International Journal of Engineering Research and Technology. ISSN 0974-3154 Vol.13, No.4 (2020), pp. 668-674
 https://dx.doi.org/10.37624/IJERT/13.4.2020.668-674

Cellular Automata Implemented on FPGA Based on Totalistic Rules for Deterministic Systems

Alexander Bautista-Torres[1], Edgar Alexander Bautista-Aleman[2] and Helbert Eduardo Espitia-Cuchango[3]

[1,2,3]Facultad de Ingeniería, Universidad Distrital Francisco José de Caldas, Bogotá, Colombia.

Abstract

This paper presents the design and implementation of a cellular automata based on totalistic rules for dynamic deterministic systems. The implementation is made on FPGA and the simulation results are shown via a software user interface. With this development, the parallelism of the FPGA is used for the simulation of dynamic systems by means of cellular automata. The results show that the proposed system obtains the simulation of the dynamic system using less time than conventional software of cellular automata.

Keywords: Cellular automata, dynamic system, FPGA, simulation.

I. INTRODUCTION

Commonly, phenomena and interactions among the different parts of a system (namely physical, biological, etc.) are described in models that in a global manner, explain the behavior of each part. Even though such approximations can be technically appropriate, those lack of viability in terms of implementation. In this regard, cellular automata arise as an alternative for the simulation in these models taking a group of cells at different stages as a start point and an upgrade rule aiming the prediction of behavior in time of a particular system.

Cellular Automata CA were initially proposed in the fifties by John von Neumann [1] and Konrad Zuse [2] and later developed in the eighties by the physicist Stephen Wolfram [3]. A CA is considered a dynamic system in discrete time in which the interactions among cells are governed by an upgrade or transition rule [4]. In itself, the concept of CA is associated with concepts like space and locality of influence [5], [6], and is widely employed in the simulation of dynamic systems of a different order (physics biology, sociology, and chemistry, among others) [4], [7], [8].

Currently, there are software applications that allow executing cellular automata of diverse kinds (NETLOGO, SPASIM, STARLOGO, SR-CA [9]). Such applications are useful when the number of evolutions and/or the number of cellular stages is small; nevertheless, when some of these parameters arise, the execution time of the CA gradually increases depending on the software type of sequential processing [4], which tends to be inefficient when having considerable increments [10].

There are several implementations of CA built on FPGA (Field Programmable Gate Array) [10-18]. Some of them pose models to evaluate and improve the performance while others employ code generators based on a high-level language to get VHDL/VERILOG code for a particular CA.

These implementations result to be more efficient when executed on parallel processing platforms than when using software applications [10], especially when the number of evolutions and/or stages of the CA arise; however, those lack versatility for the absence of a suitable graphic interface to visualize results and change the parameters of the simulation of the CA, moreover, it is necessary that the user interact with tools of FPGA/C++ [13].

Consequently, this document proposes the design of hardware architecture of a CA for a maximum size of 10.000 cells, which is executed in a time considerably less extended than when using software applications and dynamically displaying the results using software designed with Java. This architecture is applied to CA deterministic based on totalistic Boolean rules and cells of four stages aiming the most optimal usage of the parallelism of the FPGA for the simulation of dynamic systems through cellular automata.

II. METHODOLOGY

This section describes the design and implementation of the system for a simulation of cellular automata on FPGA, including the revision of the concepts of cellular automata, the tool for developing the FPGA, and lastly, the design and implementation of the cellular automaton is displayed in detail.

II.I Cellular Automata

A cellular automaton refers to a discrete dynamic system formed by a set of cells distributed in a grid which have a finite set of stages that change according to an upgrade rule applied in steps of discrete time [19]. CAs are uniform since the cells are upgraded from the same set of rules, parallel since all the cells evolve simultaneously, and local since the further stage of each cell depends on its current stage and the stage of the neighbors [4]. Fig. 1 shows the graphic representation of a single cellular automaton.

International Journal of Engineering Research and Technology. ISSN 0974-3154 Vol.13, No.4 (2020), pp. 668-674
 https://dx.doi.org/10.37624/IJERT/13.4.2020.668-674

The CA execution algorithm consists of three steps. First, the assignment of initial stages to the cells, second, the execution of the transition rule on each one, and third, the cells upgrade according to the result obtained in the second step [19].

This process must be executed concurrently on the cells [10]. Every time that these three steps are made represents the accomplishment an evolution process.

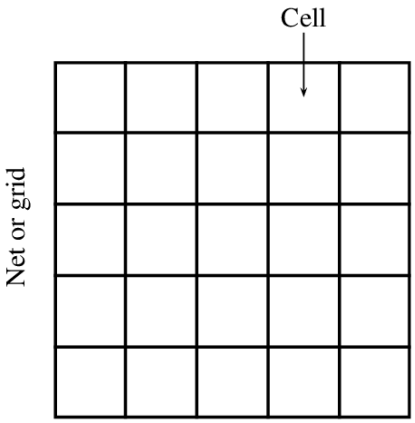

Fig. 1. Grid of cells or cells of a cellular automaton.

CAs can be classified according to their rules in deterministic and probabilistic, or according to the spatial distribution in one-dimensional, two-dimensional, or three-dimensional; thus, the main components of a CA are grid, cells [3], transition rules [4], neighborhood [20], and border conditions [21]. For practical issues, this document considers two-dimension, deterministic CAs, and border conditions fixed or recurring.

II.II Development Tool

While implementing the CA the parallelism is aimed during the execution, therefore, the Micro Blaze Development Kit Spartan-3E1600E card was employed which is a robust development platform that allows multiple projects of a different kind. The board has an FPGA Spartan3E with 1600K gates. It also includes a Flash Xilinx platform, USB interfaces and parallels for programing JTAG with multiple configuration options for the FPGA using an Intel Strata Flash and a Serial Flash ST [22].

II.III Cellular Automaton Design

Two principles are necessary for the design of the hardware architecture: first, the number of cells, which is required to be large enough to have a wider concept of the dynamic of the behavior, and second, the time interval between two consecutive evolutions (time of evolution) that needs to be as small as possible to accelerate the execution (especially when the number of evolutions is high).

From the perspective of the hardware employed in the proposed architecture, each cell of the CA needs to have an arithmetic-logic unit for solving the rule of transition stages which would demand a large number of hardware resources, making almost impossible implementing a CA of 100X100 cells in an FPGA Spartan-3E1600E.

Thus, for solving this issue a semi-parallel processing architecture is proposed which executes blocks of 100 cells in a recurring way.

II.IV System General Architecture

The system consists of a software and hardware module interacting exchanging information on the process of execution of the CA. Fig. 2 presents a system general diagram of blocks.

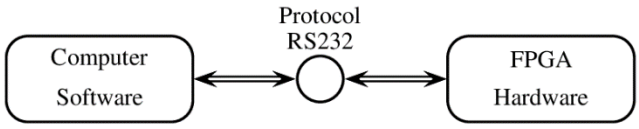

Fig. 2. Cellular automaton block composition.

II.V Software Module

This includes an application developed using Java, this module communicates with the FPGA using RS232 to transmit and receive information of the CA. The graphic interface permits the user simulate and configure the parameters in the CA (number of evolutions, type of neighborhood whether Von Neumman or Moore and 100X100 maximum size); likewise, it allows visualizing the results after a set of evolutions programed.

II.VI Hardware Module

This module includes an FPGA SPARTAN3E1600 and contains algorithms programmed in VHDL for the execution of the CA and communication processes with the software. Internally, the module is composed by the block of communications, which is established by a half-duplex channel between the FPGA and the software module through the RS232, and ASCII character commands; next is the processing information block that codes and decodes the signals interchanging with the software module and the cellular automaton block in charge of the execution. Fig. 3 shows the way of operation of the blocks.

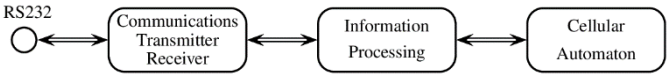

Fig. 3. Composition of the hardware module.

International Journal of Engineering Research and Technology. ISSN 0974-3154 Vol.13, No.4 (2020), pp. 668-674
 https://dx.doi.org/10.37624/IJERT/13.4.2020.668-674

II.VII Cellular Automaton Architecture

The architecture here presented is designed for executing four CA based on deterministic models with totalistic rules of maximum four stages. Fig. 4 presents a diagram of the internal distribution of this block.

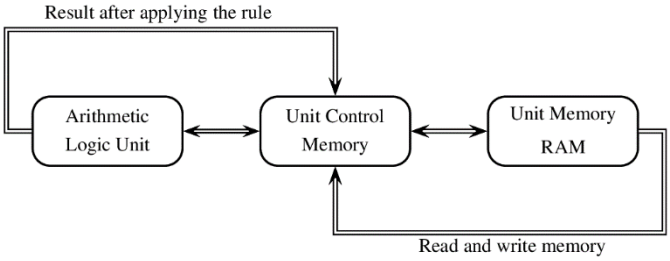

Fig. 4. Composition of the cellular automaton block.

Fig. 4 shows the RAM Unit Memory (RAM-UM) which storages the cells current and future stages in two volatile memory spaces, the Unit of Control of Memory (UCM) is also seen, this is in charge of the reading/writing processes, and the Arithmetic Logic Unit (ALU) corresponding to a one-dimensional of a hundred (100) cells based on logical-mathematical algorithms that concurrently execute one of the four CA.

II.VIII Execution Process

The evolutive process of a CA is made of three steps: Reading of the current stage in each and the surrounding cells, calculations of the future stage of each cell through the CA model, and data update of the different stages of each cell. These processes are sequential and are executed concurrently on a one-dimensional vector. The maximum length of the vector is 100 cells; however, it varies according to the CA executed. This process is repeated sequentially as many times as the size of the CA indicates. In cases where the CA is 50X80, the sector may have 50 cells and the three steps would be repeated 80 times. Fig. 5 shows an example of this situation.

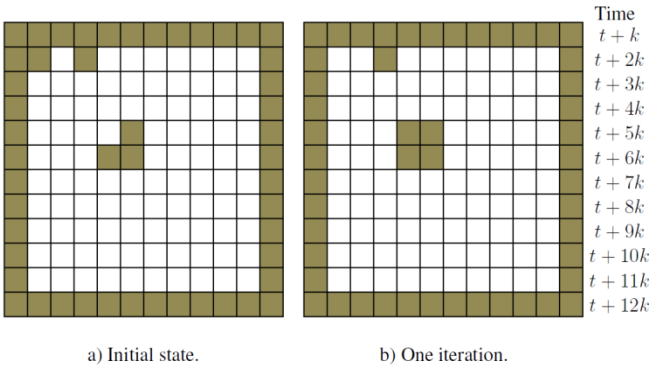

Fig. 5. CA execution process by sectors.

In Fig. 5, the example displays a CA of 10X10 whose upgrade rule is the game of life, green cells are neighbors of the CA extremes and define the border conditions; Fig. 5-a presents the CA with zero evolutions in a time t, while Fig. 5-b exposes the same CA with one evolution. The step from zero to one evolution happens in a time equals to $t + 12k$, k is the product of the lapse of the clock signal multiplied by the number of clock cycles used when performing the three processes.

Likewise, the time taken by a CA for a single step is given by:

$$T_{Evolution} = t_0 + (N + 2) \cdot T_{Clock} \cdot N_{Cycles} \qquad (1)$$

Here, t_0 is the instant where the automaton starts another evolution, t_0 which means the lapse of the system clock which comes of the Central Processing Unit (CPU) of the FPGA, and N_{Cycles} is the number of cycles employed in the execution of read-write processes and cells upgrade. Such implementation takes place with a 100 MHz clock. To manage the two memory spaces during the CA process is necessary to consider that each evolution uses one memory space to read the current stages and the other to write the future stages; thus, it is proposed that the memory space limited by 0X00 and 0X65 (0 and 101) is employed to read the current stages when the number of evolutions is even, and to write the future stages when the number of evolutions is odd. On the other hand, the memory space two, limited by 0X66 and 0XCB is used in reading mode when the number of evolutions is odd and in writing mode when even. When the first evolution takes place, initial stages of the CA must be written in the memory space 1; Table 1 shows the ranges of different directions and the possibilities.

Table 1. Memory space configurations.

Range of Directions	Even Evolutions	Odd Evolutions	Memory Space
0X00-0X65	Read	Write	One
0X66-0XCB	Write	Read	Two

According to this, there must be the same number of consecutive evolutions and iterations in accordance with the number of sectors in the grid. Below the review of the steps taken in each iteration:

Step one: Read and store the stages of the cells of a sector which is the same number of iterations, that is, iteration one is chosen from sector one, iteration two is chosen from sector two, and so on.

Step two: Next, read and store the stages of the cells hosted in the previous sector and next to the chosen sector. This is done to obtain eight neighbors for each cell of the selected sector in step 1. Reading operations made in step 1 and 2 are made in the reading sector of the memory space.

Step three: After obtaining the stages of the cells located in the selected sector with those of the neighbors, calculations and storage can be made on the future stage of the cells using the general transition rule.

Step four: Once obtained the future stage of the cells, such stages are written in the same memory space predetermined for writing.

According to this, each iteration needs to generate four different directions that vary according to parameters of the number of evolutions and the number of the sector selected in each iteration; where N represents the evolution stage while S is the number of the sector selected in each iteration, thus, four directions may be generalized as:

$$\begin{aligned} ADD_{Secr}(N,S) &= \begin{cases} S, & N = 2n-1 \\ S+102, & N = 2n \end{cases} \\ ADD_{vec1}(N,S) &= \begin{cases} S-1, & N = 2n-1 \\ (S-1)+102, & N = 2n \end{cases} \\ ADD_{vec2}(N,S) &= \begin{cases} S+1, & N = 2n-1 \\ (S+1)+102, & N = 2n \end{cases} \\ ADD_{Secw}(N,S) &= \begin{cases} S+102, & N = 2n-1 \\ S, & N = 2n \end{cases} \end{aligned} \quad (2)$$

Where $n \in Z^+$ and ADD_{Secr} is the direction of the sector memory. Meanwhile, ADD_{vec1} and ADD_{vec2} are the directions of memory from where the neighbors of the cells are obtained and ADD_{Secw} is the direction of the sector where the future stages of the cells of the sector selected are written. Table 2 displays value ranges that can take the aforementioned directions whether the number of evolutions is even or odd for the CA implemented.

Table 2. Range of directions in each iteration.

Direction	Range of directions for even evolutions	Range of directions for odd evolutions
ADDsecr	0X67 - 0XCA 103 - 202	0X01 - 0X64 1 - 100
AADvec1	0X66 - 0XC9 102 - 201	0X00 - 0X63 0 - 99
ADDvec2	0X68 - 0XD5 104 - 203	0X02 - 0X65 2 - 101
ADDsecw	0X01 - 0X64 1 - 100	0X67 - 0XCA 103 - 202

III. RESULTS

Firstly, the comparison is made using conventional software to simulate cellular automata. Later, as an example, the results delivered for one of the models implemented are qualitatively shown using graphics and a table containing the associated characteristics with the respective simulation.

The execution of the experiments considered different automata models like:

- Model 1: Lotka-Volterra (predator-prey).
- Model 2: Greenberg and Hastings.
- Model 3: Population growth with limited resources.

III.I Execution Time

Software SR-CA is employed aiming to have a comparison for the time employed in the execution, this software is widely used for cellular automaton [9]. This applet is developed using Java since this permits a set of predetermined rules, including the virtual life game; this software includes nine predetermined square grids with 16X16 as minimum and 4096X4096 as the maximum size.

Using it as a pattern of comparison includes the average measurement of the time employed by the software to complete an evolution; for this purpose, a fixed lapse of 10 seconds is considered then counting the numbers of evolutions achieved during that lapse. Table 3 shows different measurements with this tool.

Table 3. Time evaluation of the software SR-CA.

Time	10 sec	10 sec	10 sec	10 sec	10 sec
Evolutions	7526	7552	7742	7292	7451
Time	10 sec	10 sec	10 sec	10 sec	10 sec
Evolutions	7417	7380	7574	7517	7403

According to the experimental data in Table 3, the average number of evolutions is 7485,4; therefore, the average lapse of one evolution is 1,33 milliseconds.

Meanwhile, Table 4 shows the times employed for an evolution in the system developed for the models considered.

Table 4. Times of one evolution of the models in microseconds.

	Execution 1	Execution 2	Execution 3	Execution 4
Model 1	13,194	15,057	13,624	13,243
Model 2	17,597	11,047	10,959	10,595
Model 3	17,597	11,047	10,959	11,047

From Table 4, considering that the average time for execution is 12,997 microseconds, and the time taken by the software is 1,336 milliseconds, then, it is observable the advantage provided by an architecture dedicated only for performing simulations on cellular automata.

III.II Greenberg and Hastings Simulation Model

Greenberg and Hasting developed a model with cellular automata simulation the reaction of Belousov-Zhabotinsky, the model employs four neighbors [23], the possible stages are $\{0, 1, 2\}$ where:

- 0 is the rest stage or global equilibrium.
- 1 or 2 is the active stage.
- 3 represent the refractory stage.

The last one is characteristics since once achieved the finite automata will be insensitive to neighbors and unable to activate them. The transition rule was established by

International Journal of Engineering Research and Technology. ISSN 0974-3154 Vol.13, No.4 (2020), pp. 668-674
 https://dx.doi.org/10.37624/IJERT/13.4.2020.668-674

Greenberg and Hasting in the original model [23] according to the expression:

$$c_{i,j}(t+1) = R\left[c_{i,j}(t)\right] + D\left[c_{i-1,j}(t), c_{i+1,j}(t), c_{i,j-1}(t), c_{i,j+1}(t)\right] \quad (3)$$

Where,

$$R = \begin{cases} 2, & if\ c_{i,j}(t) = 1 \\ 0, & if\ otherwise \end{cases} \quad (4)$$

$$D = \begin{cases} k, & if\ c_{i,j}(t) = 0 \\ 0, & if\ otherwise \end{cases} \quad (5)$$

In equation (5) $k = 1$ if there is at least one active neighbor cell in the Von Neumman neighborhood, otherwise, $k = 0$. It is noteworthy that *R* refers to the process of reaction and *D* to the process of diffusion of the equation reaction-diffusion of Greenberg and Hastings.

Multiple stages of refractory and excitement can be present when employing Greenberg and Hastings model. The simulations made of this model are performed using a resting 0 stage, two stages of excitement 1, 2, and a refractory stage 3.

Fig. 6 displays the configuration of initial stages where the blue cells represent the resting stage, magenta cells the stage of excitement, and the green cells represent the refractory stage. It is noteworthy that the neighborhood of Von Neumman and constant border conditions were used for the simulation.

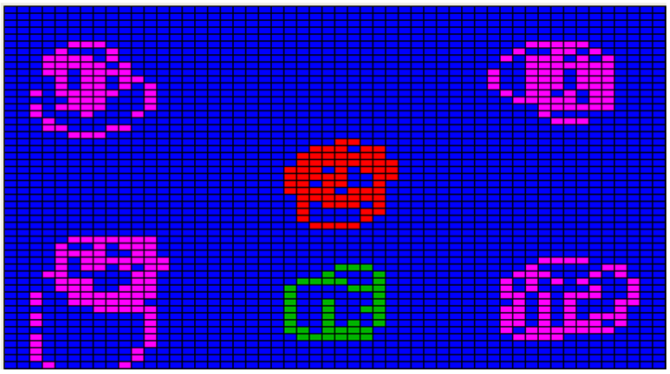

Fig. 6. Configuration of initial stages.

The results obtained after subduing the initial pattern to ten evolutions are shown in Fig. 7 where it is observed how the waves are generated in the center of the grid and move toward the exterior part of it in a way that allows clear identification of the processes of reaction-diffusion of the model Greenberg and Hastings. Note that the pattern of spatial distribution shows certain homogeneity in the central part which is shown in the wave shape.

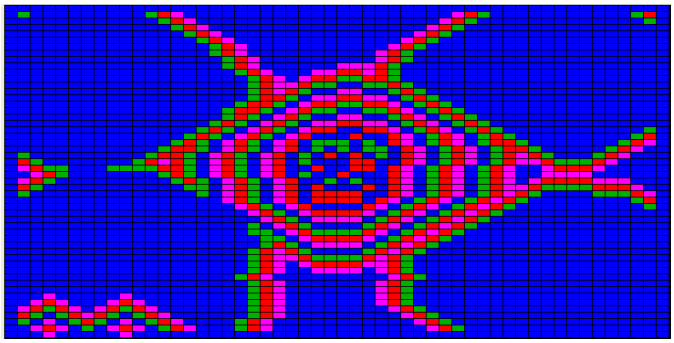

Fig. 7. Spatial distribution after 10 evolutions.

After 63 evolutions, Fig. 9 shows how the waves are generated from the center of the grid and move toward the external part. Here, the diffusion characteristic in the Greenberg and Hastings model is more notorious displaying also homogeneity in the wave generated from the central pattern of the grid. Note how the changes mold the expanded wave starting from the cells in the excitement stage. There is also a great density of cells in the refractory stage that pose a prompt stabilization of the expanded wave.

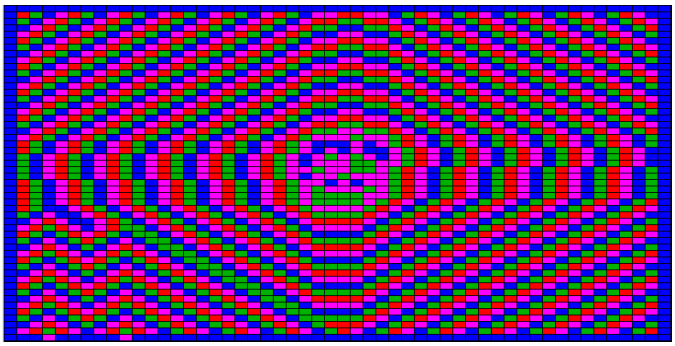

Fig. 8. Spatial distribution after 63 evolutions.

Fig. 8 shows that the waves are produced from the center of the grid and move toward the external part after 100 evolutions, no refractory-stage cells are observed in the central part and the homogeneity is preserved starting from the central pattern of the grid.

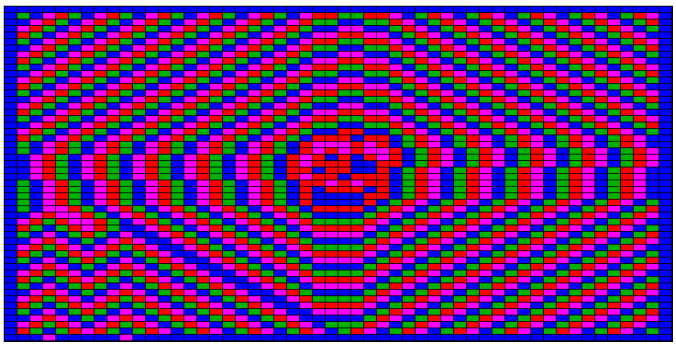

Fig. 9. Spatial distribution after 100 evolutions.

For 100 evolutions, Fig. 10 displays an identical pattern respect from Fig. 9, which indicates that the wav arrived to a

stable state; therefore, both processes of reaction and diffusion remain constant.

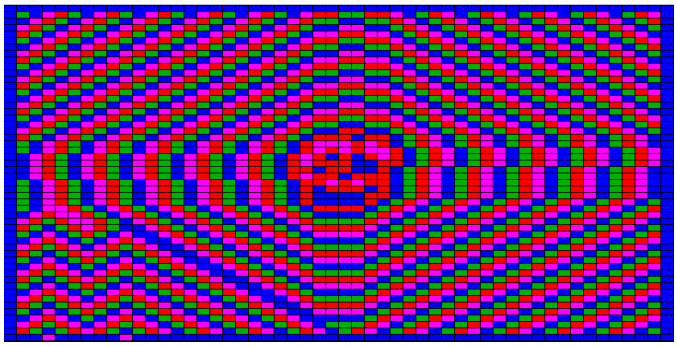

Fig. 10. Spatial distribution after 1000 evolutions.

Table 5 shows the results of the simulations previously mentioned for the Greenberg and Hastings model, connecting the number of cells in the resting stage to each evolution, active excited, and refractory. Also, considering the results of the same table and the times calculated in Table 4 (using a software for the simulation of cellular automata), it is noteworthy the benefit obtained from having a distributed system FPGA for the automaton simulation.

These results also allow observing that the systems achieve the respective evolutions of the automaton.

Table 5. Data of the model of Greenberg and Hastings.

Pattern	# Evol	Repose	Active	Excited	Refractory	T-1 Evol (µsec)
Pattern # 1 50x50	0	2119	270	42	69	0
	10	1973	213	278	250	17,597
	63	812	650	596	646	11,047
	100	848	612	648	596	10,959
	1000	848	612	648	596	10,595

VI. CONCLUSIONS

This work permits to observe the parallel nature of CAs. These are useful tools to simulate the behavior of some dynamic systems subjected to certain initial conditions.

The execution of the cellular automaton in an absolutely parallel architecture demands a great number of hardware resources since for each cell an arithmetic-logic unit is necessary, among other requirements for executing the proper operations.

The tools for the development provided by Xilinx in its environment are useful for partial implementation of the CA since on numerous occasions the FPGA counts with DDR-SDRAM (Double Data Rate Synchronous Dynamic Random Access Memory) and SD (Secure Digital) memories, among others that require suitable management.

The convergence of the software application and hardware development allowed the creation of a tool with strengths in both sides, having a user interface to interact with the tool and hardware platform that optimizes the execution time of the cellular automaton.

An improvement of the system may well modify the standard of communication to optimize the connection between the modules of software and hardware as there exists a more universal standard than RS-232 such as USB and TCP/IP.

Acknowledgements

The authors express their gratitude to the Facultad de Ingeniería de la Universidad Distrital Francisco José de Caldas.

REFERENCES

[1] J. von Neumann, Theory of self-reproducing automata, (University of Illinois Press, Urbana, 1966).

[2] K. Zuse, Rechnender raum, (Vieweg+teubner Verlag, 1969).

[3] S. Wolfram, Statistical mechanics of cellular automata, Reviews of Modern Physics, 55(3), 1983, 601-644.

[4] J. Schiff, Cellular Automata: A discrete view of the world, (Wiley, 2007).

[5] R. Lahoz, Bioinformática: simulación, vida artificial e inteligencia artificial, (Ediciones Díaz de Santos, 2004).

[6] J. Gómez, Comportamiento no-trivial en autómatas celulares, Tesis, Centro de Investigación y de Estudios Avanzados del I.P.N. México, 2000.

[7] C. Guan, P. G. Rowe, Should big cities grow? Scenario-based cellular automata urban growth modeling and policy applications, Journal of Urban Management, 5(2), 2016, 65-78.

[8] M. Guidolin, A. Chen, B. Ghimire, E. Keedwell, S. Djordjević, D. Savić, A weighted cellular automata 2D inundation model for rapid flood analysis, Environmental Modelling & Software, 84, 2016, 378-394.

[9] T. Tyler, Cellular Automata Links, [Online accessed 2020], Available: http://cell-auto.com/links/

[10] M. Halbach, R. Hoffmann, Implementing cellular Automata in FPGA Logic, in 18th International Parallel and Distributed Processing Symposium, Santa Fe, USA, 2004.

[11] I. Dogaru, R. Dogaru, A comparative study of several 2D cellular automata implementations in FPGA, in International Symposium on Fundamentals of Electrical Engineering (ISFEE), Bucharest, Romania, 2014, 1-4.

[12] W. Huang, General purpose cellular automata programming, Master of Science thesis, Department of Computer Science, Iowa State University, 2002.

[13] T. Kobori, T. Maruyama, T. Hoshino, A cellular automata system with FPGA, in 9th Annual IEEE Symposium on Field-Programmable Custom Computing Machines, Rohnert Park, USA, 2001, 120-129.

[14] J.L. Weston, P. Le, FPGA implementation of cellular automata spaces using a CAM based cellular architecture, in IEEENASA/ESA Conference on Adaptive Hardware and Systems, Noordwijk, Netherlands, 2008, 315-322.

[15] S. Murtaza, A.G. Hoekstra, P.M.A. Sloot, Performance modeling of 2D cellular automata on FPGA, in IEEE International Conference on Field Programmable Logic and Applications, Amsterdam, Netherlands, 2007, 74-78.

[16] W. Heenes, R. Hoffmann, S. Kanthak, FPGA implementations of the massively parallel GCA model, in 19th IEEE International Parallel and Distributed Processing Symposium (IPDPS-05), Denver, USA, 2005.

[17] Y. Liu, H. Yao, J. Wang, S. Fu, The implementation of oxidation process of the VLSI fabrication based on cellular automata in a FPGA, in International Conference on Computer Application and System Modeling (ICCASM), Taiyuan, China, 2010.

[18] P. Corsonello, G. Spezzano, G. Staino, D. Talia, Efficient implementation of cellular algorithms on reconfigurable hardware, in 10th Euromicro Workshop on Parallel, Distributed and Network-based Processing (EUROMICRO-PDP.02), Canary Islands, Spain, 2002, 211-218.

[19] E. Bilotta, P. Pantano, Cellular automata and complex systems: methods for modeling biological phenomena, (Medical Information Science Reference, First edition, 2010).

[20] E.R. Berlekamp, J.H. Conway, R.K. Guy, Winning ways for your mathematical plays, (Academic Press, 1982).

[21] B. Chopard, M. Droz, Cellular automata modeling of physical systems, (Cambridge University Press, 1998).

[22] Xilinx Inc, MicroBlaze Development Kit Spartan-3E1600E Edition User Guide, UG257 (v1.1) December 5, 2007.

[23] J.M. Greenberg, B.D. Hassard, S.P. Hastings, Pattern formation and periodic structures in systems modeled by reaction-diffusion equations, Bulletin of the American Mathematical Society, 84(6), 1978, 1296-1327.