Hindawi Publishing Corporation
Discrete Dynamics in Nature and Society
Volume 2013, Article ID 250593, 4 pages
http://dx.doi.org/10.1155/2013/250593

Research Article

Generation of a Reconfigurable Logical Cell Using Evolutionary Computation

I. Campos-Cantón,[1] L. M. Torres-Treviño,[2] E. Campos-Cantón,[3] and R. Femat[3]

[1] *Facultad de Ciencias, Universidad Autónoma de San Luis Potosí, Alvaro Obregón 64, 78000 San Luis Potosí, SLP, Mexico*
[2] *Centro de Innovación, Investigación y Desarrollo en Ingeniería Electrónica, Universidad Autónoma de Nuevo León, Km. 10 de la Nueva Carretera al Aeropuerto Internacional de Monterrey, PIIT Monterrey, 66600 Apodaca, NL, Mexico*
[3] *División de Matemáticas Aplicadas, Instituto Potosino de Investigación Científica y Tecnológica, Camino a la Presa San José 2055, Colonia Lomas 4 Sección, 78216 San Luis Potosí, SLP, Mexico*

Correspondence should be addressed to I. Campos-Cantón; icampos@fciencias.uaslp.mx

Received 26 October 2012; Revised 12 February 2013; Accepted 13 February 2013

Academic Editor: Gualberto Solís-Perales

In nature, an interesting topic is about how a cell can be reconfigured in order to achieve a different task. Another interesting topic is about the learning process that seems to be a trial and error process. In this work, we present mechanisms about how to produce a reconfigurable logical cell based on the tent map. The reconfiguration is realized by modifying its internal parameters generating several logical functions in the same structure. The logical cell is built with three blocks: the initial condition generating function, the tent map, and the output function. Furthermore, we propose a reconfigurable structure based on a chaotic system and an evolutionary algorithm is used in order to tune the parameters of the cell via trial and error process.

1. Introduction

Adaptation in nature is a relevant research area that has many applications in artificial systems, which can be used for the benefit of society. Particularly in biology, a neuron can reconfigure itself to develop different tasks using the same structure; however, this procedure is a mystery. The processes of adaptation, learning, and coupling between them have been research pursuits therein, and the understanding of this processes can help to build artificial devices. The act of joining living tissue with electronics has long been imagined in the world of science fiction, but cybernetic organisms are now one step closer to reality, thanks to work emerging from researchers that have built tiny electronic meshes out of silicon nanowires and have used them as scaffolds to grow nerve, heart, and muscle tissue.

Chaotic dynamical systems are commonly used in communication systems, like the generation of pseudorandom signals, and encryption, among others. Chaotic systems are recognized by the richness of their dynamics; however, these systems could be very sensible to perturbations or a combination of an infinite number of instabilities. Chaotic systems of one dimension may generate a remarkable variety of behaviors if they are observed as a function of time due to different initial conditions or their parameters [1]. The use of one-dimensional mapping is feasible, as the logistic map for instance or the tent map, to generate logical functions [2–4].

Nowadays, the searching for alternative solutions in the hardware used in computational systems is a very important aspect in the area [5]; one of them is the development of reconfigurable hardware, where the chaotic circuits are candidates to perform these types of tasks [6–8]. As expected, the chaotic elements might generate different logical functions, with reconfiguration capabilities [9, 10]. Other alternatives consist of architectures that use programmable gates circuits (FPGA's), whose configuration is made by rewiring multiple static gates of single purpose; that is, a single logic gate has a fixed configuration. In this work, a tent map 1 is used to build a reconfigurable cell that generates different logical basic functions where an evolutionary algorithm is used to adjust its parameters [11–13].

$$f(x) = \begin{cases} \mu x, & \text{if } x < 0.5, \\ \mu(1-x), & \text{if } x \geq 0.5. \end{cases} \quad (1)$$

Evolutionary computation is a paradigm inspired by the natural selection theory proposed by Charles Darwin, where several evolutionary processes can be accounted for in computing including three of them in the developing of an evolutionary algorithm: the fitness assignation, the inclusion of diversity, and the selection of more fitted individuals that can reproduce and have offspring. In evolutionary computation the same process that occurs in nature is emulated, but in a simple way. Genetic algorithms, genetic programming, and estimation of distribution algorithms are examples of this kind of algorithms [11, 12]. In this work, an easy implementation of an estimation of distribution algorithm called Evonorm [13] is used for tuning the parameters of the reconfigurable logic cell. This counts as the actual way for tuning the parameters usually made by trial and error process. Using this algorithm generated all the sixteen logic functions considering two inputs.

Although previous paragraphs seem to be uncorrelated, the possibility of designing reconfigurable logic cells by electronic circuits allows one (i) to explain how a same structure of artificial systems (circuits) can be reconfigured to develop different tasks; (ii) to incorporate chaotic dynamics as the basis of reconfiguration at the artificial systems (circuits); (iii) to define alternative architectures, distinct from FPGA's, such that the reconfiguration goes beyond rewiring; and (iv) to exploit evolutionary computation towards the optimization of parameters in the reconfiguration process. The main contribution of this paper is on a reconfigurable logic cell. This reconfigurable logic cell can represent several logical functions, all in the same structure changing specific parameters. The setting of these parameters is made by an evolutionary algorithm. In Section 2, a reconfigurable logic cell is presented. In Section 3, the Evonorm evolutionary algorithm is described and used to set the parameters of reconfigurable logic cell proposed. In Section 4, the proposal architecture is used to generate different logical functions. Conclusion and future work are given in the last section.

2. Reconfigurable Dynamical Logic Cell

In our proposal, the reconfigurable logical cell is constituted by an architecture consisting of three blocks: (i) a generator of initial conditions, (ii) a tent map, and (iii) an output block. Each block obeys the following:

(i) the initial condition of the reconfigurable logic cell is defined by the following equation:

$$x_0 = (x_s + BU) \mod 1, \quad (2)$$

where $x_s \in (0,1)$ is an ignition seed, $B = [b_1 \ \ b_2] \in R^2$, and $U = [u_1 \ \ u_2]^T$ with $u_1, u_2 \in \{0,1\}$;

(ii) the tent map yields the first iteration using (1); that is,

$$x_1 = f(x_s + b_1 u_1 + b_2 u_2); \quad (3)$$

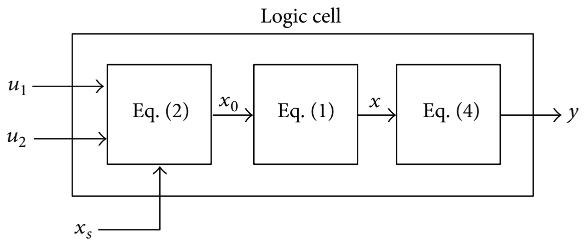

Figure 1: Block diagram of the logical cell.

(iii) the output block is a function $y:(0,1) \to \{0,1\}, x_1 \mapsto \{0,1\}$, generating a bistable response as follows:

$$y(x) = \begin{cases} 1, & \text{if } x_1 > \beta, \\ 0, & \text{otherwise}, \end{cases} \quad (4)$$

where β is a threshold reference signal to determine the output of the system, a logical zero, or a logical one.

Figure 1 illustrates a block diagram of a reconfigurable logic cell defined by (1), (2), and (4). Considering the inputs u_1, u_2, the reconfigurable logic cell generates 16 logic functions. Without a loss of generality, we explain two functions f_{14} and f_8 (OR & AND logic gates) of these 16 whose outputs satisfy the true Table 1.

2.1. Function f_{14} (OR). For the function f_{14}, the reconfigurable logical cell is tuned by considering the following parameters $\mu = 2$, $b_1 = b_2 = 0.2$, $\beta = 0.5$, and $x_s = 0.1$. These parameters satisfy the response shown in Table 2, column Y_{OR}.

The behavior of the reconfigurable logic cell is described in what follows. If the initial condition is $x_0 = 0.1 + 0.25u_1 + 0.25u_2$ and inputs $u_1 = 0$, $u_2 = 0$, then $x_0 = 0.1$ and the first iteration of the tent map 1 is $f(x) = 0.2$. Thus, the output defined by 4 is $y = 0$. Repeating the exercise with $u_1 = 0$, $u_2 = 1$, we have $x_0 = 0.35$, $f(x) = 0.7$, and $y = 1$. Following this way, all the possible combinations are considered with two logic inputs to get the results illustrated in column $Y_{\text{OR}}(x)$ of Table 2, so the logical cell under the parameters defined above can generate the behavior of a logical function OR.

2.2. Function f_8 (AND). For this case, the same parameter values used above are considered, except the β parameter is now tuned to 0.75. Note that only one parameter is reconfigured. The output of the logic gate is shown in Table 2, column y_{AND}. This means that only one parameter is changed for the reconfiguration of the logical cell generating from the logical function OR to the logical function AND.

Note that the selection of the parameter values can be adjusted by trial and error but this was an exhausted task; hence, the evolutionary algorithms arise as a convenient alternative. We propound the Evonorm evolutionary algorithm to be used for the selection of the parameter values and the algorithm explained in the next section.

Table 1: True table for OR and AND logic gates.

u_1	u_2	OR	AND
0	0	0	0
0	1	1	0
1	0	1	0
1	1	1	1

Table 2: Outputs for the logical functions f_{14} and f_8 (OR and AND logical gates).

u_1	u_2	x_s	$f(x)$	$Y_{\mathrm{OR}}(x)$	$Y_{\mathrm{AND}}(x)$
0	0	0.1	0.2	0	0
0	1	0.1	0.7	1	0
1	0	0.1	0.7	1	0
1	1	0.1	0.8	1	1

3. Evolutionary Computation

An important question is about the existence of an efficient way to determine the values of the parameters of the reconfigurable logic cell. In this section, we show how parameters can be tuned in order that the logical functions will be generated. Particularly, we illustrate the case of 2 inputs generating 16 outputs. An evolutionary algorithm is used to this end. Essentially, an evolutionary algorithm is used for searching solutions [11, 12]. Here we exploit it to tune the parameters of the reconfigurable logic cell. The selected evolutionary algorithm is called Evonorm [13] and is based on four items as follows:

(1) representation of individuals;

(2) evaluation or fitness calculation of every individual;

(3) selection of the best fitted individuals;

(4) generation of new individuals.

Tracking the above four items, every potential solution is represented as an individual of a population where it is assigned an evaluation value depending on its performance to solve the problem of tuning the parameters. Specifically, In Evonorm, a marginal random variable is used for yielding a new population via an estimation of a normal distribution. The implementation of the algorithm requires two matrices and three vectors. One of the two matrices is named $P \in \mathbb{R}^{T_i \times T_P}$, which represents the population of solutions, where T_i is the number of total individuals and T_P is the number of parameters. The other matrix is denoted as $P_s \in \mathbb{R}^{T_s \times T_P}$, which stores selected individuals, where T_s is the number of selected individuals, usually ten percent of the total population. The vector $FE \in \mathbb{R}^{T_i \times 1}$ stores the evaluation value per individual. The vectors $\mu \in \mathbb{R}^{T_P \times 1}$ and $\sigma \in \mathbb{R}^{T_P \times 1}$ are used for storing the mean and standard deviation, respectively, of the random variable used per parameter. The above definitions correspond to the first item of evolutionary algorithm Evonorm. The initial population is generated randomly using a uniform distribution function.

In what follows, the Evonorm evolutionary algorithm is highlighted for determining the initial conditions of the reconfigurable logic cells. Every individual k of the population is evaluated as follows.

Step 1. Evaluation and extraction of the logical cell parameters. Extraction of the parameters: $x_s = P(k,1)$, $\mu = P(k,2)$, $\beta = P(k,3)$, $b_1 = P(k,4)$, and $b_2 = P(k,5)$.

Step 2. Calculation of $r = \sum_{\mathrm{pr}=1}^{T_P} |yd_{\mathrm{pr}} - y(x_s), \mu, k, b_1, b_2)|/4$, where yd_{pr} is a vector that represents the expected output values corresponding to every input combination in the cell. For example, the function f_7 has the following outputs $yd_{\mathrm{pr}} = [1; 1; 1; 0]$ that correspond to the inputs $[00; 01; 10; 11]$, respectively. The maximum number of outputs is considered to perform the normalization; in this example, number four is used for the normalization of the evaluation.

Step 3. Determine the evaluation as $FE_k = 1 - r$. A maximization of this function is expected.

Step 4. Selection of the most fitted individuals. This procedure sorts the individuals of matrix P using the evaluation vector FE as a criterion. Then a selection of T_i individuals is made for generating a new population P_s. In this procedure is stored the best fitted individual in vector $I_x \in \mathbb{R}^{T_s \times T_P}$.

Step 5. Calculate the mean and the standard deviation per parameter:

$$\mu_{\mathrm{pr}} = \frac{\sum_{k=1}^{T_s} \left(P_{s_{k,\mathrm{pr}}}\right)}{T_s},$$
$$\sigma_{\mathrm{pr}} = \frac{\sqrt{\left(\sum_{k=1}^{T_s} \left(P_{s_{k,\mathrm{pr}}} - \mu_{\mathrm{pr}}\right)\right)}}{T_s}. \tag{5}$$

Step 6. Generate a new population considering a marginal random variable with normal distribution:

$$P_{k,\mathrm{pr}} = \begin{cases} N\left(\mu_{\mathrm{pr}}, \sigma_{\mathrm{pr}}\right), & U() > 0.5, \\ N\left(I_{x_{\mathrm{pr}}}, \sigma_{\mathrm{pr}}\right), & \text{otherwise}. \end{cases} \tag{6}$$

$N(\mu_{\mathrm{pr}}, \sigma_{\mathrm{pr}})$ is a random variable that generates numbers with a normal distribution function considering a mean μ_{pr} and a standard deviation σ_{pr}, precalculated previously. The approximation of a standard random variable with a normal distribution is calculated using (7), where $U(\ )$ is a random variable with normal distribution:

$$N(\mu, \sigma) = \mu + \sigma \left(\sum_{i=1}^{12} (U()) - 6 \right). \tag{7}$$

Steps 1–6 are repeated several times in cycles called generations. This algorithm is capable of adjusting the parameters of the reconfigurable logical cell and generates all the logical functions for two inputs u_1, u_2.

Table 3: Logic functions.

u_1	u_2	f_0	f_1	f_2	f_3	f_4	f_5	f_6	f_7	f_8	f_9	f_{10}	f_{11}	f_{12}	f_{13}	f_{14}	f_{15}
0	0	0	1	0	1	0	1	0	1	0	1	0	1	0	1	0	1
0	1	0	0	1	1	0	0	1	1	0	0	1	1	0	0	1	1
1	0	0	0	0	0	1	1	1	1	0	0	0	0	1	1	1	1
1	1	0	0	0	0	0	0	0	0	1	1	1	1	1	1	1	1

Table 4: Parameter's values for generating 16 logic functions.

Función	μ	b_1	b_2	β	x_s
f_0	2	0.25	0.25	0.85	0.111
f_1	2	0.3	0.3	0.9	0.45
f_2	2	0.7	0.15	0.5	0.111
f_3	2	0.3	0.2	0.8	0.4
f_4	2	0.15	0.7	0.5	0.111
f_5	2	0.2	0.4	0.5	0.4
f_6	2	0.45	0.45	0.25	0.111
f_7	2	0.45	0.45	0.1	0.111
f_8	2	0.25	0.25	0.75	0.111
f_9	2	0.5	−0.1	0.6	0.3
f_{10}	2	0.1	0.25	0.5	0.111
f_{11}	2	−0.1	0.4	0.6	0.3
f_{12}	2	0.25	0.1	0.5	0.111
f_{13}	2	0.4	−0.1	0.6	0.3
f_{14}	2	0.25	0.25	0.5	0.111
f_{15}	2	0.25	0.25	0.2	0.111

4. Tuning Parameter Values by Evonorm

Evonorm algorithm is used to generate a better parametric solution for all the logic functions considering two input variables, as shown in Table 3.

The parameter values for all the logic functions with two inputs were generated by an evolutionary algorithm (Table 4). For example, the boolean function f_{12} is generated by considering the parameter values $x_s = 0.111$, $\mu = 2$, $b_1 = 0.25$, $b_2 = 0.1$, and $\beta = 0.5$ according to Table 4. The initial condition x_0 is equal to 0.11 with inputs $u_1 = 0$, $u_2 = 0$; so these value are used in (1) to get in the first iteration $f(x) = 0.222$, and the output $y = 0$ is given by (4). Repeating this process, all the outputs are generated, as illustrated in Table 4.

5. Conclusion

A development of a reconfigurable logic cell based on tent map is presented. The reconfiguration is made by tuning the parameter values: x_s, β, b_1, b_2, and μ. The Evonorm algorithm is a useful tool for tuning the parameter values of the reconfigurable logic cell and generating all the 16 boolean functions of two logical inputs (remaining fixed μ, Table 4). As a future work, a comparison between other evolutionary algorithms and the use of reconfigurable logic cells with three or more inputs is expected.

Acknowledgment

E. Campos-Cantón acknowledges CONACYT for the financial support through Project no. 181002.

References

[1] E. Campos-Cantón, R. Femat, and A. N. Pisarchik, "A family of multimodal dynamic maps," *Communications in Nonlinear Science and Numerical Simulation*, vol. 16, no. 9, pp. 3457–3462, 2011.

[2] S. Sinha and W. Ditto, "Dynamics based computation," *Physical Review Letters*, vol. 81, no. 10, pp. 2156–2159, 1998.

[3] S. Sinha and W. Ditto, "Computing with distributed chaos," *Physical Review E*, vol. 60, no. 1, pp. 363–377, 1999.

[4] T. Munakata, S. Sinha, and W. L. Ditto, "Chaos computing: implementation of fundamental logical gates by chaotic elements," *IEEE Transactions on Circuits and Systems I*, vol. 49, no. 11, pp. 1629–1633, 2002.

[5] D. Kuo, "Chaos and its computing paradigm," *IEEE Potentials*, vol. 24, no. 2, pp. 13–15, 2005.

[6] K. Murali, S. Sinha, W. L. Ditto, and A. R. Bulsara, "Reliable logic circuit elements that exploit nonlinearity in the presence of a noise floor," *Physical Review Letters*, vol. 102, no. 10, Article ID 104101, 4 pages, 2009.

[7] W. L. Ditto, A. Miliotis, K. Murali, S. Sinha, and M. L. Spano, "Chaogates: morphing logic gates that exploit dynamical patterns," *Chaos*, vol. 20, no. 3, Article ID 037107, 7 pages, 2010.

[8] K. Murali, A. Miliotis, W. L. Ditto, and S. Sinha, "Logic from nonlinear dynamical evolution," *Physics Letters A*, vol. 373, no. 15, pp. 1346–1351, 2009.

[9] I. Campos-Cantón, E. Campos-Cantón, J. A. Pecina-Sanchez, and H. C. Rosu, "A simple circuit with dynamic logic architecture of basic logic gates," *International Journal of Bifurcation and Chaos*, vol. 20, no. 8, pp. 2547–2551, 2010.

[10] H. Peng, Y. Yang, L. Li, and H. Luo, "Harnessing piecewise-linear systems to construct dynamic logic architecture," *Chaos*, vol. 18, no. 3, Article ID 033101, 6 pages, 2008.

[11] J. H. Holland, *Adaptation in Natural and Artificial Systems: An Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence*, MIT Press, Ann Arbor, Mich, USA, 1975.

[12] D. E. Goldberg, *Genetic Algorithms in Search, Optimization, and Machine Learnin*, Addison-Wesley, New York, NY, USA, 1989.

[13] L. Torres, "Evonorm: easy and effective implementation of estimation of distribution algorithms," in *Research in Computing Science*, A. Gelbuckh and S. S. Guerra, Eds., vol. 23, pp. 75–83, 2006.