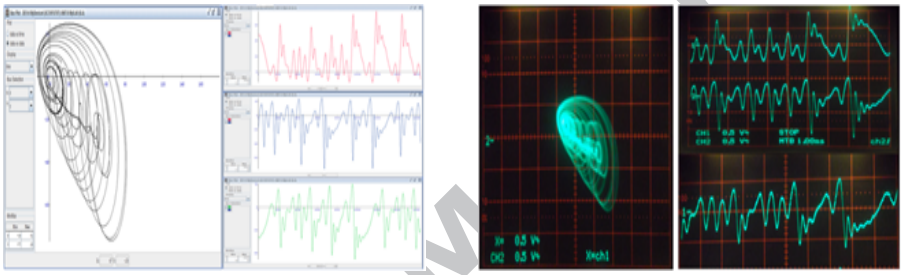
4.9. Resources Summary

The following Table. 9 compares the FPGA resources used by all implemented systems.

We can see that Chua chaotic system is the slowest in terms of maximum working frequency (27 MHz) which is due to the piece-wise linear function $g(x)$. The fastest chaotic system is Linz-Sprott (72 MHz) which is due to the simple non-linearity of type magnitude, as confirmed in [56].

Table 8: Sprott type H chaos generator

Nbre of Slice Registers	Nbre of Slice LUTs	Nbre of LUT Flip Flop pairs used	Nbre of DSP48Es	Max Frequency (MHz)
96 (0%)	229 (0%)	283	16 (12%)	40



(a) using Chipscope (b) using Oscilloscope

Figure 18: Sprott type-H chaotic phase portrait and time series.

Table 9: FPGA resources used by all implemented systems

	Nbre of Slice Registers	**Nbre of Slice LUTs**	**Nbre of LUT F F pairs used**	**Nbre of DSP48Es**	**Max Frequency (MHz)**
Lorenz	96 (0%)	500 (1%)	554	24 (18%)	44
Rössler	96 (0%)	364 (0%)	418	18 (14%)	43
Chua	96 (0%)	447 (0%)	500	24 (18%)	27
Chen	96 (0%)	349 (0%)	382	24 (18%)	43
Linz Sprott	96 (0%)	252 (0%)	313	4 (3%)	72
Sprott-B	96 (0%)	234 (0%)	282	20 (15%)	39
Sprott-E	96 (0%)	232 (0%)	275	20 (15%)	39
Sprott-H	96 (0%)	229 (0%)	283	16 (12%)	40

5. Conclusion

This paper has presented a step-by-step methodology to study, simulate and implement chaos generators for FPGA targets on the widely used Matlab/Simulink environment. The method relies on MATLAB HDL Coder and Fixed Point toolbox to obtain a synthesizable description of chaotic models constructed under Simulink. The Simulink environment allows full control of all system parameters from the integration time step, initial conditions and parameters of chaotic system, to the number representation format. This permits to perform extensive study on all the aspects affecting the dynamics of the chaotic system with great flexibility. Furthermore, the studies and simulations performed in Simulink are fully descriptive of the implemented chaotic system. Note that whilst analogue implementations of chaos generators preserve the continuity of the generated signals and yield truly chaotic behavior, digital implementations approximate the chaotic response and yield pseudo-chaotic behavior. This propriety will be investigated in future work where we will put the implemented chaos generators into practical applications.

[1] Lorenz T.E.N. Deterministic Non periodic Flow. Journal of the Atmospheric Sciences. 1963;20(2):130-141.

[2] Sprott J.C. Elegant Chaos. World Scientific Publishing, Singapore 2010.

[3] Boukabou A., Chebbah A., Belmahboul A. Stabilizing Unstable Periodic Orbits of the Multi-Scroll Chua's Attractor. Non linear Analysis: Modelling and Control 2007;12(2):469-477.

[4] Senouci A., Boukabou A. Predictive control and synchronization of chaotic and hyperchaotic systems based on a T-S fuzzy model. Mathematics and Computers in Simulation 2014;105:62-78.

[5] Arena P., Fiore S., De Fortuna L., Frasca M., Patane L., Vagliasindi G. Weak Chaos Control for Action-Oriented Perception: Real Time Implementation via FPGA. International conference on Biomedical Robotics and Biomechatronics, Biorob, Pisa, Italy 2006.

[6] Illing L. Digital communication using chaos and nonlinear dynamics. Journal of Nonlinear Analysis 2009; 71:2958-2964.

[7] Quyen N.X. On the performance of low-rate wireless correlation-delay-shift-keying system. AEU - Int J Electron Commun 2017;71:37-44.

[8] Quyen N.X., Yem V.V., Hoang T.M. A Chaos-Based Secure Direct-Sequence/Spread-Spectrum Communication System. Abstract and Applied Analysis 2013;2013:11.

[9] Zaher A., Abu-Rezq A. On the Design of Chaos-Based Secure Communication Systems. Communications in Nonlinear Science and Numerical Simulation 2011;16:3721-3737.

[10] Senouci A., Boukabou A. Chaotic Cryptography Using External Key. 7^{th} Workshop on Systems, Signal Processing and their Applications, Algeria 2011.

[11] Trejo-Guerra R., Tlelo-Cuautle E., Cruz-Hernandez C., SanchezLopez C. Chaotic communication system using chua's oscillators realized with CCII plus s. International Journal of Bifurcation and Chaos 2009;19(12):4217-4226.

[12] Tlelo-Cuautle E., Carbajal-Gomez V.H., Obeso-Rodelo P.J., Rangel-Magdaleno J.J., Nunez-Perez J.C. FPGA realization of a chaotic communication system applied to image processing. Nonlinear Dynamics 2015;82(4):1879-1892.

[13] Rajagopal K., Karthikeyan A., Srinivasan A.K. FPGA implementation of novel fractional-order chaotic systems with two equilibriums and no equilibrium and its adaptive sliding mode synchronization. Nonlinear Dynamics 2017; 87(4):22812304.

[14] Kovacic I., Brennan M.J. The Duffing Equation: Nonlinear Oscillators and Their Behavior. John Wiley and Sons 2011.

[15] Sprott J.C., Linz S.J. Algebraically Simple Chaotic Flows. International Journal of Chaos Theory and Applications 2000;5(2).

[16] Gerardo de la Fraga L., Tlelo-Cuautle E. Optimizing the maximum Lyapunov exponent and phase space portraits in multi-scroll chaotic oscillators. Nonlinear Dynamics 2014;76(2):1503-1515.

[17] Valtierra J.L., Tlelo-Cuautle E., Rodriguez-Vazquez A. A switched-capacitor skew-tent map implementation for random number generation. International Journal of Circuit Theory and Applications 2017;45(2):305-315.

[18] Sanchez-Lopez C., Munoz-Pacheco J.M. Design and Applications of Continuous-Time Chaos Generators. World's largest Science, Technology and Medicine Open Access book publisher 2011.

[19] Srisuchinwong B., Liou C.H. High Frequency Implementation of Sprott's Chaotic scillators Using Current-Feedback Op Amps. Signals, Circuits and Systems, ISSCS. International Symposium 2007.

[20] Keuninck L., Van der Sande G., Danckaert J. Simple Two-Transistor Single-Supply Resistor-Capacitor Chaotic Oscillator. IEEE Transactions On Circuits And Systems 2015;62(9).

[21] Senouci A., Ouslimani A., Kasbari A., Boukabou A. Theoretical and experimental investigation of an L-band chaotic oscillator. 2^{nd} International Conference on Computational and Experimental Science and Engineering, ICCESEN, Antalya, Turkey 2015.

[22] Rodriguez-Vazquez A., Delgado-Restituto M. CMOS design of Chaotic Oscillators using state Variables: A monolithic Chua's Circuit. IEEE Transactions on circuits and systems-II: analogue and digital processing 1993;40(10).

[23] Delgado-Restituto M., Rodriguez-Vazquez M. Integrated Chaos Generators. proceedings of the IEEE 2002;90(5).

[24] Trejo-Guerra R., Tlelo-Cuautle E., Carbajal-Gomez V.H., Rodriguez-Gomez G. A survey on the integrated design of chaotic oscillators. Applied Mathematics and Computation. 2013;219(10):5113-5122.

[25] Trejo-Guerra R., Tlelo-Cuautle E., Jimenez-Fuentes J.M., Sanchez-Lopez C., Munoz-Pacheco J.M., Espinosa-Flores-Verdad G., Rocha-Perez J.M. Integrated circuit generating 3-and 5-scroll attractors. Communications in Nonlinear Science and Numerical Simulation. 2012;17(11):4328-4335.

[26] Munoz-Pacheco J.M., Tlelo-Cuautle E., Toxqui-Toxqui I., Sanchez-Lopez C., Trejo-Guerra R. Frequency limitations in generating multi-scroll chaotic attractors using CFOAs. International Journal of Electronics 2014;101(11):1559-1569.

[27] Makkey M.Y. Implementation of real time chaotic generator models (continuous and discrete) using digital hardware. Electronics, Circuits and Systems. ICECS. Proceedings of the 10^{th} IEEE International Conference; 2003.

[28] Dmitriev A., Starkov S., Yemetz S. Chaotic Communication using Digital Signal Processors. Institute of Radio Engineering and Electronics, Russian Academy of Science; 1998.

[29] Bocheng B., Qiang X., Jianping X. multi-scroll hyperchaotic system based on colpitts model and its circuit implementation. journal of electronics 2010;27(4).

[30] Pano-Azucena A.D., Rangel-Magdaleno J.J., Tlelo-Cuautle E., Quintas-Valles A.J. Arduino-based chaotic secure communication system using multi-directional multi-scroll chaotic oscillators. Nonlinear Dynamics. 2017;87(4):2203-2217.

[31] Aseeri M.A., Sobhy M.I. Field Programmable Gate Array (FPGA) as a new approach to implement the chaotic generators. Conference: 3^{rd} International Conference on Advanced Engineering Design AED, At Prague, Czech Republic, Volume: AED003; 2003.

[32] Senouci A., Benkhaddra I., Boukabou A., Bouridane A., Ouslimani A. Implementation and Evaluation of a New Unified Hyperchaos-based PRNG. 26^{th} International Conference on Microelectronics, ICM, Doha, Qatar; 2014.

[33] Tlelo-Cuautle E., Range-Magdalen J.J., Pano-Azucena A.D., Obeso-Rodelo P.J., Nunez-Perez J.C. FPGA realization of multi-scroll chaotic oscillators. Commun Nonlinear SciNumerSimulat 2015;27:66-80.

[34] Tlelo-Cuautle E., Pano-Azucena A.D., Rangel-Magdaleno J.J., Carbajal-Gomez V.H., Rodriguez-Gomez G. Generating a 50-scroll chaotic attractor at 66 MHz by using FPGAs. Nonlinear Dynamics. 2016;85(4):2143-2157.

[35] Tlelo-Cuautle E., Rangel-Magdaleno L., Gerardo de la Fraga L. Engineering Applications of FPGAs. Springer International Publishing, 2016.

[36] Tlelo-Cuautle E., Quintas-Valles A.J., Gerardo de la Fraga L., Rangel-Magdaleno L. VHDL Descriptions for the FPGA Implementation of PWL-Function-Based Multi-Scroll Chaotic Oscillators. Plos One 2016;11(12).

[37] Valtierra Sanchez de la Vega J-L., Tlelo-Cuautle E. Simulation of Piecewise-Linear One-Dimensional Chaotic Maps by Verilog-A. IETE Technical Review. 2015;32(4):304-310.

[38] Zidan M.A., Radwan A.G., Salama K.N. Random Number Generation Based on Digital Differential Chaos. Circuits and Systems (MWSCAS), IEEE 54^{th} International Midwest Symposium; 2011.

[39] Castro H., Taborda J.A. Rapid Prototyping of Chaotic Generators using LabView-FPGA. Circuits and Systems (CWCAS), IEEE 4^{th} Colombian Workshop; 2011.

[40] Xilinx System Generator for DSP User Guide, 2008. Available at www.xilinx.com.

[41] Altera DSP Builder User Guide, 2009. Available at www.altera.com.

[42] Dabal P., Pelka R. A Chaos-Based Pseudo-Random Bit Generator Implemented in FPGA Device. Design and Diagnostics of Electronic Circuits and Systems (DDECS), IEEE 14^{th} International Symposium; 2011.

[43] Xunzhang Y.L., Sun L., Li Z. Research on New Implementation Method of Chaotic Model Based on FPGA. 7^{th} International Conference on ASIC, Guilin; 2007.p. 241-244.

[44] Shi T., Rui G., Zhang Y., Zhang S. Design Method for Duffing System Based on DSP Builder. International Conference on Systems and Informatics, ICSA2012, Yantai; 2012.p. 121-124.

[45] Antonelli M., De Micco L., Gonzalez C.M., Larrondo H.A. Analysis of the Digital Implementation of a Chaotic Deterministic-Stochastic Attractor Argentine School of Micro-Nano electronics, Technology and Applications; 2012.

[46] Feckan M. Bifurcation and Chaos in Discontinuous and Continuous Systems. Higher Education Press, Beijing and Springer-Verlag Berlin Heidelberg; 2011.

[47] Butcher J.C. Numerical Methods for Ordinary Differential Equations. 2^{nd} edition, John Wiley and Sons; 2008.

[48] Zidan M.A., Radwan A.G., Salama K.N. The Effect of Numerical Techniques on Differential Equation Based Chaotic Generators. International Conference on Microelectronics, ICM, Hammamet, Tunisia; 2011.

[49] De Micco L., Larrondo H.A. FPGA implementation of a chaotic oscillator using RK4 method. Programmable Logic (SPL), VII Southern Conference; 2011.

[50] Varsakelisa C., Anagnostidis P. On the susceptibility of numerical methods to computational chaos and superstability. Communications in Nonlinear Science and Numerical Simulation 2016;33:118-132.

[51] Chen X., Xiang F., Zhang L. The Dynamics of Runge-Kutta Methods. International Conference on Int. J. Bifurcation and Chaos 2012;2:427-449.

[52] Rossler O.E. An Equation for Continuous Chaos. Physics Letters 1976;54(5).

[53] Chua L.O. Archive for Elektronik and Uebertragungstechniko 1992;46(4):250-257.

[54] Chen G., Ueta T. Yet Another Chaotic Attractor. International Journal of Bifurcation and chaos; 1999.

[55] Linz S.J., Sprott J.C. Elementary chaotic flow. Physics Letters A 1999;259:240-245.

[56] Soares L., Elvio I., Dutra C., Junior S., Glesner M. Advantages of the Linz-Sprott Weak Nonlinearity on the FPGA Implementation of Chaotic Systems: a Comparative Analysis. Signals, Circuits and Systems, ISSCS. International Symposium; 2005.