Applied Mathematical Sciences, Vol. 7, 2013, no. 5, 237 - 246

Design and FPGA Implementation of Lorenz Chaotic System for Information Security Issues

Lahcene Merah[1], Adda Ali-Pacha[2], Naima Hadj Said[3] and Mustafa Mamat[4]

[1, 2] Department of Electronics
[3]Department of Computer Sciences
University of Science and Technology of Oran (USTO)
BP 1505 El M'Naouer Oran 31036, Algeria
a.alipacha@gmail.com
[4]Department of Mathematics, Universiti Malaysia Terengganu
21300 K.Terengganu, Malaysia

Abstract. Taking the Lorenz chaotic system as an example, FPGA (Field Programmable Gate Array) technology is applied to obtain chaotic sequence in this paper, the Xilinx system generator technology was used for the conception of Lorenz chaotic system and generating the VHDL code. This code is used again to configuring the FPGA. Experiment shows the output sequence of the designed system has a good result. The generated chaotic sequence may be used for information security in the modern communication field.

Keywords: FPGA, Lorenz chaotic attractor, chaos encryption

1. Introduction

In the recent years, the revolution of media exchange, the high debit and quantity of information through the open internet, satellites, mobiles and any kind of network, makes easy to access the content of this information. Protecting unauthorized information is the major problem for researchers; the usual encryption systems are unable to resist the evolution of hacker's attacks. Developing a robust cryptosystems is always an important requirement.

As many crypto systems based on the generation of pseudo random sequences for hiding clear messages, using chaotic systems as generator of those sequences becomes an important study subject for many years.
The chaos is obtained from nonlinear dynamic system; it corresponds to an aperiodic, chaotic, and sensitive to small changes to initial values and control parameters behavior [1].
There are two approaches when using chaotic dynamics in cryptography. The first one uses chaotic systems to generate pseudo-random sequences, which are then used as key streams to mask the plaintext in a manifold of ways. In the second approach, the plain text is used as initial state and the cipher text follows from the orbit being generated. The first approach corresponds to stream ciphers, while the second to block ciphers, both in secret and public key cryptography [2].

Theoretically, chaos based Pseudo-Random Number Generators (PRNG) is proofed with good randomness and infinite period as well, whilst the nonlinear characters significantly enhance the complexity of the PRNGs' structures. Additionally, the widely existing chaotic functions provide countless options that broadly increase the pseudo random number generating methods. Therefore, many Chaotic Pseudo Random Number Generators (CPRNG) have been proposed in the literature, but hardware realization, especially, the chip implementation is still a great challenge [3].

In this paper, one of efficient technologies for designing digital systems was used, we mean Xilinx ISE suite design environment that including "Xilinx system generator" (XSG) used to design and implement Lorenz system. That generate a complex VHDL code from a simple blocks design, this VHDL code used later to configure the target FPGA board. The example of chaotic system taken in this paper is the Lorenz system (Lorenz chaotic attractor).

2. Lorenz Equations System

The Lorenz system, named for Edward N. Lorenz, is an example of a non-linear dynamic system corresponding to the long-term behavior of the Lorenz system. The Lorenz system is a 3-dimensional dynamical system that exhibits chaotic flow. The graphical representation of such system shows how the state of a dynamical system (the three variables of a three-dimensional system) evolves over time in a complex, non-repeating pattern.

The equations that govern the Lorenz system are:

$$\begin{cases} \frac{dx}{dt} = \sigma * (y - x) \\ \frac{dy}{dt} = r * x - y - x * z \\ \frac{dz}{dt} = x * y - \beta * z \end{cases}$$

where σ, r and β called the control parameters. All σ, r, β > 0, but usually σ = 10, β = 8/3 and r is varied. The system exhibits chaotic behavior for r = 28. To resolving this system we can use the RK-4 method with a numerical tool like Matlab for example.

3. Implementation of Lorenz System Using Xilinx System Generator (XSG)

System Generator is a DSP design tool from Xilinx that enables the use of the Math Works model-based Simulink design environment for FPGA design. Previous experience with Xilinx FPGAs or RTL design methodologies is not required when using System Generator. Designs are captured in the DSP friendly Simulink modeling environment using a Xilinx specific block set. All of the downstream FPGA implementation steps including synthesis and place and route are automatically performed to generate an FPGA programming file [4].

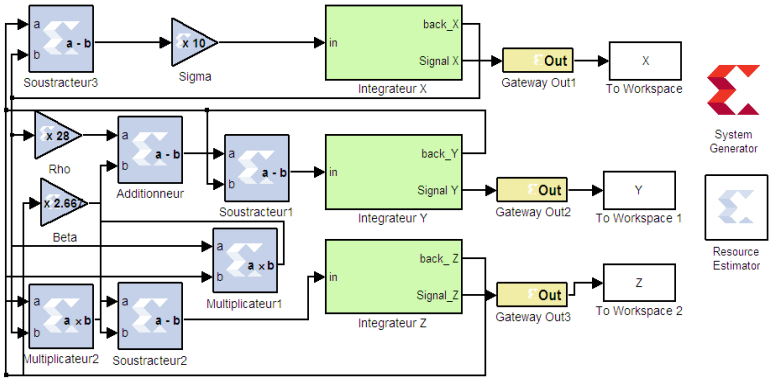
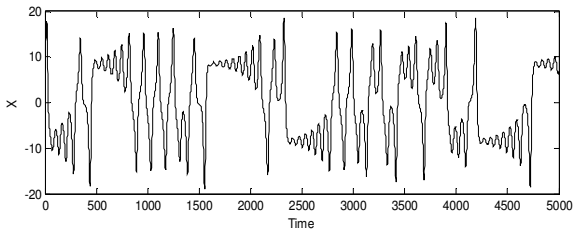


Figure .1 Lorenz system using Xilinx system generator.



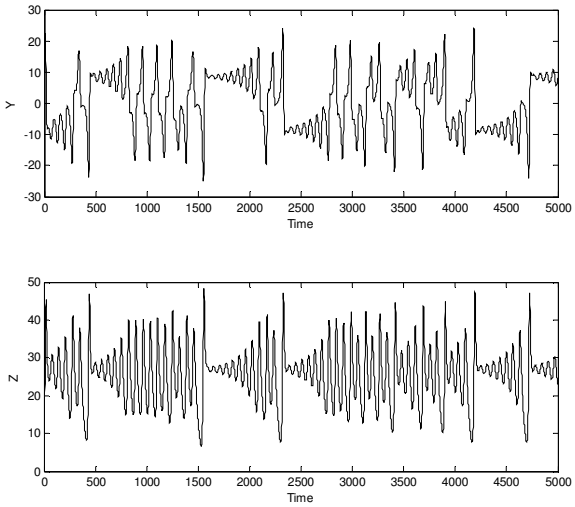


Figure. 2 *Signals x, y and z generated by the Lorenz system created using XSG.*

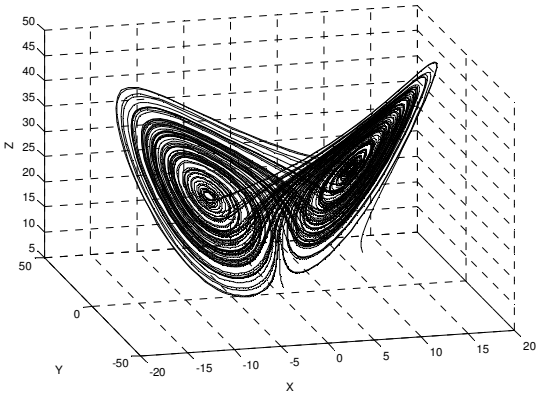


Figure. 3. *Three dimensional Lorenz system*

We adopt the implementation with a fixed point and with a representation of the real data on 32 bits (12Q20), 12 for the entire and 20 for the fraction.

The figures below are obtained by fixing the following parameters: $\sigma = 10$, $\beta = 8/3$ and $r = 28$. The initial conditions $x_0 = 10$, $y_0 = 10$ and $z_0 = 10$.

It seems clearly that the different signals x, y and z have a random behavior. It should be noted that these results are identical to those obtained using the method of RK-4 in a previous work.

4. FPGA Digital Implementation of Lorenz System

Once the design is completed, the hardware implementation VHDL code can be generated.

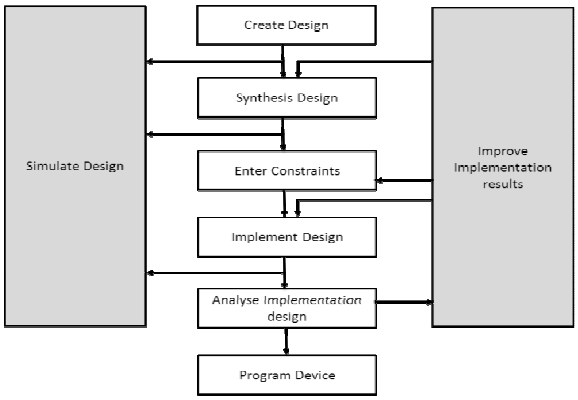


Figure .4 *ISE Design Flow Overview [5]*

We leave XSG tool, we using in the next step another tool; it is Xilinx ISE design suite that allows us to generate the FPGA's programming file after some steps as the following scheme show:

- *Design Creation*: already created in the last section.
- *Synthesis:* During synthesis, the synthesis engine compiles the design to transform VHDL sources into an architecture-specific design netlist.
- *Simulation:* At various points during the design flow, we verify the functionality of the design using a simulation tool. From within the ISE viewing environment, we use ISim, which is delivered with the ISE software or ModelSim simulators. Alternatively, we can simulate our design outside of ISE Project Navigator using any supported simulator.
- *Constraints Entry:* Using design constraints, we specify timing, placement, and other design requirements. The ISE software provides editors to facilitate constraints entry for timing constraints as well as I/O pin and layout constraints.
- *Implementation Analysis*: After implementation, we can analyze our design for performance against constraints, device resource utilization, timing performance, and power utilization. We can view results in static report files.

Implementation Improvement: Based on the analysis of our design results, we can make changes to design sources, process properties, or design constraints and then, rerun synthesis, implementation, or both to achieve design closure.

Device Configuration and Programming: After generating a programming file, we configure our device. During configuration, we generate configuration files and download the programming files from a host computer to a Xilinx® device. We used in our study the Xilinx FPGA SPARATAN 3E (xc3s1200e-4fg320) as shown in Figure 5.

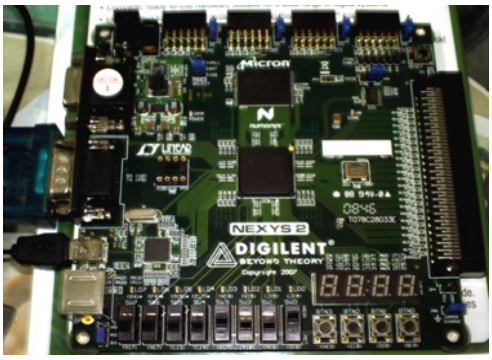


Figure. 5 *FPGA evaluation board used to Lorenz generato*

5. Simulation Using Isim

Before giving the results of the real FPGA implementation of our design, we give the results of the VHDL code simulation using Xilinx Isim simulator as shown in the following figure:

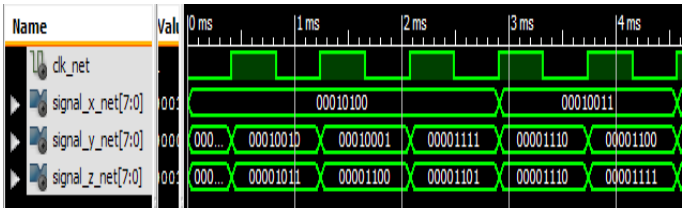


Figure.6 *Simulation results using Isim.*

The Fig.6 presents a portion of the waveform of each signal x, y and z. we took some samples for each signal and compared to those generated by Lorenz system created by Xilinx system generator, we found that the results are identical. It should be noted that the output for each signal was presented on 8 bits, the form of signals doesn't change as those presented on 32 bits.

6. Implementation Analysis

After the simulation step and ensuring the proper functioning of our VHDL code, we go to the next step that consists the generation of bit file to program our FPGA. Before going to this step we must demonstrate some things, in fact, our FPGA will

communicate with the outside world (like computer) via the RS232 port. The RS232 cable is a serial information support, in other words, the data pass bit after bit serially. So, we need to convert each parallel 8 bits data to a serial data via the RS232 cable bit after bit and retrieve the 8 bits data in the other side (receiver). This is the role of UART.

The UART (Universal Asynchronous Receiver Transmitter) is a component used to convert serial data to parallel data, and parallel data to serial data.

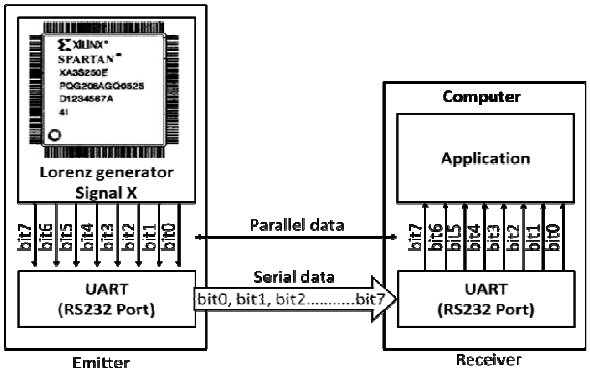


Figure.7 *Sending and receiving data via UART component*

The serial data transferred into the UART is placed on an output bus after the UART converts it into parallel information. This bus can then be used as input to other logic in the gate array. The resulting data can then be sent back out serially by using the UART component again [6]. A UART VHDL code was added to our design to ensure the communication between FPGA and computer via RS232 port. The UART transfer data rate was calibrated with about of 9600 bit /s; it mean that the time needed for sending 1 bit is 104 µs firstly. Secondly the total time for sending one data (8 bits) is 832 µs, we can conclude that the Lorenz generator needs 832 µs for give us one data, it should be noted that 832 µs not the real time that the FPGA work at, but the modified value in order to adapt with the RS-232 port. The minimum period that the FPGA works at is as shown on the table 1.

The total number of FPGA board resources needed to implement our project is the sum of resources needed for Lorenz system and UART component. The tables 1 and 2 present respectively, timing and mapping final report.

7. FPGA Configuration

The final step is generating a bit file (bitstream) to program the xc3s1200e-4fg320 FPGA, the Fig.8 and Fig.9 present the real time signals of x, z and the 2-D x-z of the Lorenz generator.

We can say that the real time signals generated by the FPGA are the same as those simulated with Xilinx system generator, and the UART component VHDL code works well.

TABLE.1 TIMING FINAL REPORT

Source clock	20 ns (50 MHz)
UART clock	104 µs (9.615 MHz)
Lorenz generator modified clock	832 µs (1.201 MHz)
Lorenz generator min clock and max freq working	55.46 ns (18.03 MHz)

TABLE.2 MAPING FINAL REPORT

Logic Utilization	Used	Available	
Total Number Slice Registers	144	17,344	1%
Number of 4 input LUTs	1,466	17,344	8%
Number of occupied Slices	1,029	8,672	11%
Total Number of 4 input LUTs	1,912	17,344	11%
Number of bonded IOBs	14	250	5%
Number of BUFGMUXs	2	24	8%
Number of MULT18X18SIOs	8	28	28%

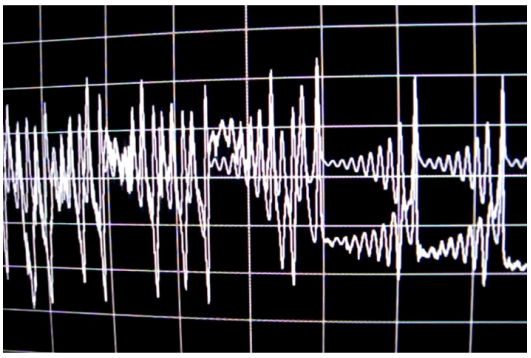


Figure. 8 *Real time x and z signals of Lorenz generator*

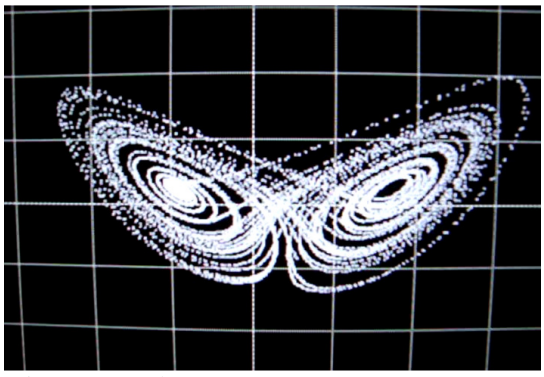


Figure .8 *Real time 2D x-z attractor of Lorenz.*

8. Conclusion

We have seen in this paper step by step the real time FPGA implementation of Lorenz system using Xilinx system generator and Xilinx SPARATAN 3E xc3s1200e-4fg320 FPGA, the results of the real implementation are identical to those obtained with Xilinx system generator tool simulation. Contrary of many papers, this work can be a good reference for anyone wants to implement digital designs on FPGA without previous knowledge of VHDL code, from a simple design blocks and after a few steps using some tools we can generate a complicated bit file for programming the FPGA board. The chaotic signals generated from the FPGA can be used later for many applications especially for information security; in fact the evaluation of Lorenz chaotic sequence randomness was out of scope of this paper.

References

[1] A. Ali-Pacha, N. Hadj-Said, A. M'Hamed, A. Belghoraf, «Lorenz's Attractor Applied to the Stream Cipher (Ali-Pacha Generator)», Chaos, Solitons & Fractals - 2007, Volume 33/5 pp.1762-1766.

[2] J.M. Amigó, J. Szczepanski and L. Kocarev, "Theory and Practice of Chaotic", Centro de Investigación Operativa, Universidad Miguel Hernández de Elche, Avda de la Universidad s/n, 03202 Elche (Alicante), March 2006.

[3] Yaobin Mao, Liu Cao, and Wenbo Liu. Design and FPGA Implementation of a Pseudo-Random Bit Sequence Generator Using Spatiotemporal Chaos. ICCCAS'06, June, 28-30, 2006.

[4] Xilinx, Inc, "System Generator for DSP, Getting Started Guide", UG639 (v 12.4) December 14, 2010.
[5] Xilinx, Inc, "Synthesis and Simulation Design Guide", UG626 (v 12.3) September 21, 2010.
[6] Digilent, Inc, "RS232 Reference Component", July 25, 2008.
[7] Yu-Huang Chu, Chang, S., Dynamical cryptography based on synchronised chaotic systems, Electronics Letters, 1999, vol. 35, Issue 12, 974-975.
[8] Q.V. Lawande B. R. Ivan and S. D. Dhodapkar, Chaos based cryptography: a new approach to secure communications, Barc Letters , N°258, July 2005, 1-11.
[9] B. Orue, G. Alvarez, M. Romera, G. Pastor, F. Montoya and Shujun Li, Lorenz System Parameter Determination and Application to Break the Security of Two-channel Chaotic Cryptosystems, arXiv:nlin/0606029v2 [nlin.CD] 5 Aug 2007
[10] J.M. Amigó, L. Kocarev, and J. Szczepanski, Theory and practice of chaotic cryptography , Physics Letters A Volume 366, Issue 3, 25 June 2007, 211-216.
[11] P.G. Vaidya and Savita Angadi, Decoding chaotic cryptography without access to the superkey, Chaos, Solitons & Fractals, Volume 17, Issues 2-3, July 2003, Pages 379-386.
[12] Einat Klein, Rachel Mislovaty, Ido Kanter, and Wolfgang Kinzel, Public-channel cryptography using chaos synchronization, Phys. Rev. E 72, 016214 (2005) [4 pages].

Received: September, 2012