Publicado en Evolvable Systems: From Biology to Hardware (ICES 2003)
Tyrrell, Haddow y Torrensen (Eds.), Lecture Notes in Computer Science 2606
Springer-Verlag, 2003, pp.249-261

Routing of Embryonic Arrays Using Genetic Algorithms

Cesar Ortega-Sanchez[1], Jose Torres-Jimenez[2] and Jorge Morales-Cruz[3]

[1]Institute of Electrical Research, Department of Control and Instrumentation
Av. Reforma 113, Palmira, Cuernavaca, Morelos, Mexico 62490
ortegas@iie.org.mx
[2] Computer Science Department, ITESM
Av. Reforma 182-A, Lomas de Cuernavaca, Temixco, Morelos, Mexico 62589
jtorres@campus.mor.itesm.mx
[3] Electronics Department, CENIDET
Interior Internado Palmira S/N, Palmira, Cuernavaca, Morelos, Mexico 62490
jmc_190777@hotmail.com

Abstract. This paper presents a genetic algorithm (GA) that solves the problem of routing a multiplexer network into a MUXTREE embryonic array. The procedure to translate the multiplexer network into a form suitable for the GA-based router is explained. The genetic algorithm works on a population of configuration registers (genome) that define the functionality and connectivity of the array. Fitness of each individual is evaluated and those closer to solving the required routing are selected for the next generation. A matrix-based method to evaluate the routing defined by each individual is also explained. The output of the genetic router is a VHDL program describing a look-up table that receives the cell co-ordinates as inputs and returns the value of the corresponding configuration register. The routing of a module-10 counter is presented as an example of the capabilities of the genetic router. The genetic algorithm approach provides not one, but multiple solutions to the routing problem, opening the road to a new level of redundancy where a new "genome" can be downloaded to the array when the conventional reconfiguration strategy runs out of spare cells.

1 Introduction

The Embryonics project was originally proposed by Mange et al [1] from the Êcole Polytechnique Fédérale de Laussane in Switzerland, and soon after the University of York, UK, joined the efforts under the lead of Tyrrell [2]. During the past few years research on Embryonics has gained momentum and the first practical demonstrations of the technology have already been shown [3, 4]. Embryonics proposes the incorporation of biological concepts like growth, reproduction and healing into the realm of silicon processor-arrays [5]. One of the possible Embryonics implementations is the MUXTREE architecture [6, 7]. In MUXTREE implementations, a logic function is translated from truth tables into multiplexer networks via ordered binary decision diagrams (OBDDs). The multiplexer network has to be mapped into the structure of an embryonic array, which is a 2-D array of processing cells. Every cell performs the

function of a two-input, one-output multiplexer. Connectivity between cells is limited to the nearest neighbours, and the number of data lines connecting cells to each other are scarce. Inputs to the multiplexer and routing of signals within each cell are set by a configuration register. In the MUXTREE architecture, each cell of the array contains a copy of the configuration registers of all the cells in the array [8].

The set of all configuration registers in an array is called "the genome" of the application. In each cell, a configuration register is selected by a unique pair of co-ordinates. When a cell fails, it is possible to reconfigure the array by changing the co-ordinates of cells so that the failing cell is logically eliminated and substituted by a healthy one. By these means embryonic arrays achieve fault-tolerance [9].

Connectivity limitations make mapping the multiplexer network into the embryonic array a task that grows in difficulty as the complexity (size) of the application grows. For small functions, it is possible, although time-consuming, to manually route the multiplexer network into the embryonic array. However, the time (and patience!) required to route medium-size applications is unacceptable for all practical purposes.

Given that Embryonics is a relatively new field, it lacks the CAD tools available for other, more mature bio-inspired technologies (artificial neural nets and genetic algorithms, for example). In its present state, MUXTREE applications can be synthesised and simulated using FPGA design software, like Xilinx's Foundation. However, the early stages of a design have to be developed by hand.

This paper presents a recently developed design tool that helps the designer in what is considered the most challenging and time-consuming task during the development of a MUXTREE application: the mapping of a multiplexer network into the fixed structure of an embryonic array. At the core of the router is a genetic algorithm (GA) that selects from a multitude of possible routings those that satisfy the connectivity of the multiplexer network given as input.

The paper is organised as follows: Section 2 introduces the MUXTREE architecture and the typical design flow of an application. Section 3 describes the genetic algorithm that solves the routing. In Section 4, a module-10 counter is presented as an example of the capabilities of the genetic algorithm employed as router. Results obtained with other examples are also presented. Section 5 resumes the conclusions and future work of this research.

2 Design Flow of a MUXTREE Application

Although the processing power of an individual MUXTREE cell is that of a multiplexer, an array of such cells can contain a multiplexer network representing any combinational or sequential logic function. The size of the network would be limited only by the physical dimensions of the array (allowing, of course, spare elements necessary for the embryonic reconfiguration).

Figure 1 shows a simplified diagram of the MUXTREE embryonic cell. A detailed description of the architecture can be found elsewhere [8].

Publicado en Evolvable Systems: From Biology to Hardware (ICES 2003)
Tyrrell, Haddow y Torrensen (Eds.), Lecture Notes in Computer Science 2606
Springer-Verlag, 2003, pp.249-261

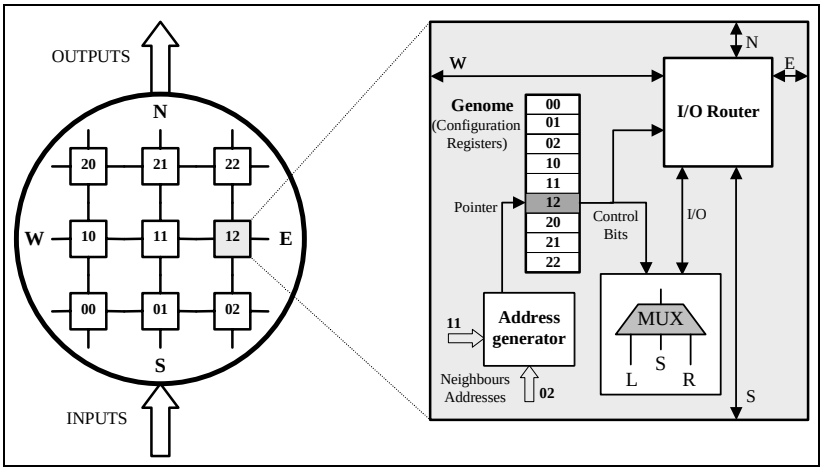


Fig. 1. Simplified diagram of a MUXTREE embryonic cell

In a typical application, the following steps are necessary to implement a combinational or sequential logic function using a MUXTREE embryonic array:

1. Describe the application as a set of logic functions or as a truth table.
2. From the description, construct an Ordered Binary Decision Diagram (OBDD).
3. Represent the OBDD as a multiplexer network.
4. Route the multiplexer network in an array of MUXTREE cells.
5. Obtain the configuration register of all the cells in the array.
6. Generate the "genome" of the application by grouping the configuration registers of all the cells in the array.
7. Use the genome as input for the software that is used to synthesise and simulate the design.

In a typical implementation, steps 1 to 6 are carried out by hand. Step 4 is the most difficult and time-consuming due to the limited interconnection between cells. A genetic algorithm-based, automatic router that implements steps 4 to 6 has been developed. The following section describes this tool in detail.

3 Routing Multiplexer Networks Using a Genetic Algorithm

During the past few years, genetic algorithms have been intensively used to solve problems whose solution either cannot be found analytically, or numeric methods take too long to find [10]. GAs are very good at finding solutions in vast search-spaces because they are inspired in the process of natural selection, where individuals who are fitter than others have more chances of passing their genes to the next generation [11].

The following elements are needed to solve a problem using GAs:

- A suitable representation of the problem so that an individual from a population can represent a possible solution.
- A randomly generated population of possible solutions.

- A fitness function that evaluates how close to solving the problem is every individual in the population.
- A selection criterion that discriminates individuals according to their fitness.
- Some genetic operators (typically mutation and crossover) to generate diversity within the population and to prevent the solutions from being stuck around local minima.
- A stop criterion that allows the system to decide when to finish the search. It can be either because a solution has been found, or because a specified number of generations have been tested.

Figure 2 shows a flow diagram representing the logical flow of a GA.

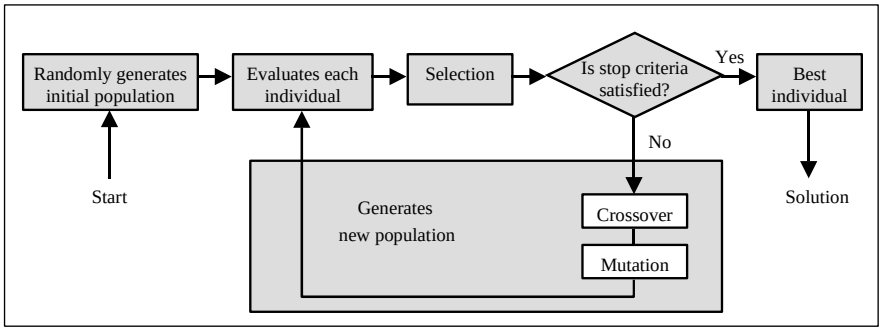


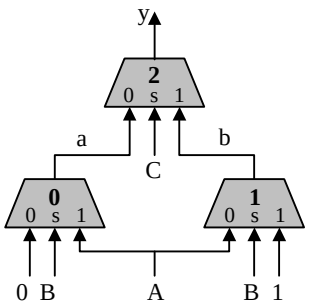
Fig. 2. Flow diagram of a typical genetic algorithm.

To solve the routing of multiplexer networks into MUXTREE arrays, the diagram in figure 2 was implemented in a C program. Three text input files indicate the topology of the target multiplexer network. Their content is presented next.

3.1 Input Files

For all practical purposes, a multiplexer network can be completely described by stating: the number of levels it contains, the number of multiplexers in each level and the inputs that arrive to each multiplexer (data and control). Additionally, in the MUXTREE architecture, it is necessary to distinguish the multiplexers that work in synchronous mode, i.e. with their outputs latched by a clock; from those that work asynchronously. The following example shows a simple multiplexer network and the input files associated to it.

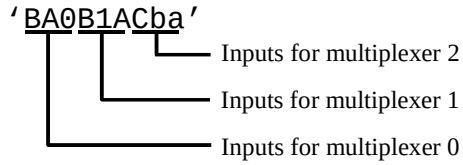
In file number one, parameters pertaining to the GA are provided along with the size of the MUXTREE array that should contain the multiplexer network. Also the features that characterise the target network are given. Figure 3 shows the multiplexer network of a 3-input voter, along with the text of input file 1.



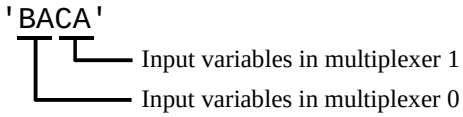

```
0.8     ;Crossover probability
0.15    ;Mutation probability
2 3 3   ;Size of destination array 2×3.
         3 multiplexers in the network.
2 2 1   ;Two levels with two multiplexers in
         the first level and one in the
         second level
0       ;No synchronous multiplexers
1 2     ;Number of outputs and tags of the
         multiplexers that carry the final
         output lines.
```

Fig. 3. 3-input voter multiplexer network and its input file 1 for the genetic router

The inputs for each multiplexer are indicated in a second input file. The file corresponding to the example in figure 3 is presented next. The order of inputs for each multiplexer is: **S**election, Right input (**1**), Left input (**0**). In a MUXTREE array, multiplexers are numbered from left to right and from bottom to top.



A third input file indicates the multiplexers that receive input variables, if any. In the MUXTREE architecture, variables can be fed to the array only through row 0, i.e. the bottom row of the array. Each multiplexer can receive up to two variables. In some difficult designs, the same variable can be fed in more than one input in order to simplify the routing. Following is the text of the third input file for the example in figure 3. For demonstration purposes, input variable A is fed through the two cells in the bottom row (the array size is 2×3).



Generation of the three input files can be easily automated, but at the present stage they are still hand-generated.

3.2 Initial Population

The function of a MUXTREE cell is defined by a 17-bit configuration register [5]. Eight of those bits define the routing of signals within the cell. This byte is called the routing byte. Therefore, the flow of signals within a particular array will be defined by the routing bytes of all cells in the array (the routing genome). The size of this genome in bits is 8 times the number of cells in the array. For a 2×3 array, its routing would be completely defined by 2×3×8= 48 bits; therefore, any randomly generated 48-bit

pattern will encode an arbitrary routing. In the genetic router proposed, every individual of the population is a routing genome. In the examples presented next, the initial population consists of 100 randomly generated individuals.

3.3 Fitness Function and Evaluation

The problem of evaluation is to find out how close an arbitrary routing is from the routing needed to implement the target multiplexer network. To achieve this, a carefully chosen fitness function must be used.

Definition 1: Two multiplexers are equivalent when they have the same inputs in the same order. Such inputs are defined as correct inputs.

Definition 2: Two multiplexers are partially equivalent when they have at least one input in the same position, i.e. at least one correct input.

By these definitions, a given routed MUXTREE array would be able to implement the same function as the target multiplexer network when it contains a set of multiplexers equivalent to those in the target network. In other words, the routing will be complete when it contains equivalent multiplexers for all multiplexers in the target network. Consequently, the fitness of a particular routing should be related to the number of equivalent and partially equivalent multiplexers. Hence, the fitness of a routing is defined as the number of correct inputs it contains. Since every multiplexer has three ordered inputs, the maximum fitness a routing can achieve is the number of multiplexers in the target network (n) multiplied by 3.

$$f_{max} = 3n \; . \tag{1}$$

To determine the fitness of a particular routing, it is necessary to locate all the correct inputs it contains. For this purpose, a connectivity matrix is used to "simulate" the propagation of signals within the network. Before propagation, the multiplexers in the embryonic array have no particular correspondent in the target network. After propagation, corresponding multiplexers are determined according to the number correct inputs every embryonic multiplexer has.

At the end of the evaluation process, every individual of a population will have a number associated to it. The bigger the number, the closer that routing is from solving the target network. For space reasons, a more detailed description of the connectivity matrix and the propagation process will be left for a future paper.

3.4 Selection

Selection is the process of choosing the individuals that will be parents of a new population. The genetic router presented in this paper implements selection by tournament, which consists in randomly selecting two genomes from the population and choosing the one with the highest fitness. For example,

If	$f_A > f_B$,	then:	$C_g = G_A$
If	$f_A < f_B$,	then:	$C_g = G_B$
If	$f_A = f_B$,	then:	$C_g = G_A$

Publicado en Evolvable Systems: From Biology to Hardware (ICES 2003)
Tyrrell, Haddow y Torrensen (Eds.), Lecture Notes in Computer Science 2606
Springer-Verlag, 2003, pp.249-261

Where:
f_A = Fitness of genome A; f_B = Fitness of genome B
G_A = Genome A; G_B = Genome B; C_g = Chosen genome

3.5 Crossover and Mutation

Crossover and mutation are used in the genetic router to prevent solutions from clustering around local minima.

Crossover consists of taking two 'parents' that have survived the selection process and mixing them to create two new individuals. Figure 4 shows two six-byte individuals being crossed over in two randomly selected points.

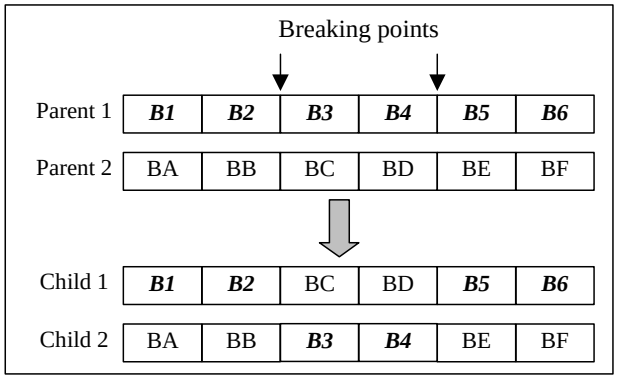


Fig.4. Crossover between two individuals of a population

The crossover points are selected in such a way that bytes remain unaltered so that well-routed cells are preserved from one generation to the next.

Crossover is performed according to a crossover probability. Every time a crossover is going to take place, a random number is generated; if the number is smaller than crossover probability, then the crossover takes place, otherwise both parents pass to the next generation unchanged. In the examples presented in this paper, crossover probability is 0.8.

It is possible to apply mutation to children generated by crossover. By means of a mechanism similar to the one used in crossover, a mutation probability will define the possibility of mutation. Mutation probability in the examples of this paper is 0.15. When mutation is applied, the byte and bit to be mutated are selected randomly. Mutation inverts the logic value of the selected bit.

3.6 New Population

A new population is generated with the individuals selected for their fitness. It is possible that two new individuals end up being identical after they have passed through crossover and mutation. If that is the case, one of the repeat individuals is

eliminated from the population and a brand new individual is randomly generated. All individuals in the new population are different to one another.

3.7 Output File

The genetic algorithm that solves the routing of multiplexer networks into MUXTREE arrays stops searching when one of the following conditions is met: A solution has been found, or the search has run a predetermined number of generations whether or not a solution has been found. The latter case can end up with none, one or multiple solutions to the routing.

If at least one solution is found, the genetic router generates a VHDL file containing the description of a look-up table that receives at its inputs the co-ordinates of a cell and returns the configuration register associated to that cell. This file has to be integrated to the design of the MUXTREE array that will implement the desired function. In section 4 there is a VHDL file generated by the genetic router.

If multiple solutions are found, a new level of fault-tolerance could be introduced to MUXTREE arrays. In its present implementation, MUXTREE arrays are disabled when spare cells have ran out and a new fault arises. However, with multiple routings capable of implementing the logic functions represented by the multiplexer network, it would be possible to download a new genome to the array every time spare cells run out. It is possible that one of the "spare genomes" can still implement the desired function.

4 Example: Routing of a Module-10 Counter

To demonstrate the functionality of the genetic router, the implementation of a module-10 counter is presented next. Table 1 shows the truth table of the counter and figure 5 the multiplexer network that implements it.

Table 1. Truth table of a module-10 counter

D	C	B	A	D^+	C^+	B^+	A^+
0	0	0	0	0	0	0	1
0	0	0	1	0	0	1	0
0	0	1	0	0	0	1	1
0	0	1	1	0	1	0	0
0	1	0	0	0	1	0	1
0	1	0	1	0	1	1	0
0	1	1	0	0	1	1	1
0	1	1	1	1	0	0	0
1	0	0	0	1	0	0	1
1	0	0	1	0	0	0	0
Other combinations				0	0	0	0

Table 2 presents the text of the three input files according to figure 5. The content of the third file is only a "0" because in a counter there are no external inputs, i.e. the application is purely sequential. This is indicated by the flip-flops at the outputs of the

multiplexers that deliver the counting function in figure 5. These outputs are latched by a clock common to all cells.

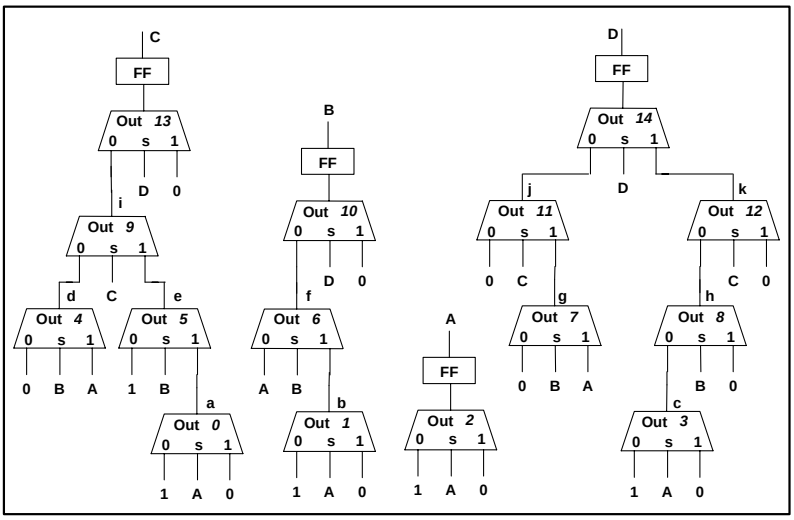


Fig. 5. Multiplexer network that implements a module-10 counter

Table 2. Input files for the genetic router

First input file	Second input file	Third input file
0.8 0.15 4 5 15 4 4 5 4 2 4 2 10 13 14 4 2 10 13 14	1A01A01A01A00BA1BaABb0BA cB0dCefD00CghC0iD0jDk	0

The following code is the VHDL file that the genetic router automatically generated when found the routing for the module-10 counter. The value of the configuration register for spare cells is specified by the case “when others”.

```
library IEEE;
entity Mem_counter9 is
    port (
        okaux: in STD_LOGIC;
        xy: in STD_LOGIC_VECTOR (7 downto 0);
        conf: out STD_LOGIC_VECTOR (16 downto 0));
end Mem_counter9;
architecture Mem_counter9_arch of Mem_counter9 is
type REG is array (16 downto 0) of bit;
begin
  process(okaux,xy)
    begin
      if ( okaux = '0' ) then
        conf <= "01001000100000000";
      else
        case xy is
          when "00000000" => conf <= 01000000101000101";
```

```
          when "00000001" => conf <= 00000000111100100";
          when "00000010" => conf <= 00000100111010011";
          when "00000011" => conf <= 00000000111000001";
          when "00000100" => conf <= 00000000010110100";
          when "00000101" => conf <= 10000000100101101";
          when "00000110" => conf <= 11001011100100001";
          when "00010000" => conf <= 01110000001110010";
          when "00010001" => conf <= 00100000101010101";
          when "00010010" => conf <= 00100001000100010";
          when "00010011" => conf <= 00100000001110001";
          when "00010100" => conf <= 00000010000000000";
          when "00010101" => conf <= 00001001110001000";
          when "00010110" => conf <= 00000000000101011";
          when "00100000" => conf <= 00011001000100110";
          when "00100001" => conf <= 01000101101110010";
          when "00100010" => conf <= 00011000001110001";
          when "00100011" => conf <= 00000001110000010";
          when "00100100" => conf <= 00000000000111001";
          when "00100101" => conf <= 00000000000110001";
          when "00100110" => conf <= 00100000111111011";
          when "00110000" => conf <= 01000101011111010";
          when "00110001" => conf <= 00000000010011010";
          when "00110010" => conf <= 00011101011110010";
          when "00110011" => conf <= 00000000001100110";
          when "00110100" => conf <= 00000000000011000";
          when others     => conf <= 01001000100000000";
        end case;
      end if;
  end process;
end Mem_counter9_arch;
```

Figure 6 shows part of the simulation of the module-10 counter implemented in a MUXTREE array. Synthesis and simulation were carried out in Xilinx's Foundation.

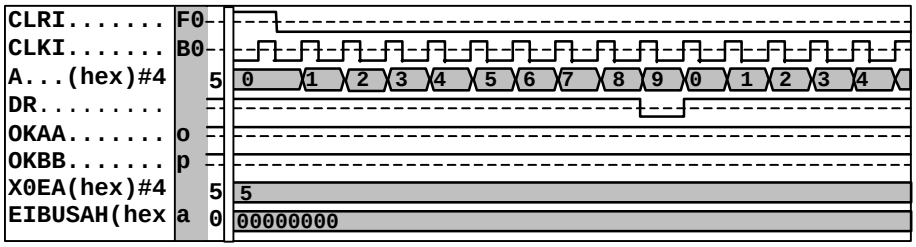


Fig. 6. Simulation of the module-10 counter implemented in a MUXTREE array

Figure 7 shows a graphical representation of one of the solutions found by the genetic router. The clock input is common to all cells. Solid lines represent fixed connections and broken lines are the connections programmed by the genome of the module-10 counter. Numbers in the multiplexers correspond to those in figure 5. A careful visual inspection of figure 7 will demonstrate that in fact, the multiplexer network in figure 5 is contained in the embryonic array presented.

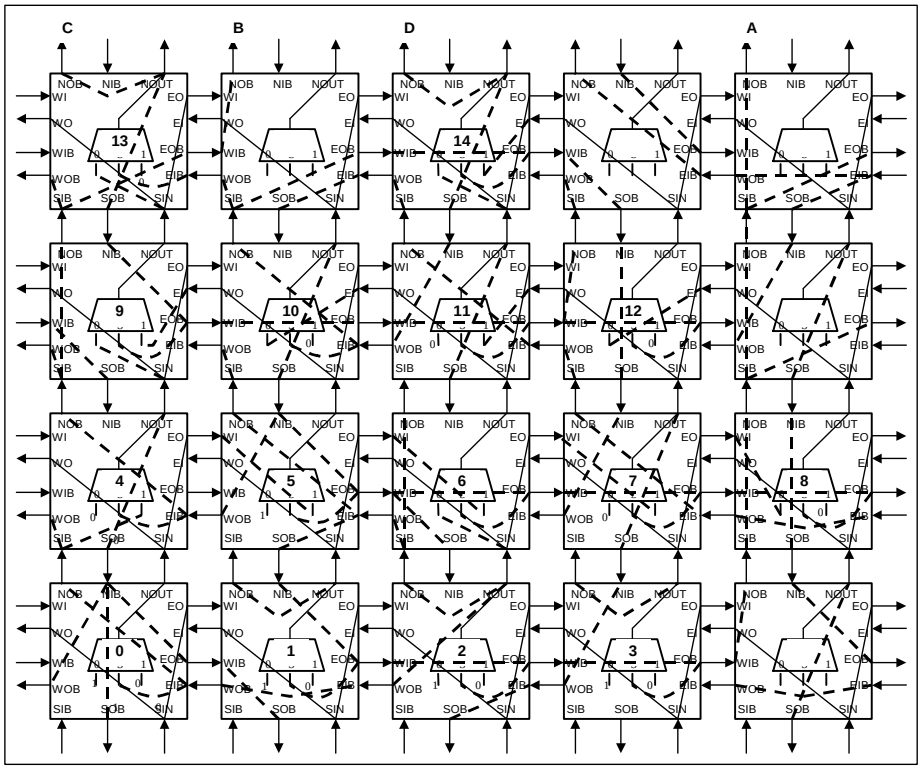


Fig. 7. Routing of a module-10 counter in a 4×5 MUXTREE array

4.1 Results with Other Examples

Table 3 resumes the results obtained when routing other applications. The column entitled ***Type*** indicates whether the application is combinational (C) or sequential (S). The ***Array size*** column indicates the size of the MUXTREE array that contained the application. The following column contains the size of the multiplexer network, followed by the maximum number of generations that the genetic algorithm searched for the solution. The following column indicates in which generation the first solution was found, followed by the time taken to find it. Next column presents the number of valid solutions delivered by the genetic router when ran the maximum number of generations. The last column presents the probability of successfully routing the corresponding application, e.g. the genetic router will find a solution to the 3-bit up/down counter, 7 out of 10 times that the program runs 10,000 generations.

Table 3. Results obtained when routing other applications.

Application	Type	Array size	Mux-net size	Num. of gen.	Gen. of 1st sol.	Time to 1st sol.	Num. of solutions	Routing prob.
3-bit voter	C	2×2	3	100	1	0.5 seg	82	1
3-bit U/D counter	S	3×3	5	10000	250	1.2 min	85	0.7
4-bit CRC generator	C	2×6	10	10000	80	45 seg	78	0.5
2-bit complete adder	C	4×8	15	30000	8500	60 min	75	0.4
4-bit parity generator	C	4×8	15	30000	2500	25 min	80	0.9
5-bit parity generator	C	5×6	11	20000	450	8 min	76	0.2
Module-10 counter	S	4×5	15	30000	6500	60 min	80	0.5

5 Conclusions and Future Work

The genetic router presented in this paper is capable of mapping a given multiplexer network into a MUXTREE array. The router could also be applied to solve routing problems in networks presenting a topology similar to that of MUXTREE arrays.

The multiple solutions delivered by the genetic router open the possibility of a new level of redundancy in applications requiring high levels of availability. Different routings could be tried in the same hardware until functionality of the array is restored.

There still are questions regarding the capabilities of the genetic router that require further research. How efficient in terms of resource-usage are the solutions found by the router? What is the maximum number of multiplexers that the router can solve? How sensitive the router's performance is to changes in the GA parameters? How well the genetic router compares against other routing techniques? [12-15] Embryonics is a very vast field where much research remains to be done.

Acknowledgements

We like to thank the Institute of Electrical Research and CONACyT for the facilities and financial support given to carry out this research. Thanks to Dr. Luis Schettino and the reviewers for their valuable comments.

References

[1] Mange D., Sanchez E., Stauffer A., Tempesti G., Durand S., Marchal P. and Piguet C., "Embryonics: A new methodology for designing FPGAs with self-repair and self-reproducing properties", Technical report 95/152, EPFL, Logic Systems Laboratory, 1995

[2] Ortega C. and Tyrrell A., "Fault-tolerant Systems: The way Biology does it!", Proceedings Euromicro 97 (Short Contributions), Budapest, IEEE CS Press, September, 1997, pp.146-151

[3] Tempesti G., Mange D., Stauffer A.and Teuscher C., "The BioWall: an Electronic Tissue for Prototyping Bio-Inspired Systems", in A. Stoica et al. (Eds.), Proceedings of the 2002 NASA/DoD Conference on Evolvable Hardware, IEEE Computer Society, Los Alamitos, Calif., 2002, pp.221-230

[4] Restrepo H.and Mange D., "An Embryonics Implementation of a Self-Replicating Universal Turing Machine", in Y. Liu, K. Tanaka, M. Iwata, T. Higuchi, M. Yasunaga (Eds.), Evolvable Systems: From Biology to Hardware, ICES 2001, volume 2210 of Lecture Notes in Computer Science, 2001, pp.74-87

[5] Ortega C., Mange D., Smith S. and Tyrrell A., "Embryonics: A Bio-Inspired Cellular Architecture with Fault-Tolerant Properties", Genetic Programming and Evolvable Machines, Vol.1-3, July 2000, pp.187-215

[6] Tempesti G., Mange D. and Stauffer A., "A Robust Multiplexer-based FPGA Inspired by Biological Systems", Special Issue of Journal of Systems Architecture on Dependable Parallel Computer Systems, February 1997, pp.719-733

[7] Ortega C. and Tyrrell A., "MUXTREE revisited: Embryonics as a Reconfiguration Strategy in Fault-Tolerant Processor Arrays", Proceedings of ICES98, Lausanne, Switzerland, September, 1998, Lecture Notes in Computer Science 1478, Springer-Verlag, 1998, pp.206-217

[8] Ortega C. and Tyrrell A., "Design of a Basic Cell to Construct Embryonic Arrays", IEE Transactions on Computers and Digital Techniques, Vol.145-3, May, 1998, pp.242-248

[9] Ortega C. and Tyrrell A., "Reliability Analysis in Self-Repairing Embryonic Systems", in Stoica A., Keymeulen D. and Lohn J. (Eds.), Procs. of 1st NASA/DoD Workshop on Evolvable Hardware, Pasadena, CA, IEEE Computer Society, July 1999, pp.120-128

[10] Holland J., Adaptation in Natural and Artificial Systems, MIT Press, 1992

[11] Goldberg D., Genetic Algorithms in Search, Optimization and Machine Learning, Addison-Wesley, ISBN: 0201157675, 1989

[12] Drechsler, Evolutionary Algorithms in VLSI CAD, Kluwer, 1998

[13] Minato Shin-Ichi, Binary Decision Diagrams and Applications for Vlsi CAD,Kluwer International Series in Engineering and Computer Science, 342, 1996

[14] Bushnell Michael Lee, Design Automation: Automated Full-Custom Vlsi Layout Using the Ulysses Design Environment, Perspectives in Computing, Vol 21, 1988

[15] Cheng Chung-Kuan (Editor), Interconnect Analysis and Synthesis, Wiley-Interscience, October 1999