Design of a self-assembling, repairing and reconfiguring Arithmetic Logic Unit

David Huw Jones, Richard McWilliam and Alan Purvis
University of Durham
England

1. Introduction

The consumer will not tolerate products that are prone to failure: cars such as the Skoda Felicia or the Ford Pinto, motorbikes such as the BSA Dandy scooter, computers such as the Apple III - these devices were all commercially unsuccessful simply because of their high failure rate. For a product to succeed, the designer must consider the intended lifespan of the device and then maximise the probability of its survival for that period.

Within the systems upon which we depend, or choose to trust, for our survival - be they aeroplanes, trains, armaments or systems critical to a country's infrastructure - failure cannot be tolerated without adequate notice and provision for repair or replacement. 500 people died when Pfizer's replacement heart valves failed, costing the company upwards of $200 million. The structural failures of the de Havilland Comet jet aeroplane caused five crashes before commercial flights were cancelled. A designer must therefore consider the significance of a component failure and make provisions for repair or replacement.

Components that are too costly to repair or replace, because of limited accessibility or limited resources, cannot be subject to failure. The early operational failures of the Hubble telescope were due to a flawed mirror. A space mission costing $8 million was required to fix the telescope with correcting lenses. Thus for a system to succeed, a designer must consider the cost of repair when deciding on the required level of reliability.

The trend is for devices to be smaller and more functional than their predecessors. Modular redundancy, the use of extra copies of failure-prone hardware that can mask, or take the place of its damaged counterparts, has costs in both size and functionality. If this trend continues, the price of modular redundancy will become greater and an alternative will need to be considered.

Aside from demanding applications, reliability engineers face a challenge from systems formed on unreliable mediums. Plastic electronics systems are formed on flexible substrates, typically by the sequential deposition of conductive or semi-conductive organic polymers (such as Poly-3,4-ethylenedioxythiophene) using analogue or digital graphic printing technologies. Potential applications include flexible displays, photovoltaic's that can cover non-planar surfaces, and smart packaging, including battery testers, flexible batteries and RFID tags. However, flexible substrates are prone to distortion, potentially compromising the deposited system and causing its failure. Redundancy is particularly well equipped to

cope with the failure of known unreliable sub-systems, but is much less well suited to coping with a system of which any part, or combination of parts, is likely to fail.
Since 2002 NASA have been running a series of workshops under the title "Ultra reliability"; this with the goal of increasing systems reliability by an order of magnitude across complex systems, hardware (including aircraft, aerospace craft and launch vehicles), software, human interactions, long life missions, infrastructure development, and cross cutting technologies (Shapiro, 2006). It is not difficult to see the difficulties standard redundancy techniques will face in long life missions (a manned mission to Mars will take upwards of six months, the $720 million Mars Reconnaissance Orbiter has a planned life of at least four years) that are vulnerable to cosmic rays.

2. Morphogenesis-inspired ultra reliability

Morphogenesis provides biological systems with a robust framework for the differentiation of undeveloped or partially developed cells. Remarkable examples of self-repair co-ordinated by morphogenesis include:

- The human liver: This organ is capable of withstanding and repairing damage to up to two-thirds of its constituent cells.
- The Salamander: If bisected from its tail, the tail will often grow back.
- Ascidians (marine filter feeders), whose blood cells alone have been reported to give rise to a fully functional organism (Berrill and Cohen, 1936).

Morphogens are soluble proteins that diffuse about source cells within developing tissue. These chemical messages co-ordinate the differentiation of cells, determining what type of cell belongs where in the tissue. This self-assembly, self-repair mechanism can be modelled on cellular automata.
Cellular automata (CA) are systems in which space and time are discrete. CA consist of an array of cells, each of which determines its next state (typically an integer or a colour) from the state of its immediate neighbours using a next-state rule that is common to every cell. The analysis presented in (Jones et al., 2008) showed that for the array of cells to converge to one particular pattern of states regardless of its initial conditions, the next-state rule cannot use its current state as an input, nor can it use the state of more than one neighbour per axis as an input. That is, if the array is two-dimensional, in determining its next state a cell can only use inputs from neighbours either above or below itself, either to the left or to the right of itself. The paper goes on to demonstrate a deterministic algorithm for the design of CA that converge to a specific pattern of states.
If each state maps to a particular logic function, the CA will converge toward a pattern of logic units that could ultimately form any effective electronic system. Such a CA will converge to this form regardless of its initial conditions, effectively making it intrinsically self-repairing. Furthermore, if the boundary conditions of this CA are altered the CA will converge to a different pattern. If this is considered during the design process, the CA becomes able to self-reconfigure.
To be of any use, a reliability mechanism must be less prone to failure than that which it is trying to protect. However, as each cell must be identical, achieving this homogeneity for large system blocks adds a significant cost in replicated function to the design. Thus the optimum scale at which to discretise the design into a cellular architecture is found at the minima of total system complexity (see figure 1).

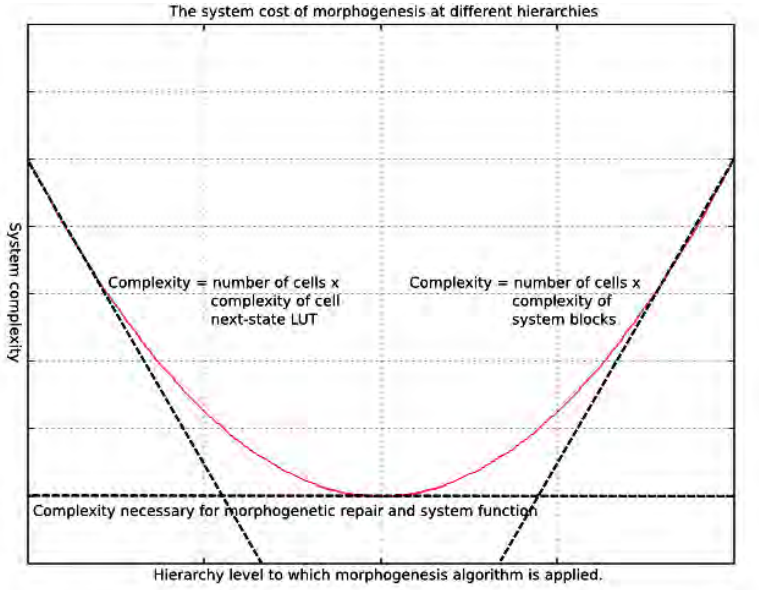


Fig. 1. The system cost of morphogenesis at different hierarchies

Existing reprogrammable devices (PLA, PAL or FPGA) are not optimised for the fine-grained self-reprogrammable logic required for this self-assembling algorithm. An appropriate custom Application Specific Integrated Circuit (ASIC) could be optimised to use fewer units of logic than an equivalent FPGA implementation. One possible embodiment of a cell within an ASIC is shown in figure 2.

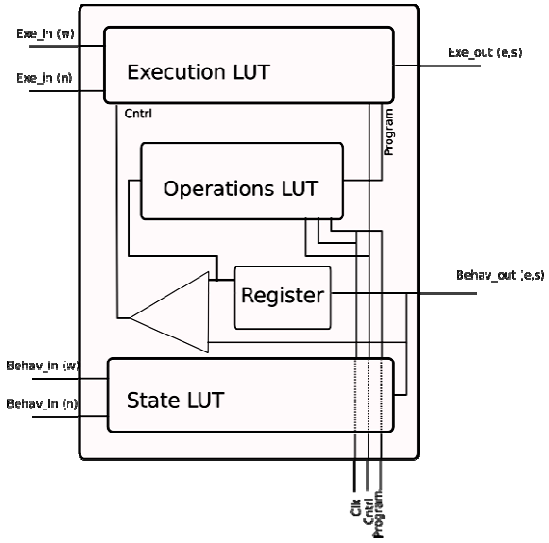


Fig. 2. An ASIC implementation

The state of each cell must map to a component function, coded in the form of a bitstream that can be written to a look-up table and executed. Every bitstream required for the automata is stored in the function look-up table. The bitstream is selected by the cell state, loaded into the execution look-up table and executed. Every time the cell configuration changes (as detected by the cell comparator), the execution look-up table is reloaded with a new bitstream.
Another consideration is the use of redundant cells. Biological implementations of morphogenesis have an advantage over any electronic implementations: in the event of a cell being permanently damaged, biology can grow a replacement. This is something that is currently not possible in electronic devices. Thus, redundant cells - cells that can take the place of any other in the event of permanent failure - must be an integral part of the design. In order for a redundant cell to take the place of any damaged cell, every cell would have to be directly connected to every other cell. An alternative is to place every cell on a shared bus and provide each cell with an appropriate interface. Dynamic, in-situ re-programmable routing is another possibility. There are already various algorithms (Thoma, 2003) for managing dynamic routing.

3. Design of a self-assembling self-repairing one-bit full-adder

The schematic for a one-bit full-adder can be seen in figure 3.

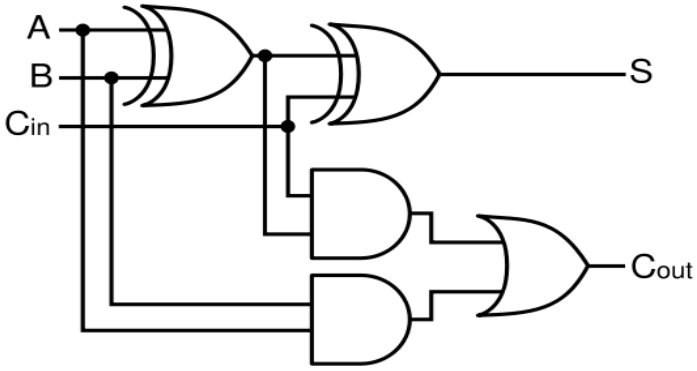


Fig. 3. A 1-bit full-adder schematic

In order to minimise the component count (and therefore the number of components that contribute to the device failure rate) there are a few constraints to the schematic design:

- Every cell determines its state from two of its immediate neighbours. Other than power and clock lines there are no global connections required for the CA to converge to its correct state. If a full-adder is to be implemented on such a platform, it would ideally not require any global connections either.
- The cell has no bi-directional communications: it relies on two-inputs and two-output lines in a feed-forward arrangement in order to converge. Likewise an ideal full-adder design should be built on this arrangement. This means each component cell cannot feedback data to a cell that lies earlier on on the data path.

- Each cell has two output lines, but the state-output is common to both. Again, the most appropriate implementation of a full-adder design will piggy-back these existing communications lines and not require additional networking. One consequence of this is that no two data lines can cross.
- Because we want this full-adder to be scalable, the one-bit full-adder modules should be stackable, that is, if the modules are arranged one on top of one another, the carry-out lines should connect to the carry-in line beneath it.
- In order for the full-adder to be scalable, the CA state pattern must repeat until it uses all the available cells.
- There should be as few different cell-types (equivalent to the size of the CA alphabet) as is necessary, and there should be as few cells per one-bit module as is necessary.

Figure 4 shows how the schematic of figure 1 has been revised to ensure there are no crossed data lines. Figure 5 shows the different cell operations and their corresponding state assignments. Figure 6 shows the schematic laid out over 16 cells. Figure 7 shows the design (implemented on an ALTERA FPGA) self-assembling. Note that this design requires the following boundary conditions:

- The cell connected to input 'A' of the full-adder must be to below a state '7'.
- The cell connected to input 'B' of the full-adder must be to the right of a state '2'.
- The top-left cell of the first bit of the full-adder must be below a state '1'.
- The cell to the right of the top-left cell of the first bit must be below a state '2'. Since the bottom row of each 16 cell design starts with the states '1' and '2', these boundary conditions propagate to the subsequent bits and the design repeats until it runs out of cells.

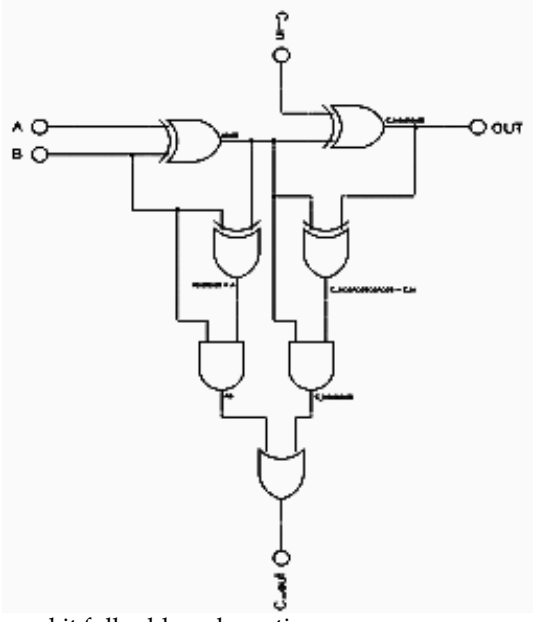


Fig. 4. An alternative one-bit full-adder schematic

Cell function	OR		XOR		XOR/ AND
Cell states	0	1,7	2	3,4	5

Fig. 5. Different types of cell

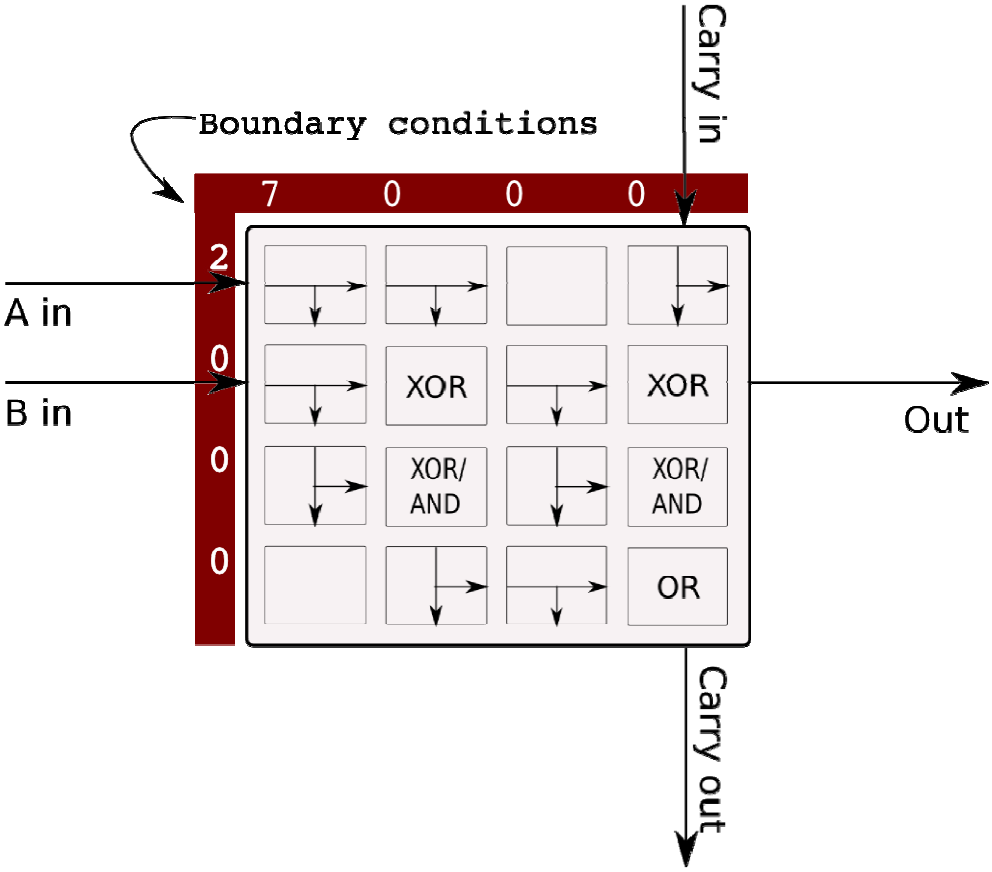


Fig. 6. A one-bit full-adder layout

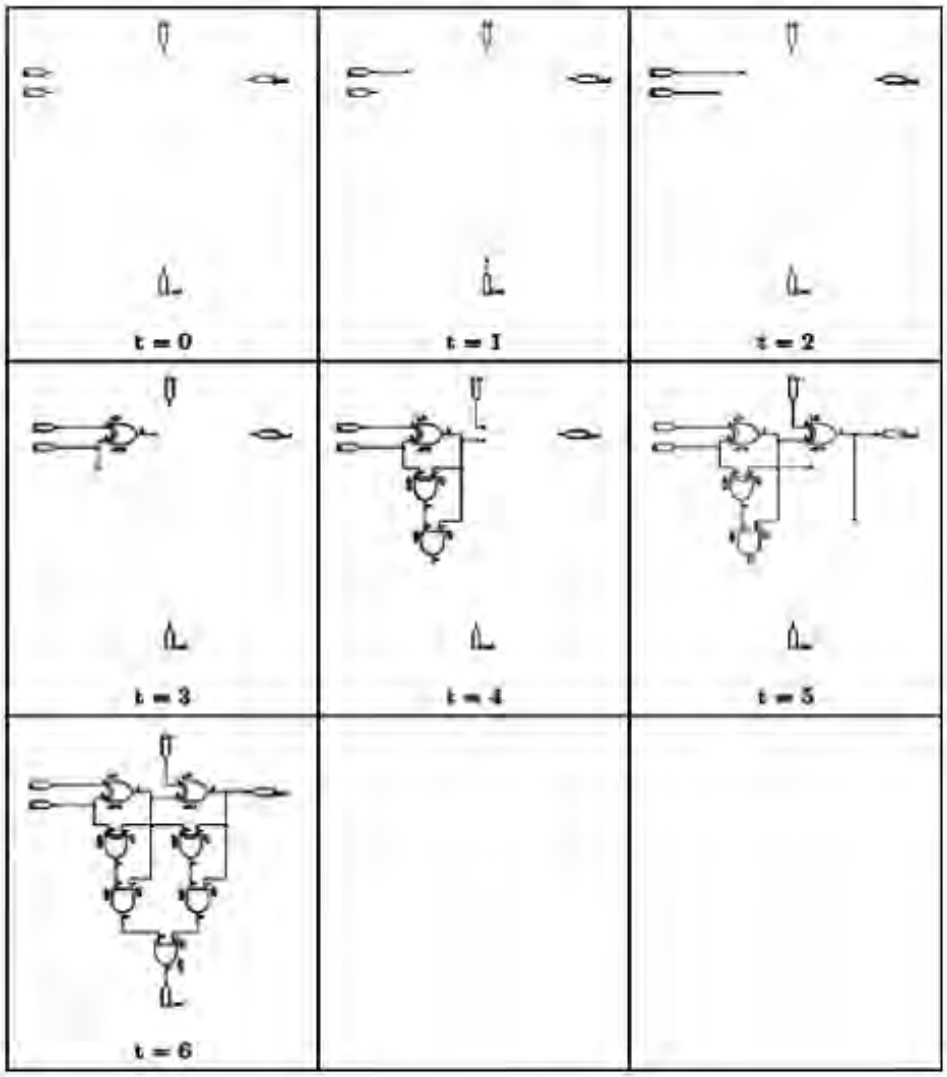


Fig. 8. A one-bit full-adder self-assembling

4. Design of a self-assembling self-repairing one-bit Arithmetic Logic Unit

In order for the full-adder design to correctly self-assemble, the boundary conditions of the array must be precisely set. If these are changed, the arrangement of cell types will change. This effect can be taken advantage of by designing the cell array to respond to changes in the boundary conditions with desired alternative arrangements. Thus, the full-adder could be converted into a full-subtracter by changing one of the boundary conditions. Likewise, the

array can be programmed to perform other functions of an arithmetic logic unit (ALU) (e.g. AND, OR and NOT gates) with different boundary conditions.
Since we want the design to scale, these boundary condition changes need to propagate to the bottom of the 16 cell arrangement so that the subsequent 16 cells can also re-configure to perform the requested function. This requirement is responsible for some of the more esoteric logic arrangements (for instance a NOT gate being built from two XOR gates and a NOT gate) present in the designs. Figure 9 shows the logic arrangements and boundary conditions for the ALU functions, AND, OR, NOT and SUBTRACT.

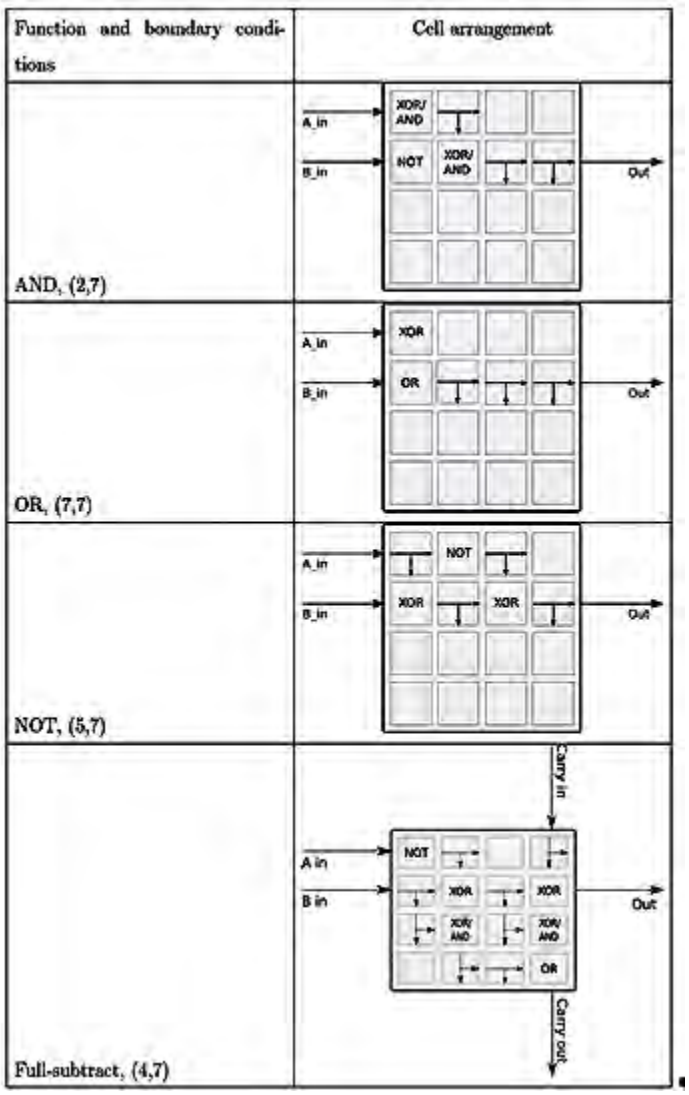


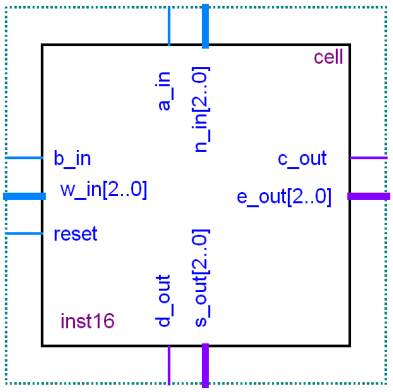
Fig. 9. Cell arrangements and boundary conditions for ALU

5. Assessing the reliability of the ALU

The mechanisms of CA and morphogenesis require a homogenous array of 'stem' cells that are capable of becoming any type of cell during system development and repair. The LUT of FPGAs are conceptually just such a cell, but while some FPGAs are capable of partitioning and reprogramming a small portion of their entire configuration, the bit-stream configuration data must be provided from a source external to the FPGA. This means that without a significant re-design of the FPGA infrastructure, the application of mitosis (replication of cell function) is not possible. As a result it is no longer enough for each cell to be capable of becoming any other type of cell; instead, each cell must be capable of performing the function of any other type of cell without any in-situ re-programming.
In order to assess the reliability of this self-repairing ALU and compare it to the reliability of a standard ALU design and an N-modular redundant ALU design, each will be implemented on a field programmable gate array (FPGA). The number of logic blocks used by each component of each design and the systems dependence on each will be used with the reliability analysis techniques described in the previous chapter in order to assess the MTBF of both.
There are a number of assumptions in this analysis:

- The FPGA is an ALTERA Stratix II, the design software is QUARTUS. The fitting process used by Quartus to convert the VHDL design into a netlist will not necessarily create a netlist that uses the fewest number of logic units. As the self-repairing ALU is more complicated than a standard ALU, if the conversion does not create the smallest design possible the self-repairing ALU will be affected to a greater extent. However, for this analysis it is necessary to assume they are both affected equally.
- Each design scales linearly. Thus an eight-bit ALU will use eight times the number of logic units as a single-bit ALU. Also a triple-modular redundant system will use three times the number of logic units (plus more for the voter) as the single ALU.
- The MTBF of the design can be assessed by considering the MTBF of the FPGA and multiplying by the fraction of it used by the design. While this ignores single points of failure (clocks, power lines, etc) it is not unreasonable as the most common failure-mode of an FPGA are temporary, localised degredation, typically due to stuck-at faults on interconnects (Touba, 1999).
- The ALTERA does not support partial reprogramming. Thus while the state of each cell, their arrangement and interconnections are intrinsically self-repairing, there is no way for the function that each cell performs to be corrected in the event of a failure. However, for the purposes of this analysis it will be assumed that the hardware does support reprogramming and the necessary logic is built into the BIST.

The cell component can be seen in figure 10 and a one-bit ALU comprised of 16 cells can be seen in figure 11.



a_in b_in	Logic inputs
c_out d_out	Logic outputs
n_in w_in	State inputs
e_out s_out	State outputs
reset	Global reset

Fig. 10. A cell of the ALU design

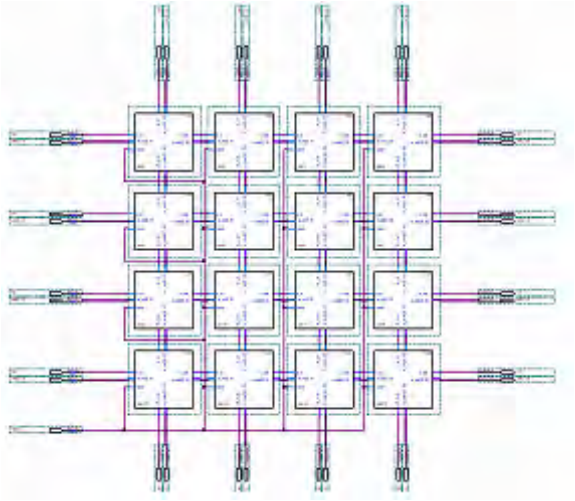


Fig. 11. A 1-bit ALU made of 16 cells

Below are the results of compiling these designs for the Stratix II FPGA.

Component	**Logic Units**	**% of FPGA**
8-bit ALU	48	<1%
3-Mod 8-bit ALU	152	2%
• ALU	48	<1%
• Voter	8	<1%
162-Mod 8-bit ALU	9120	100%
8-bit self-assembling ALU	9120	100%
• Cell	87	1%
• BIST	1	<1%

Table 1. Comparative logic unit usage of different ALU designs

A 162-modular redundant system uses the same resources as the self-assembling self-repairing design. A simple analysis shows the self-assembling solution can tolerate a greater number of failures than its static-redundancy equivalent.

The 162-modular system can tolerate $\frac{N}{2}-1=80$ module failures without a system failure.

The self-assembling system consists of 128 identical cells. Provided the self-repair code is still working, the system can tolerate 127 cell failures at any one time because each failed cell can copy the operating code from the one remaining cell.

The table below shows the reliability parameters appropriate to the ALTERA Stratix II FPGA. Using the MIL-HDBK-217F standard and a part-stress analysis, the failure rate of the FPGA can be determined as $\lambda_p = 0.32$ failures per 10^6 hours. Using this and the table above the failure rates for individual components of the design can be determined.

Variable	**Nomenclature**	**Category**	**Value**
Base failure rate	C_1	PLA > 5000 gates	0.042
Operating temperature	π_t	25°C	0.1
Package failure rate	C_2	> 300 Pins	0.16
Environment factor	π_E	Ground, Fixed	2
Quality factor	π_Q	Undocumented	1
Learning factor	π_L	> 2 years	1

Table 2. Logic unit reliability parameters for an FPGA

While it is possible to form a fault-tree model of the ALU, this would be inappropriate because of its state-dependent failure modes. A more appropriate approach is the use of a Markov model (see figure 12) of the three operating states.

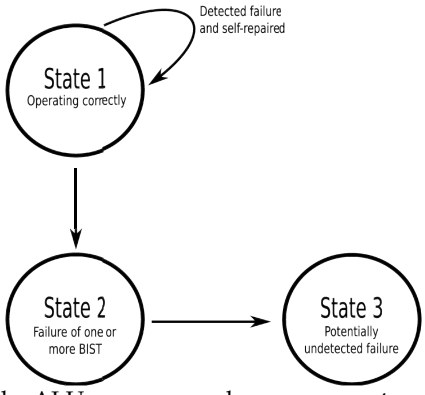


Fig. 12. Markov model of the ALU, memory and processor systems

Using the Chapman-Kolmogorov equation (1) for Markov analysis, it is possible to analyse this model in order to determine the probabilities, *P(t)*, of each failure mode.

$$\frac{d}{dt}P(t) = P(t)Q(t) \tag{1}$$

$$P(0) = [1,0,0,0] \tag{2}$$

Q(t), the transition matrix, is derived from the Markov model. Each failure rate, λ, refers to a transition between states.

$$Q(t) = \begin{pmatrix} -\lambda_1 & 0 & 0 \\ \lambda_1 & -\lambda_2 & 0 \\ 0 & \lambda_2 & 0 \end{pmatrix} \tag{3}$$

From (2) and (3) it is possible to form three coupled differential equations, (4), (5) and (6).

$$\frac{d}{dt}P_1(t) = -\lambda_1 P_1(t) \tag{4}$$

$$\frac{d}{dt}P_2(t) = \lambda_1 P_1(t) - \lambda_2 P_2(t) \tag{5}$$

$$\frac{d}{dt}P_3(t) = \lambda_2 P_2(t) \tag{6}$$

The solution to (4) that meets the initial conditions (2), is (7).

$$P_1(t) = e^{-\lambda_1 t} \tag{7}$$

Equation (5) can be rearranged to form (8).

$$\frac{d}{dt}P_2(t) + \lambda_2 P_2(t) = \lambda_1 P_1(t) \tag{8}$$

By multiplying (8) by the integrating factor $e^{\lambda_2 t}$ we obtain equations (9) and (10).

$$e^{\lambda_2 t}\frac{d}{dt}P_2(t) + e^{\lambda_2 t}\lambda_2 P_2(t) = e^{\lambda_2 t}\lambda_1 P_1(t) \tag{9}$$

$$\frac{d}{dt}\left[e^{\lambda_2 t}P_2(t)\right] = e^{\lambda_2 t}\lambda_1 P_1(t) \tag{10}$$

Substituting (7) into (10) gives (11) and (12).

$$\frac{d}{dt}\left[e^{\lambda_2 t}P_2(t)\right] = e^{\lambda_2 t}\lambda_1 e^{-\lambda_1 t} \tag{11}$$

$$\frac{d}{dt}\left[e^{\lambda_2 t}P_2(t)\right] = e^{(\lambda_2-\lambda_1)t}\lambda_1 \tag{12}$$

Integrating both sides of (12) with respect to *t* gives:

$$e^{\lambda_2 t} P_2(t) = \frac{\lambda_1}{\lambda_2 - \lambda_1} e^{(\lambda_2 - \lambda_1)t} + C \tag{13}$$

By dividing both sides of (13) by $e^{\lambda_2 t}$ we obtain:

$$P_2(t) = \frac{\lambda_1}{(\lambda_2 - \lambda_1)e^{\lambda_2 t}} e^{(\lambda_2 - \lambda_1)t} + \frac{C}{e^{\lambda_2 t}} \tag{14}$$

$$P_2(t) = \frac{\lambda_1}{\lambda_2 - \lambda_1} e^{-\lambda_1 t} + Ce^{-\lambda_2 t} \tag{15}$$

Substituting the initial conditions (2) into (15) we can determine *C*.

$$0 = \frac{\lambda_1}{\lambda_2 - \lambda_1} e^{-\lambda_1 .0} + Ce^{-\lambda_2 .0} \tag{16}$$

$$0 = \frac{\lambda_1}{\lambda_2 - \lambda_1} + C \tag{17}$$

Substituting (17) into (15) we get a final answer for P_2(t).

$$P_2(t) = \frac{\lambda_1}{\lambda_2 - \lambda_1} (e^{-\lambda_1 t} - e^{-\lambda_2 t}) \tag{18}$$

The sum of the state probabilities must be one, because the system must reside in one of the three states.

$$\sum\nolimits_i P_i(t) = 1 \tag{19}$$

Thus the solution to (6) is found by subtracting the results from the previous two results.

$$P_3(t) = 1 - P_1(t) - P_2(t) \tag{20}$$

Figure 13 shows the reliability $(P_1(t) + P_2(t))$ versus time of the self-repairing ALU and an N-modular redundant ALU of an equivalent size.

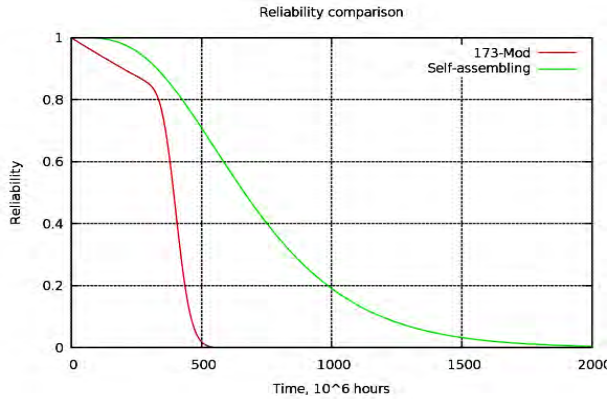


Fig. 13. Reliability comparison (No redundant cells)

The reliability demonstrated by this self-repairing design is due to it being able to tolerate those failure modes that can be corrected by reconfiguring the device. The reliability of the self-repairing ALU can be further increased by introducing small numbers of redundant cells to the design to replace irreparably broken cells. Figure 14 shows the reliability of a self-repairing ALU with 2 redundant cells as well as the reliability of an N-modular redundant system of equivalent size. Figure 15 shows the reliability of a system with 5 redundant cells, figure 16 shows the reliability of a system with 10 redundant cells.

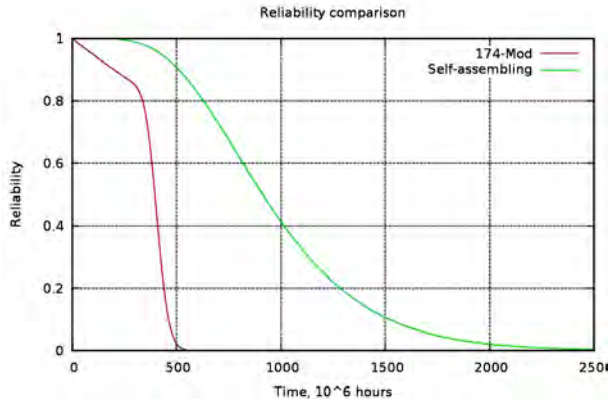


Fig. 14. Reliability comparison (2 redundant cells)

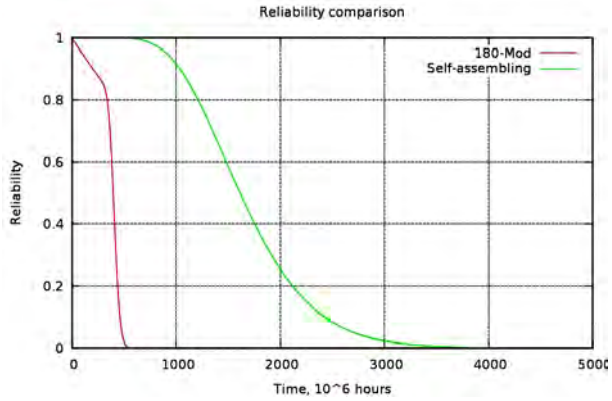


Fig. 15. Reliability comparison (5 redundant cells)

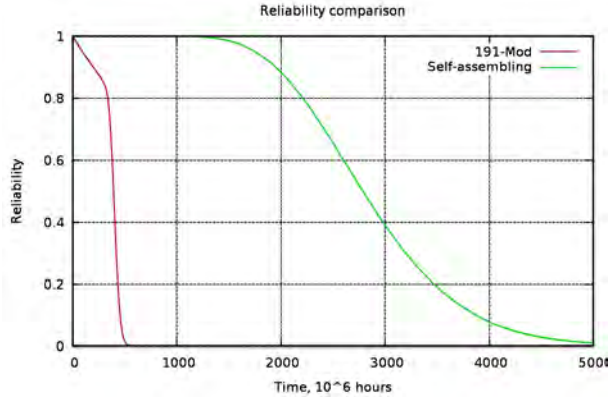


Fig. 16. Reliability comparison (10 redundant cells)

In order to quantify the reliability improvement shown in figures 13-16 the mean time to failure (MTTF) of each system will be calculated at 400.10^6 hours.

$$MTTF = \frac{400}{ln\left(\frac{\lambda_1}{\lambda_2 - \lambda_1}(e^{-400.\lambda_1} - e^{-400.\lambda_2}) + e^{-400.\lambda_1}\right)}$$

Redundant cells	**MTTF (10^6hrs) of self-assembling technique**	**MTTF (10^6hrs) of N-mod technique**
0	$5.730x10^2$	$5.004x10^2$
1	$2.144x10^3$	$5.009x10^2$
2	$1.007x10^4$	$5.362x10^2$
5	$3.560x10^6$	$5.367x10^2$
10	$1.240x10^{12}$	$5.040x10^2$

With no redundant cells available to the self-assembling design, the MTTF is slightly longer than that of its N-modular equivalent. However making available small numbers of cells significantly increases its MTTF (by a factor of 10^{10} hours in the case of 10 redundant cells) performance over its equivalent N-modular design.

6. Conclusions

This technique, whilst demonstrating an improved reliability over an equivalent N-modular redundant system, is not without cost:

- The minimum size of the self-assembling ALU is a factor of a hundred times greater than that of a single ALU and over thirty times that of a triple modular redundant ALU.
- The design algorithm is not trivial. Modular redundancy can be easily applied to any system without a significant schematic re-design. This is not true of the self-assembling approach. Firstly an appropriate level for cellular discretisation must be selected. Next, the system function must be divided between cells such that short, preferably not overlapping, buses between nearest neighbour cells are the primary means of data transmission. Lastly the rules necessary to co-ordinate the assembly and differentiation of these must be generated.

For the reasons above, this approach may not be particularly applicable to the design of reliable, commercial projects. However it certainly fits into the 'Ultra reliability' category. Any system that must operate in harsh environments (e.g. satellites and space shuttles), or that cannot tolerate failure, could benefit from this technique.

The self-assembling, self-repairing capabilities of morphogenesis can be mimicked in the design of self-assembling, self-repairing electronic circuits. These circuits are formed on an array of discrete, identical cells. To achieve correct system performance, each cell determines its state then uses this to determine its component type. Furthermore, these arrays can be designed to metamorphosise into different circuits depending on the boundary conditions of the array. This design approach has been demonstrated with an ALU design on an FPGA and its reliability assessed using state-based models and a stochastic analysis. Subsequently, this reliability was compared to the reliability of an N-modular ALU of equivalent FPGA logic unit use, and found to perform significantly better.

7. References

Berrill N.J, & Cohen, A. (1936). Regeneration in Clavellina lepadiformis, *Journal of experimental biology*, 13.

Jones, D. H; McWilliam, R. P. & Purvis, A. (2009), Designing convergent cellular automata, *Biosystems*.

Shapiro, A. (2006), Ultra-Reliability at NASA, *44th AIAA Aerospace Sciences Meeting and Exhibit*.

Thoma, Y; Sanchez, E; Arostegui, J. M. & Tempesti, G. (2003), A dynamic routing algorithm for a bio-inspired reconfigurable circuit, *Lecture notes in computer science*.

Touba, D. N. A. (1999), A low cost approach to detecting, locating and avoiding interconnect faults in {FPGA}-based reconfigurable systems, *Proceedings of the International conference VLSI design*.