A Reconfigurable Self-healing Embryonic Cell Architecture*

X. Zhang, G. Dragffy, A. G. Pipe, N. Gunton and Q. M. Zhu
Faculty of Computer Science, Engineering and Mathematics Science
University of the West of England, Bristol, United Kingdom

Abstract

The long-term goal of our work described in this paper is the development of a biologically-inspired cellular fault-tolerant hardware system. The basic structure of the system is a multi-cellular embryonic array that is capable of achieving self-diagnostics, self-repair and fault recovery. The 'nucleolus' of the cell is a general purpose function unit comprised of a 2-to-1 multiplexer and a D-type flip-flop. Input and configuration data to the function unit are provided by the DNA segment memory and I/O router. A diagnostic logic monitors the error free operation of the cell. When an error is detected the diagnostic logic requests the reconfiguration unit to kill the cell, transferring its function to a fault-free neighbouring cell. Once permission is granted, the faulty cell is eliminated and becomes transparent. The functionality of each cell will shift in the array until a spare cell is found. Finally the whole embryonic array recovers. Fault-free operation is then continued.

1. Introduction

Nature offers some remarkable examples of how it deals with complexity and unreliability. For example, the human body is one of the most complex systems ever known. Local failures are common, but the overall function of our organism is highly reliable. This is because of the highly sophisticated self-diagnostic and self-healing mechanisms that work ceaselessly throughout our body. If we apply the main biological principles, which provide these mechanisms, to the design of electronic systems, the result is a revolutionary new approach to implementing fault-tolerant systems.

Embryonics (Embryology & Electronics) [1] is one such biologically inspired approach to guarantee fault-tolerant dependable hardware systems. A multi-cellular biological system is developed from the zygote, a single fertilised cell; the process is called embryonic development. Embryonics is inspired by the embryonic process. By adopting certain features of cellular behaviour and organisation, and by transposing them to the two-dimensional world of integrated circuits on silicon, embryonics shows that properties found in the living world, such as self-reproduction and self-repair, can also be applied to integrated circuits.

In this paper, we will discuss a new embryonic model illustrating a process of cell apoptosis and replication when an error is detected in its functionality.

2. Embryonics

Currently there are two main groups of researchers in the field of embryonics: University of York, England [2], [3] and Swiss Federal Institute of Technology, Switzerland [1].

The work of both groups has been inspired by molecular cell biology and therefore their approaches show similar characteristics in both cases. The fundamental building block of their biologically inspired systems is a hardware embryonic cell. The idea of being able to replace damaged cells on the silicon with healthy ones when they die has led to the same conclusion. Unlike in the biological world, cells –in order to take over the functionality of those which die- cannot reproduce. The limitations of our current technology will dictate the avenues open to us. Defective devices and the faulty cells, which they occupy, cannot be repaired on the silicon. Nor can new cells be grown to replace the ones that died. However cell redundancy is an acceptable alternative solution. This then leads to the conclusion that repair mechanisms at the system level can only be accomplished if the faulty cell is removed and replaced by a spare one. This is achieved by making the cell transparent, as far as all its neighbouring cells are concerned, and by transferring its functionality is to a healthy stand-by cell.

In a generic embryonic model, the array consists of a number of identical cells which are linked together so as to create the functionality of the organism. Applying the world of biology to that of electronics guarantees the three fundamental characteristics of all living things: multi-cellular organisation, cellular division, and cellular differentiation [1], [4].

*: This paper is published in the proceedings of the International Conference on Engineering of Reconfigurable System and Algorithms (ERSA'03), pp.134-140, June 2003, USA.

Multi-cellular organisation is achieved by forming an organism from an array of programmable cells. At the time of initialisation, all of the cells are identical. C*ellular division* is achieved when the array is programmed. A mother cell (whose position can be defined as (0,0)) propagates the genome to its neighbouring cells creating two daughter cells until all the cells in the array have been programmed. In this generic model every one of the cells contains an entire copy of the organism's genome. Finally *cellular differentiation* is provided by the cell's position within the array, which defines which gene will be interpreted by this cell.

In order to satisfy the three fundamental biological characteristics of all living things the embryonic cell that we are proposing needs five functional elements: the DNA segment memory, co-ordinate generator, function unit, diagnostic logic and the I/O router [1] [2]. However, the repair of DNA memory faults, monitoring via error detection the cell's functional integrity, and subsequent system reconfiguration presents further requirements. Thus, our final embryonic cell is comprised of eight functional elements.

3. Mode of a Self-healing Embryonic Cell

The embryonic cell that we are proposing is developed from the University of York model [4]. Each cell (Figure 1) contains eight functional units: DNA segment memory, DNA-repair unit, function unit, diagnostic logic, I/O router, co-ordinate generator, reconfiguration unit and a core register.

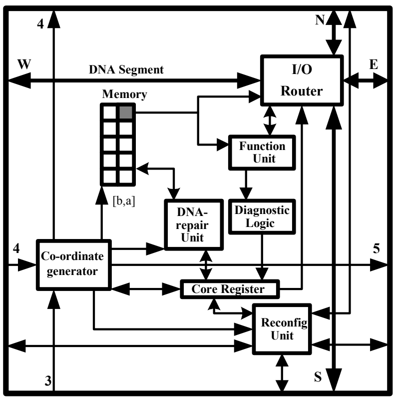


Figure 1. Model of a self-healing embryonic cell.

The co-ordinate generator calculates both the pointer's address for the DNA segment memory and the cell's own address in the array. The latter is dependent on its relative position with respect to its neighbouring cells. The memory pointer is used to select one set of configuration data (gene) from a segment of the system's DNA. The I/O router forwards to the function unit three input signals from the four neighbouring cells and will connect its output to one of those cells. The function unit contains a 2:1 multiplexer-based universal logic element and a D-type flip-flop. Any error in the function unit is detected by the diagnostic logic. The DNA-repair unit detects and repairs any transient error in the DNA segment memory. Finally, depending on the status of the core register, the reconfiguration unit may kill the cell and reconfigure the array so that the faulty cell is replaced by a fully functional one.

3.1. Function Unit

The function unit is the 'heart', the 'nucleolus' of the embryonic cell. It contains a 2:1 multiplexer and a D-type flip-flop. The former acts as a universal logic gate for the implementation of any kind of combinational function, while the latter aids the realisation of sequential functionality. The actual configuration of these elements, in order to achieve required operation is controlled by the configuration data in the DNA segment memory. This will determine which neighbouring cells' output will be connected via the I/O router to the A, B and X inputs and which neighbouring cell will receive the E output of the function unit. An appropriate bit in the DNA segment memory also determines the status of the SEQ input. If only combinational and no sequential operation is required from the cell then the SEQ input is set to "0" and the D-type flip-flop is bypassed. Otherwise the SEQ input will be set to "1" and the cell will support sequential mode of operation.

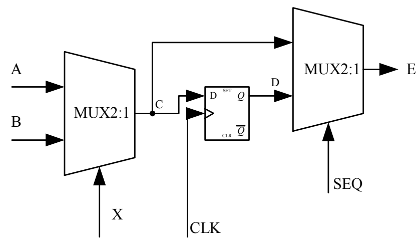


Figure 2. Function Unit.

3.2. Diagnostic Logic

An error in the function unit will be detected by Built-In-Self-Test (BIST) logic. In essence, it uses an Exclusive-OR gate to compare the output of the function unit with its duplicate. If an error is detected, the BIST logic will set an error flag in the core register. This in turn will signal a cell-elimination request to the reconfiguration unit.

3.3. DNA Segment Memory

The gene, in the form of configuration data that our proposed embryonic cell requires, is held in the DNA segment memory. The gene of each cell will define the direction of its input/output, will determine the specific behaviour of the function unit. It will also enable or bypass the D-type flip-flop as required and determine the direction of the cell's output signal. In order to satisfy all these requirements, the storage of each gene within the DNA segment memory will require a 13-bit register.

Each embryonic cell has an operational mode (figure 3) and a bypass mode (figure 4). A control bit 'BP' will determine which mode is selected. If 'BP' is equal to 0, the cell will function in the operation mode, otherwise it will work in the bypass mode.

In operation mode, the function unit receives its inputs via the I/O router as supplied by the four neighbouring cells or they can be set to logic zero or logic one. The I/O router can also determine which neighbouring cells' input should be connected to the cell's output. This unit is also under the control of the DNA segment memory's gene.

In bypass mode, the cell will transfer signals from two neighbouring cells to two other neighbouring cells. The cell's function unit is non-operational and signals are simply routed through the cell.

Operation Mode:

SEQ	OUT1	OUT0	IN5	IN4	SET2	IN3	IN2	SET1	IN1	IN0	SET0	BP

Figure 3. The Operation Mode of a gene (BP = 0).

Bypass Mode:

X	OUT1	OUT0	OUT3	OUT2	SET2	IN3	IN2	SET1	IN1	IN0	SET0	BP

Figure 4. The Bypass Mode of a gene (BP = 1).

3.4. Co-ordinate Generator

The co-ordinate generator (figure 5) normally has a dual function.

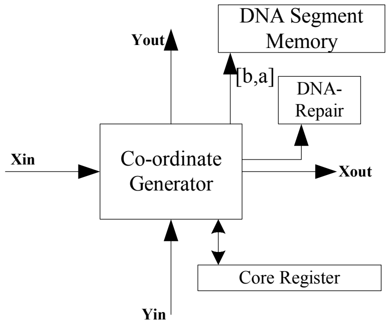


Figure 5. Co-ordinate generator and its interconnect.

Its *first* function is to calculate an address from the signals received from the cells of the array on its left and below, and to pass this new address to the cells on its right (formula 3.1) and above (formula 3.2). Its *second* function is to calculate which location of the DNA segment's memory the pointer should be addressing. After cell reconfiguration, the co-ordinate generator will recalculate the new addresses of the East (left) and North (above) cells and their pointer's address.

$$X_{out} = X_{in} + 1 \qquad (3.1)$$
$$Y_{out} = Y_{in} + 1 \qquad (3.2)$$

In our proposed embryonic cell the pointer's address (b, a) of the gene in the DNA segment, will be calculated by the co-ordinate generator. If at the initial stage there are A spare columns and B spare rows, then the co-ordinate generator's output pointer address is (B, A).

3.5 I/O Router

During normal operation, the I/O router will connect the cell to all four of its neighbouring cells. It also selects, from three neighbouring cells, three input signals for the function unit and transmits the output signal of the cell's function unit to one of its neighbours. Cell reconfiguration is in two steps. *First*, faulty cells will be replaced by spare ones located in same row. *Second*, when the number of faulty cells in a row exceeds the number of available spare cells, then the whole row is eliminated. The eliminated cells or rows will be transparent to their neighbouring cells. Since the transparent cells only use simple combinational logic to connect the data channels from the lower to the upper cell or from the left to the right cell, we can, for the moment, assume that this routing is fault-free. Cell elimination strategy requires three channels in a row for row-wise cell to cell communication and only one channel for column-wise communication (figure 6).

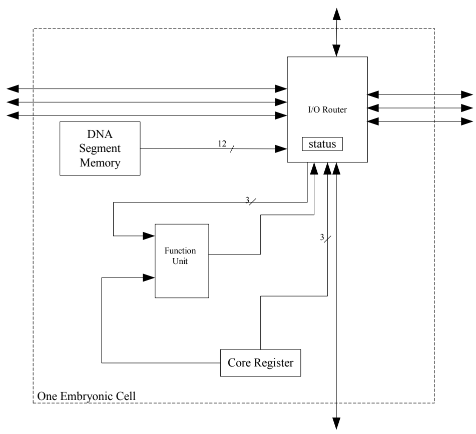


Figure 6. I/O router and its interconnect.

3.6 Core Register

In order to satisfy the requirements of both the DNA repair and cell-elimination functions, two status signals (OK or Not-OK) are no longer sufficient. The requirement is for a more sophisticated 14-bit wide core register that will also store status signals as well as request signals for cell-elimination. (Figure 7).

Core Register:

CF	FR	LR	FC	LC	PMT	RE1	RE0	CE1	CE0	DR1	DR0	SP	ER

Figure 7. Core register content.

Table 1. Core Register Content Definition:

Field	Bits	Description
ER	0	*Cell's error status* 0 OK 1 Error
SP	1	*Cell function status* 0 Spare cell 1 Working cell
DR0	2	*DNA-repair requirement* 0 The normal status 1 The DNA-repair required
DR1	3	*DNA-repair flag* 0 The normal status 1 DNA-repair is finished
CE0	4	*Cell-elimination requirement* 0 The normal status 1 Cell-elimination required
CE1	5	*Cell-elimination flag* 0 The normal status 1 Cell-elimination completed
RE0	6	*Row-elimination requirement* 0 The normal status 1 Row-elimination required
RE1	7	*Row-elimination flag* 0 The normal status 1 Row-elimination completed
PMT	8	*Permission flag of reconfiguration* 0 Reconfiguration is forbidden 1 Reconfiguration is available
LC	9	*Last cell flag* 0 This cell is not the last cell in a row 1 This cell is the last cell in a row
FC	10	*First cell flag* 0 This cell is not the first cell in a row 1 This cell is the first cell in a row
LR	11	*Last row flag* 0 This cell is not in the last row of the array 1 This cell is in the last row of the array
FR	12	*First row flag* 0 This cell is not in the first row of the array 1 This cell is in the first row of the array
CF	13	*System error flag* 0 Normal status 1 System can no longer be repaired

The 14-bit core register supports both the DNA-repair and cell reconfiguration mechanisms, thus no external support for these operations is required.

3.7 DNA-repair Unit

The DNA-repair function is inspired by the biological DNA-repair mechanism [5]. The DNA-repair unit will detect any error in the gene, and in the co-ordinate generator. If an error is found, it will set the error flag in the core register. But if the error it finds is a result of gene mutation then a backup gene is used for the repair. This operation can only be carried out once to prevent a possible endless loop of futile repairs. In this case the error is possibly permanent and non-repairable. If a further error in the gene is detected, then in the core register, the DNA-repair unit will set an error flag. In this case, control of the cell is handed over to the reconfiguration unit that, via cell-elimination, will restore the fault-free functionality of the embryonic array.

3.8 Reconfiguration Unit

When an unrecoverable error inside the embryonic cell is detected, then the cell must be killed and the array is reconfigured. Moreover, the type of reconfiguration must also be decided. There is a clear case therefore to include another functional module within the cell in the form of a reconfiguration unit. Since the task expected is demanding it should ideally be undertaken by the cell itself rather than entrusting this to some external hardware. The reconfiguration unit, with the aid of the core register that monitors the cell's status, is doing exactly that.

3.9 Functional behaviour of the Embryonic cell

Figure 8 illustrates the behaviour of an embryonic cell.

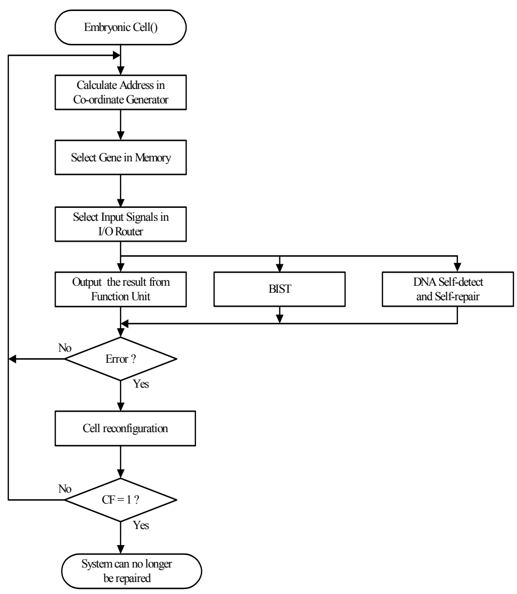


Figure 8. Functionality of the embryonic cell.

In its normal mode of operation, the gene in the DNA segment memory determines the functionality of the embryonic cell. If content mutation in the gene's memory is detected, the mechanism will correct this error. If, however, the error in either the gene's memory or in the function unit persists, then the embryonic cell will kill itself. Only one DNA-repair is allowed. The array under the autonomous control of the cell will reconfigure the array, whereby the functionality of the faulty cell will be mapped onto the live and spare cells of the array. The system is stable again and fault-free operation is resumed.

4. Reconfiguration using cell-elimination

When the diagnostic logic detects a fault in any one of the cells a reconfiguration mechanism is triggered and the cell will carry on functioning correctly. To achieve this, various cell-elimination methodologies have been studied.

This paper will demonstrate a cell-elimination strategy, where only a segment of the organism's DNA will be present in any one of the cells. Two co-ordinates will define the exact position of the embryonic cell in the array. For reconfiguration purposes its configuration register will only have as much memory as required by the most exhaustive reconfiguration the cell will ever experience. Unlike living things, for every cell to have a full copy of the DNA is an unnecessary overhead. No other gene but that of the cell concerned, and those of the adjacent ones, which as a result of reconfiguration may overtake its functionality, will ever be required.

During cell-elimination, spare cells replace faulty cells in two stages. *First*: spares located in the same row replace faulty cells. *Second*: when the number of faulty cells in a row exceeds the number of spare cells assigned to that row, then the whole row is eliminated and cells are logically shifted upwards (row-elimination). In this case a spare row will take over the functionality of the faulty one. Figure 9 shows a cell elimination example in an array that has one spare column and one spare row.

4.1 Reconfiguration Unit

The strategy that we applied to cell-elimination is that when any one of the cells in the array needs to be killed, the reconfiguration unit will at first always initiate cell-elimination. If this is no longer permitted (no more spare cells in a row), then row-elimination will be executed. When the cell's error flag in the Core Register is set to '1', only the type of reconfiguration indicated by the status of the cell will take place. If the cell has done no cell-elimination previously, its reconfiguration unit will send a cell-elimination requirement (bit 'CE0' in the Core Register) to its right neighbour which in turn will be propagated to the last cell in the row. If the last cell is a spare cell, it will signal permission (bit 'PMT' in the Core Register) to its left neighbour that will be propagated back until the faulty cell is reached (figure 10). The address of the faulty cell will be recalculated (formula 4.1.1, 4.1.2) and the cell thus killed will become transparent.

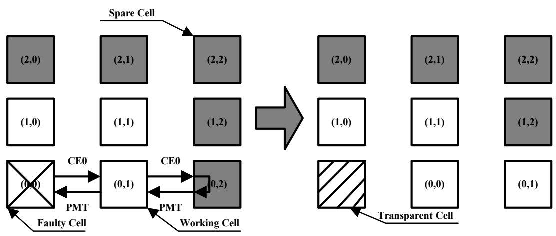


Figure 10. Stage 1: Cell-Elimination.

$$X_{out} = X_{in} \qquad (4.1.1)$$
$$Y_{out} = Y_{in} + 1 \qquad (4.1.2)$$

If the last cell is found to be a working cell, then it will transmit a row-elimination signal (bit 'RE0' in the Core Register) to its left neighbour. This will be propagated all the way back to the first cell in this row. Every cell in the row will now go into a row-elimination ('RE0') mode and will signal this requirement to their neighbour above. The signal is then propagated to the last cell in the column. If

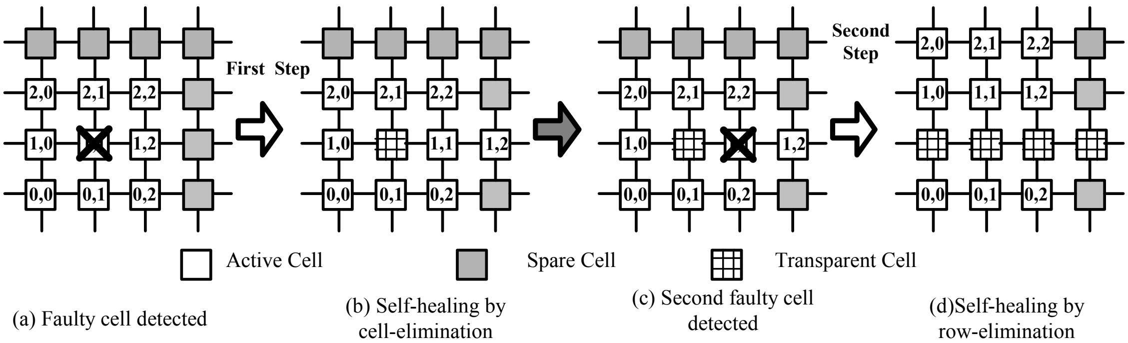


Figure 9. The strategy of cell-elimination in an embryonic array.

this is a cell of the spare row of cells, then it will return a row-elimination permitted (PMT) signal to its neighbour below. This will be returned to the row that requested row-elimination. The cell's addresses of the row will be updated (formula 4.2.1 and 4.2.2) and the entire row will also be eliminated (figure 11).

$$X_{out} = X_{in} \quad (4.2.1)$$
$$Y_{out} = Y_{in} \quad (4.2.2)$$

Cell reconfiguration can also be monitored externally and, depending on the number of spares rows the system uses, different levels of operational alarm can be set.

During cell-elimination, the co-ordinate generator of the faulty cell will recalculate (formula 4.3.1 and 4.3.2) the pointer address (b, a) for its DNA segment memory that that the gene requires. In the formula i and j represent the number of times respectively, that cell- and row-elimination has already occurred.

$$a = A - i \quad (4.3.1)$$
$$b = B - j \quad (4.3.2)$$

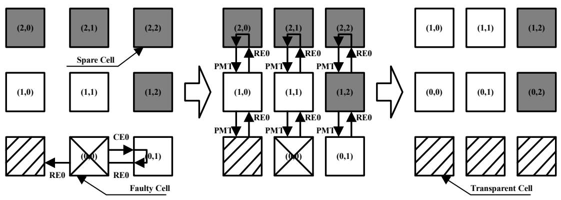
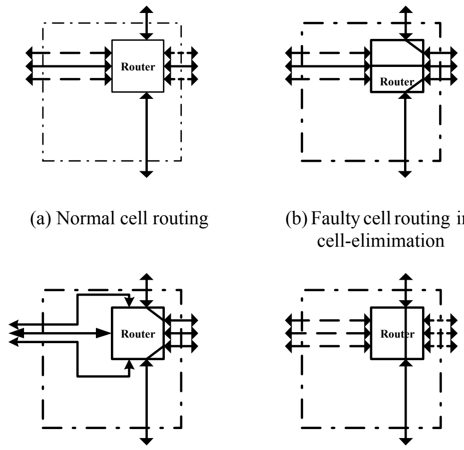


Figure 11. Stage 2: Row-Elimination

4.2 Cell Routing

Restriction of the range of possible cell routing will keep reconfiguration time to the minimum. During the first stage of cell-elimination, with spare cells located on the right-hand side of the array, only the faulty cell and its right neighbour need to be reconfigured. No other cells are affected. In this case, three different types of cell routing are possible: *normal* cell routing (figure 12 a), *faulty* cell routing (figure 12 b) and *modified* cell routing (figure 12 c). In the latter case the cell is on the right-hand side of the faulty one. During row-elimination the faulty cell is transparent and the signal is passed through it (figure 12 d).

When a cell is eliminated, its genome and thus the functionality of the cell must be retained in the 'new' cell it now occupies. Additional inter-cell communication channels are required for the row to pass this information on. If, for example, the row uses one spare cell the number of inter-cell communication channels it will have, is three. The *upper* channel will pass the data from the north cell to the east cell. The middle channel carries data from the left cell to the right cell. The *lower* channel will pass the data from the south cell to east cell. Figures 12.b, c and 13 show all possible cell routing.



(a) Normal cell routing

(b) Faulty cell routing in cell-elimination

(c) Modfied cell routing in cell-elimination

(d) Faulty cell routing in an eliminated row

Figure 12. Types of cell routing

The number of inter-cell communication channels required will depend on the number of spare cells available for a row. If the number of spare cells is n then the row must have N channels (formula 4.4).

$$N = 2n + 1 \quad (4.4)$$

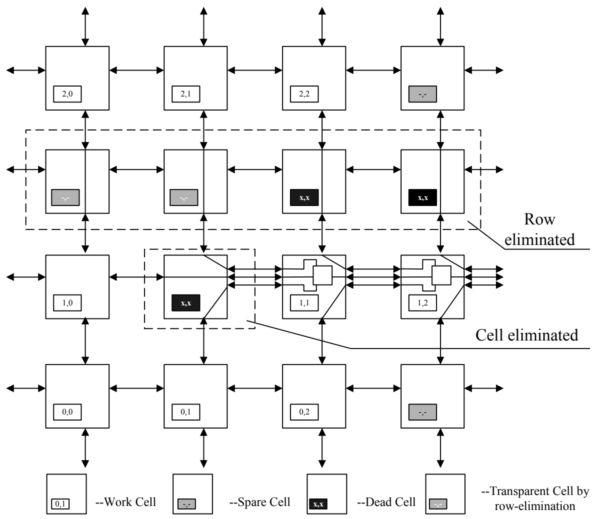


Figure 13. Possible routing of cells.

5. Conclusion and further work

In the presence of a faulty cell for self-reconfiguration our proposed embryonic cell uses a cell-elimination strategy. This reconfiguration is inspired by the biological world, where in a living thing failures are common, but the overall function of the biological organism is highly reliable because of the self-diagnosis and self-healing mechanisms that work ceaselessly throughout the organism.

Cell-elimination is one of the possible methodologies to reconfigure an embryonic cell array. Individual row-elimination and column-elimination [4] [6] can also be used for reconfiguration purposes. It might reduce the complexity of the embryonic cell, but it is also wasteful of spare cells. In such cases, an entire row and/or column of cells will be eliminated by the presence of just one faulty cell. So, the question of finding a fine balance between the efficient use of spare cells and the reconfiguration strategy used is an important area for any future work.

6. References

[1] Gianluca Tempesti: PhD thesis ‘A Self-Repairing Multiplexer-Based FPGA Inspired by Biological Process’, Logic Systems Laboratory, Swiss Federal Institute of Technology, Lausanne, Switzerland, 1998.

[2] Cesar Ortega-Sanchez and Andrew Tyrrell, Fault-tolerant Systems: The way Biology does it! Proceedings Euromicro 97, EIEEE CS Press, Budapest, pages 146-151, Sep. 1997.

[3] Cesar Ortega-Sanchez and Andrew Tyrrell, Biologically Inspired Reconfigurable Hardware for Dependable Applications, IEE Colloquium on Hardware Systems for Dependable Applications, Digest No:97/335, Lodon.Nov. 1997.

[4] Cesar Ortega-Sanchez and Andrew Tyrrell, Design of a Basic cell to Construct Embryonic Arrays, IEE Transactions on Computers and Digitial Techniques, Vol.145-3, pp.242-248, May, 1998.

[5] Harvey Lodish, Molecular Cell Biology, third edition, pp.365-404, New York, N. Y.: Scientific American Books, 1995, ISBN: 0716723808.

[6] R. O. Canham and A.M. Tyrrell: A Multilayered Immune System for Hardware Fault Tolerance within an Embryonic Array, in proceedings of 1st International conference on Artificial Immune System, pp.3-11, September, 2002