Automatic Design of FPGA Processor for the Backtracking of DNA Sequences Evolution Using Cellular Automata and Genetic Algorithms

Georgios Ch. Sirakoulis

Democritus University of Thrace, Department of Electrical and Computer Engineering,
Laboratory of Electronics,
GR 67100 Xanthi, Greece
gsirak@ee.duth.gr

Abstract. In several cases, the DNA sequences of an organism are available in different stages of its evolution and it is desirable to reconstruct the DNA sequence in a previous evolution stage for which the exact sequence is not known. A CAD tool for backtracking the DNA sequence evolution based on Cellular Automata (CA) and Genetic Algorithms (GAs) was developed. Furthermore, the proposed system is able of automatic production of synthesizable VHDL code corresponding to the CA model. More specifically, DNA is modeled as a one-dimensional CA with four states per cell, i.e. the four DNA bases A, C, T and G. Linear evolution rules, represented by square matrices, are considered. The evolution rule can be determined using the global state of the DNA sequence in various evolution steps. This determination is accomplished using GAs. Moreover, because of the final produced CA's binary states and its local rule simplicity, the hardware implementation of the proposed model is straightforward. Finally, the FPGA processor that executes the CA model was fully designed, placed and routed.

1 Introduction

Bioinformatics research has proven to be very successful. Thanks to the development of advanced biochemical and biophysical instrumentation methods, we are able to collect valuable information about genome and proteome sequences, and structures of biological macromolecules [1]. The collected data, however, are often noisy and ambiguous, and thus the need for better techniques to solve complex problems connected with proper interpretation and plausible reconstruction (in terms of models) of the obtained biochemical information. It requires more accurate and faster database and data processing technologies, and better computational intelligence algorithms. Biochemical and biophysical laboratories collect data only about constituent elements that must be combined, analysed, and processed in order to obtain valid bioinformatics models [2]. Fortunately, several of the existing computational intelligence techniques can be adopted for solving bioinformatics problems, and new methods are being developed almost daily. In general, we can use computational intelligence methods providing that they are wisely combined with bioinformatics, as illustrated by the authors in [2].

Following the aforementioned outline, we figured out that DNA can be modelled as a one-dimensional Cellular Automaton (CA) [3]. In this model the phosphate chain corresponds to the CA lattice and the deoxyribose sugars to the CA cells. At each sugar molecule one of the four bases A (Adenine), C (Cytosine), T (Thymine) and G (Guanine) may bind. These four bases correspond to the four possible states of the CA cell. CAs appeared to be a promising model for DNA [3], because the DNA structure, function and evolution can be simulated using several mathematical tools (such as linear algebra and operators), introduced through the use of CAs. Following that line we developed a simulator, named *CAs for DNA*, for the study of DNA sequences with the help of CAs. *CAs for DNA* is an interactive simulation tool that includes a Graphical User Interface [GUI] which has been implemented using Matlab facilities. Moreover, in elementary CAs, the CA evolution rule can be extracted from a given number of CA evolution patterns. This method can also be applied to the CAs that model DNA. As a result, the developed simulator is able for modelling DNA evolution by extracting CA rules using Genetic Algorithms (GAs) [4]. The evolution rule can be determined by providing the global state of the DNA sequence in various evolution steps. Then, since the rule of evolution and the sequences of DNA are known for several evolution steps, it may be possible to determine the DNA sequence in previous evolution steps.

Finally, in order to speed up the application of CA to the study of DNA sequences the proposed tool is capable of producing Very High Speed Integrated Circuit (VHSIC) Hardware Description Language (VHDL) synthesizable code for the hardware implementation of the CA rules that model DNA. More specifically, *CAs for DNA* using a translation algorithm, that checks the CA parameters values previously determined by the user with the help of GA, automatically produces the synthesizable VHDL code that describes the CA algorithm. It should be mentioned that CAs are one of the computational structures best suited for hardware realization. The CA architecture offers a number of advantages and beneficial features such as simplicity, regularity, ease of mask generation, silicon-area utilization, and locality of interconnections. As a result, the automatically produced VHDL code can be fed into a commercial Field Programmable Gate Arrays (FPGA) CAD system, and the layout of the dedicated hardware that executes the CA algorithm can be designed to any FPGA Programmer. In this paper, the design processing of the finally produced VHDL code, i.e. analysis, elaboration and simulation, has been checked out with the help of the Quartus II, v. 7.2® design software of the ALTERA® Corporation. Test benches were automatically constructed by our system, for the simulation needs of the VHDL code, and the Simulator of Quartus® was used to simulate the operation of the dedicated processor described by the VHDL code obtained. Consequently, the implementation of the resulting VHDL code results in a FPGA, which is able to perform some real experiments, and to serve as a powerful "virtual lab" dedicated to the modelling of the backtracking of DNA sequences evolution.

2 Modeling DNA in Terms of Cellular Automata

In this session we suggest strategies for modelling DNA in terms of CAs by including a simultaneous translation of DNA properties into CAs. In the presented model the phosphate chain corresponds to the CA lattice and the deoxyribose sugars to the CA cells. At each sugar molecule one of the four bases A, C, T and G may bind. These four bases correspond to the four possible states of the CA cell. In non-sexual reproduction, the DNA molecule is passed from an individual to its offspring, whereas in sexual reproduction, the DNA of the offspring consists of parts of the parental DNA. We define as an *evolution event* a change in state, which may occur in one or more CA cells. Therefore, mutation is an evolution event and it corresponds to cell state changes. The time step in CA evolution is the time interval between two CA cell changes and, therefore, the time flow is not uniform. A result of modeling DNA as a CA is that the DNA strand and the individuals passing it from one generation to the other may exist in different time scales and, therefore, the DNA evolution is time-like separated from the life of the individuals that carry it.

The main question that rises when one tries to model DNA is whether mutations are completely random or not. If mutations are completely random, then CAs, which are deterministic computational models, can not model DNA evolution. In this case probabilistic methods, such as Markov chains may be appropriate. Although the answer to this question is not known, there are some indications that mutations and, therefore, DNA evolution may not be completely random [5, 6].

We will proceed to the model construction by assuming that mutations, i.e. CA cell changes are not completely random, but depend on the states of some of the cells that are located near by. Neighbor-dependent mutation has been studied using Markov chains and revealed biases in mutation rates that depend on the neighboring bases. Suppose that a state change at the *i*th cell occurs, and a time step is taken. In the model presented here it is supposed that the state of this cell has changed as a result of the effect of the states of its neighbors. The new state of the *i*th cell at this time step (which is generally the *t+1* step) is given by:

$$C_i^{t+1} = \hat{M}\left(C_{i-r}^t, \ldots, C_{i-3}^t, C_{i-2}^t C_{i-1}^t, C_i^t, C_{i+1}^t, C_{i+2}^t, C_{i+3}^t, \ldots, C_{i+r}^t\right) \tag{1}$$

where operator, $\hat{M}$, may be a mathematical function, a logic function, a matrix etc. In the case of linear evolution rules the operator $\hat{M}$ of equation (1) is a matrix, *M* [3]. While, a vast number of evolution rules can be applied to the CA that models DNA, the study of linear rules reveals the dynamics of the CA evolution and provides a very good insight to the structures created by evolution. The use of linear rules is further justified by the fact that a linear algebra has already been successfully used to the analysis of mutation rates. Most of the studies on mathematical models of DNA are limited to nearest neighbor interaction. Because of that, we have chosen to use in our simulations an evolution rule that incorporates only nearest neighbor interaction. As a result, all the elements in a matrix row of *M* are zero, except the three neighboring elements that are equal to one. In equation (1) cell states are one of the four bases A, C, T and G, which are represented by numbers of the quaternary number system, which contains only four numbers, i.e. *0, 1, 2* and *3*. We represent the bases with numbers as follows: $A \rightarrow 0$, $C \rightarrow 1$, $T \rightarrow 2$, and $G \rightarrow 3$. In elementary CAs, given an evolution pattern the evolution rule that generated it can be determined. It is very probable for such a method to exist for CAs that model DNA evolution. In this case if the evolution of the DNA sequence at various time steps is given, it will be possible to determine the evolution rule (or rules) that generated this evolution. After that, since the evolution rule and the DNA sequence at present time are known, it may be possible to predict the next evolution event (or events) and, therefore, the DNA sequence at the next time step.

3 *CAs for DNA* CAD Tool

CAs for DNA is an automated simulation and hardware implementation tool for extracting with high success the CA evolution rule, or rules that model the evolution of DNA sequence with the usage of GAs. A vast number of evolution rules can be applied to the presented CA that models DNA evolution. The CA rule space comprises all the possible local rules that may be applied to the CA cells. For CAs with only two states per cell, the number of all possible rules is given by 2^{2^n}, where n is the number of cells in the neighbourhood. In one-dimensional CAs with only two states per cell, the neighbourhood of which comprises the left, the right and the same cell, the number of all possible rules is 2^8, while in the four-state CA these rules are extended to 4^{4^3} or 4^{64}. The whole rule space of such CAs must be searched in order to find the possible CA evolution rules that model the DNA sequence evolution. In this work GAs are used to search the CA rule space. A possible evolution scheme of the proposed CA is shown in Fig. 1, where the first row gives all the possible states the cells within the neighbourhood could take. The r_i's in the second row are the rule components which take values from the discrete set {0, 1, 2, 3}. The last row shows the coefficients associated with the corresponding components. The rule R can be defined as $R=(r_0, r_1, r_2, r_3, \ldots, r_{63})$. The numerical label D assigned to R is given by: $D = \sum_{s=0}^{2^6-1} r_s 2^s$, which is simply the sum of the coefficients associated with all nonzero components.

333	...	013	012	011	010	003	002	001	000	Possible States
r_{63}	...	r_7	r_6	r_5	r_4	r_3	r_2	r_1	r_0	Rule Components
4^{63}	...	4^7	4^6	4^5	4^4	4^3	4^2	4^1	4^0	Coefficients

Fig. 1. Evolution rules of the four state classical CA model of DNA evolution

After the assignment of the origin DNA sequence by the user of the simulation tool, an initial population *P* that contains *n* possible solutions, meaning *n* CA evolution rules is constructed randomly. The value of *n* is user-defined and should be a compromise between accuracy and computer time and memory. For each possible solution *i* of population *P* with *n* individuals an error function is given by:

$$Mer(i) = \sum_{j=0}^{SET} | y(i,j) - \hat{y}(i,j) | \tag{2}$$

where $y(i, j)$ is the measured state at data point j for chromosome i and $\hat{y}(i,j)$ is the predicted state, in correspondence. Each chromosome in the current population is ranked with respect to *Mer* of equation (2). The chromosome with the least *Mer* occupies the first position, the chromosome with the second least *Mer* occupies the second position and so on. Chromosomes with the same error share the same rank. After the final ranking we calculate the fitness function for each chromosome. The fitness function of the i^{th} chromosome is defined as:

$$fit(i) = \frac{MAX(rank(i)) - rank(i)}{MAX(rank(i)) - MIN(rank(i))} \tag{3}$$

The pseudocode of the GA algorithm for the selection of CA evolution rules can be summarized as follows:

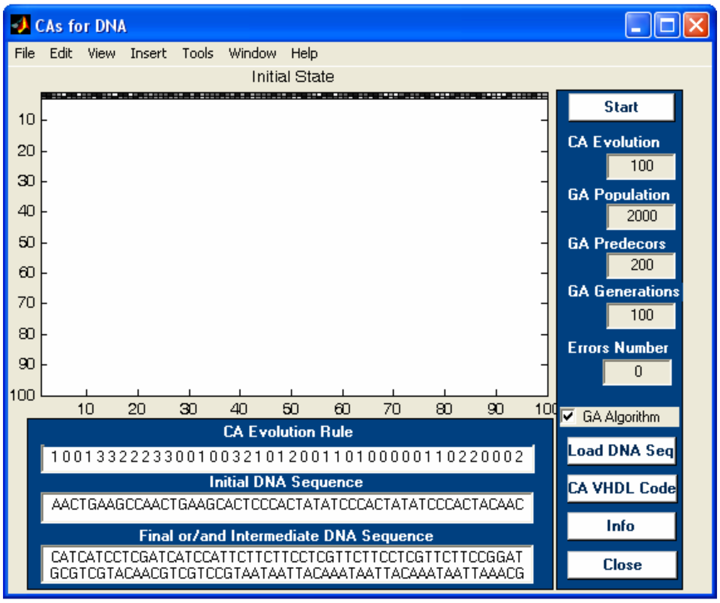
Table 1. Pseudocode of the GA algorithm for the selection of CA evolution rules

Pseudocode	Comments
1. Start	
2. Set the current generation number *i* =1.	/* Take a time step */
3. Set the GA algorithm parameters.	/* User defined GA parameters */
4. Generate the population set *P* with *n* individuals.	
5. Compute *Mer* (modulus of error function) for each individual in *P*.	/* Use Equation 2 */
6. Rank the individual in *R*.	
7. Calculate the fitness function.	/* Use Equation 3 */
8. Apply the parent selection technique to *P*.	
9. Employ crossover and mutation to *P* to produce the corresponding offspring set *P*'.	
10. Calculate the corresponding fitness function for the chromosomes in the offspring set *P*'. Select the n fittest individual from both the population set *P* and the corresponding set *P*', by comparing the fitness value. Reset *P* using the corresponding newly selected n individuals and nullify the offspring set *P*'.	
11. Set the generation number *i*=*i*+1.	
12. If generation number less than a prespecified number of generations *G*	/* *G* is the user defined number of generations */
Go to 7 and repeat until has been reached	
Else finish.	/* Final state */
13. Stop	

A paradigm of the functional operation of *CAs for DNA* is presented in Fig. 2. These simulations show that the evolution data visualization is straightforward, and the evolution patterns can be studied and interpreted [7].

4 Automatic FPGA Implementation

To implement the aforementioned CA model in hardware, synchronous very large scale integrated (VLSI) circuits should be used. These implementations could lead to dedicated FPGA processors that can be designed using commercially available FPGA CAD systems. Furthermore, the hardware implementation of the algorithms could be achieved after the manual translation of their parts into a synthesizable subset of a hardware description language (HDL).



CA Evolution: 100
GA Population: 2000
GA Predecors: 200
GA Generations: 100
Errors Number: 0
✓ GA Algorithm

CA Evolution Rule
1 0 0 1 3 3 2 2 2 3 3 0 0 1 0 0 3 2 1 0 1 2 0 0 1 1 0 1 0 0 0 0 0 1 1 0 2 2 0 0 0 2

Initial DNA Sequence
AACTGAAGCCAACTGAAGCACTCCCACTATATCCCACTATATCCCACTACAAC

Final or/and Intermediate DNA Sequence
CATCATCCTCGATCATCCATTCTTCTTCCTCGTTCTTCCTCGTTCTTCCGGAT
GCGTCGTACAACGTCGTCCGTAATAATTACAAATAATTACAAATAATTAAACG

Fig. 2. The final screen of the *CAs for DNA* after the execution of the GA algorithm. (The graphical user interface of the simulator) (A: white, C: dark gray, T: light gray and G: black.)

The presented system *CAs for DNA* is able of automatic translation of the CA algorithm's code into synthesizable VHDL code based on the user's choice of simulation parameters. There are many reasons for implementing an algorithm, which simulates a system, using a hardware description language, and especially VHDL, instead of using standard VLSI design CAD tools. Mainly, because the VHDL models present the most reliable design process with the minimum cost and time and, furthermore, because they are capable of avoiding design errors. Furthermore, because the execution time in software depends on the complexity of the rule, while the execution time (throughput) in hardware is almost independent of the rule complexity. In our CAD tool, the primary parameters of the translation algorithm are used to produce the VHDL code. In the beginning, the CA algorithm is read by the translation algorithm. After the CA algorithm is read, the translation algorithm searches the CA code to detect the CA rule found by the GA algorithm in order to produce the VHDL code for the main component, i.e. the CA cell. This will be the behavioral part of the final

VHDL code, containing process and signal assignment statements. Subsequently, the translation algorithm searches the CA code to detect the lattice size, the boundary and initials CA conditions, in order to construct the structural part of the final VHDL code. The structural part implements the final module as a composition of subsystems, like the aforementioned main component (schematic Fig. 3). The final VHDL code produced by translation algorithm, including both the behavioral and structural parts, addresses all the basic VHDL concepts (i.e. interfaces, behavior, structure, test benches) included in the IEEE Standard 1076-2002. No previous knowledge of VHDL is required, since the VHDL code is directly produced from the high-level programming language code through the translation algorithm. However, there is always a possibility of functional simulation of the VHDL code with the use of the appropriate automatically generated test benches. The simulation results of the VHDL code are guaranteed to be found in complete agreement with the compilation results of the CA algorithm, produced during the phase of estimating the CAs algorithm performance and of verifying its functional correctness.

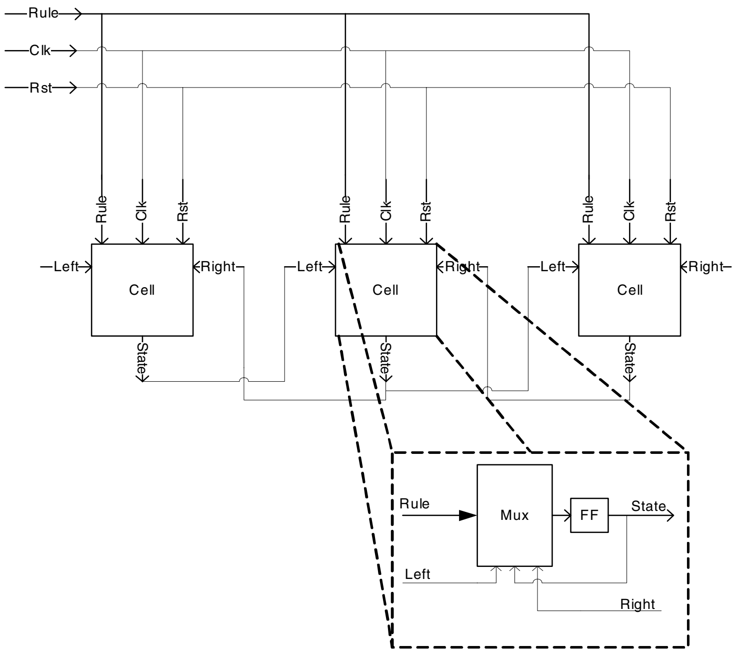


Fig. 3. Three (3) cells neighbourhood of the proposed CA architecture and its corresponding basic structural element

The automatically produced synthesizable VHDL CA code is translated into a hardware schematic of the defined architecture using predetermined timing constraints in Quartus II, v. 7.2® design software. The next step includes translation and mapping. In this phase the hardware schematic is mapped to the specific hardware of the FPGA and the communication channels between the generated components are specified. The final phase is the generation of a configuration file. Finally, this translates the mapped design into a stream of bits that control the switchboxes, LUTs and other components of the FPGA. It should be mentioned that for hardware implementation purposes, the performance and the size of reconfigurable hardware such as FPGAs have been drastically improved in the last several years. With the latest FPGA chips, more than one hundred grids can be computed in one clock cycle (less than 50 ns), and the reconfigurability of FPGAs makes it possible to compute any kind of CA on the same chip.

Design of the proposed processor results in an ALTERA Stratix EP1S10F484C5 FPGA device, which indicates a maximum clock rate around 240MHz and consists of 100 CA cells. Initial data is loaded in a semi-parallel way and the automatic response of the processor provides the CA evolution of the DNA sequence under test. More specifically, inputs to the dedicated processor are the lines through which the initial conditions are transferred to the CA, the clock, the reset and load control signals, the boundary condition signals, as well as the power and ground connections. Furthermore, for comparison purposes we have evaluated the speed of the traditional software Matlab CA code implementation running on a typical Pentium IV 3GHz Windows XP computer system and the results justify the aforementioned integration of the FPGA processor. More specifically, a speed-up of 6 times for a medium length CA and a speed up of 23 times for an extra long length CA was measured concluding that the implementation of CA is significantly faster in FPGA hardware than in optimized software, thus enabling real parallel processing of data using custom digital structures. As a result, the proposed CA is running faster when implemented to a dedicated ASIC processor compared to a general purpose computer.

5 Conclusions

In this paper, *CAs for DNA*, an automated simulation and hardware implementation tool for DNA sequence evolution by extracting CA rules with the usage of proper GAs was developed. *CAs for DNA* was based on a CA DNA evolution model. Based on this model, a CAD tool of DNA evolution was developed, a GA methodology has been presented to determine the evolution rules generating given evolution patterns and a fast FPGA processor that executes the CA model was fully automatically designed, placed and routed. Speed is extremely significant in this application domain and it is really important to observe that hardware performance becomes available out of a general purpose FPGA card. As future work, the FPGA processor as well as the GA algorithm for the selection of the CA evolution rule, can be calibrated with real data (DNA sequences) of different microorganisms in various evolution steps targeting to the production of suitable drugs. Moreover, Register Transfer Level (RTL) design could be used by the proposed tool for the CA implementation giving better results. Furthermore, this work will facilitate the development of CA models of the self-organizing properties of DNA. As CAs models are developed, they are expected to contribute to the interpretation of DNA sequences, and possibly indicate new directions in the field of artificial intelligence for bioinformatics.

References

1. Cios, K.J., Mamitsuka, H., Nagashima, T., Tadeusiewicz, R.: Computational intelligence in solving bioinformatics problems. Artificial Intelligence in Medicine 35, 1–8 (2005)
2. Baxevanis, A.D., Ouellette, B.F.: Bioinformatics, a practical guide to the analysis of genes and proteins. Wiley-Interscience, New York (1998)
3. Sirakoulis, G.C., Karafyllidis, I., Mizas, C., Mardiris, V., Thanailakis, A., Tsalides, P.: A cellular automaton model for the study of DNA sequence evolution. Computers in Biology and Medicine 33, 439–453 (2003)
4. Goldberg, D.A.: Genetic Algorithms in Search, Optimization and Machine Learning. Addison-Wesley, Reading (1989)
5. McFadden, J., Al-Khalili, J.: A quantum mechanical model of adaptive mutation. BioSystems 50, 203–211 (1999)
6. Schwefel, H.P.: Deep insight from simple models of evolution. BioSystems 64, 189–198 (2002)
7. Mizas, C., Sirakoulis, G.C., Mardiris, V., Karafyllidis, I., Glykos, N., Sandaltzopoulos, R.: Reconstruction of DNA sequences using genetic algorithms and cellular automata: Towards mutation prediction? Biosystems 92, 61–68 (2008)