A Bio-inspired Method to Realize Fault-tolerance Online

Yi Xiao[1]*, Huaxiang Lu[1,2], Gang Chen[1], Wenyu Mao[1]
[1]*Institute of Semiconductors, CAS, Beijing, China*
[2]*CAS Center for Excellence in Brain Science and Intelligence Technology, Beijing, China*
xiaoyi@semi.ac.cn, luhx@semi.ac.cn, chengang08@semi.ac.cn, maowenyu@semi.ac.cn

***Abstract*—A multi-layered array inspired by biological processes and structures to support the self-diagnosis and self-healing of circuits is proposed. A dedicated distributed dynamic maximum flow (DDMF) algorithm is carefully designed to run on the array with an acceptable cost of time and hardware resources which makes self-adaptive online recovery possible. The array consists of three layers logically. Cells of the Programmable Logic Layer can be configured to different circuit functions. The Immune Layer, which cooperates with cells to detect and locate faults, is inspired by the human body's powerful immune system. Reconfiguration Manage Layer executes the dynamic routing algorithm to map the circuit function to spare cells automatically to realize online recovery. A fault-tolerance hardware platform is built based on the array. As the simulation and synthesis results show, the array performs well as a programmable platform and has the ability of distributed online self-diagnosis and self-healing.**

Keywords—fault-tolerance, bionic hardware, self-healing, self-diagnosis, distributed routing algorithm, intelligent system

I. INTRODUCTION

Nowadays, the density and complexity of chips are growing rapidly. More complex systems are more difficult to avoid manufacturing errors or occasional internal faults [1,2] The majority of electronic systems require long-term fail-free operations especially those who work as the key components in harsh environments like outer space and abysmal sea, in which they cannot be manually repaired or replaced easily. Therefore, the ability of an electronic system to operate correctly in the presence of faults or to be fault-tolerant attracts a lot of attentions of researchers all over the world.

In recent years, many novel methodologies inspired by biological processes were worked out by scientists [3]. The human body is one of the most complex systems ever known. Failures are not rare, but the overall function is highly reliable with the self-diagnosis and self-healing mechanisms working ceaselessly throughout our bodies [1]. Lots of efforts have been made to exploit the possibilities to design systems that are able to self diagnose and self heal just like their natural counterparts [8].

Embryonic electronics [3-7] draws inspiration from the cell system of creature [2]. Immunotronic [9,10,17] intends to explore the processes performed by the human immune system to realize self-diagnosis. POEtic [11-13] machines aim to create a multi-cellular, self-contained substrate to interact with the environment to develop and adapt its functionality through processes of evolution, development and learning.

The method presented in this paper is based on prior works, however, different from them in many fundamental ways. Cost performance is the foremost challenge they face [2]. A large number of resources are used to construct the bio-mimicked structures. For instance, the electronic DNA of a cell, which has a complement copy of the description of the system, tries to achieve the same abilities of cellular differentiation and self-restoration as their biological counterparts. The resources needed to implement this method will increase largely with the expansion of the scale of the system. The electronic DNA does contribute to the fault-tolerance and self-adaptation features to some extent. However, the cost is too high for a price-sensitive and area-sensitive electronic system A new multi-layer fault-tolerant array and the dedicated self-diagnosis and self-repairing mechanisms are proposed in this paper. Our work aims to provide reliability and engineering practicability at the same time.

II. STRUCTURE AND BIO-INSPIRATION

A. Overview

As shown in Fig. 1, the array is composed of three layers logically. The Programmable Logic Layer can be programmed to specific functions. It is monitored by the Immune Layer. If faults are diagnosed, the Reconfiguration Manage Layer will start the self-healing processes.

B. Programmable Logic Layer

We draw our inspiration from what is similar to Embryonics [3-6]. The Programmable Logic Layer consists of identical programmable units that we call cells. Cells are the smallest unit of programmable hardware. Each cell is connected to its four neighbors directly. Fig. 2 presents the inside structure of the cell.

The Function Module is the logic execution unit of the cell. Its structure is very similar to standard FPGAs. It mainly contains look up tables (LUTs) and flip-flops (DFFs).The former acts as a universal logic gate to implement any type of combinational function while the latter aids the realization of sequential one.

The Switch Box is the routing resource of the cell. Neighborhood connections of cells and long connections

National Defense Science and Technology Innovation Fund of Chinese Academy of Sciences(CXJJ-17-M152); Pilot project of Chinese Academy of Sciences (XDA18040400); National Natural Science Foundation of China(No. 61701473); Science and Technology Planning Projects of Beijing (Z181100001518006).
*Corresponding author

978-1-7281-0467-6/19/$31.00 ©2019 IEEE

crossing the array can be achieved through cooperation among the Switch Boxes.

The nucleus stores genes — the configuration strings, as shown in Fig. 3. They determine how the Function Modular works and how the cells connect with each other. The Hamming-testing Module tests and corrects the segmented genes strings continuously.

The Immune Judge Module is one of the self-diagnosis facilities. As shown in Fig. 2 it accepts signals from different immune cells, judges its own state and also sends diagnosis signals to other part. This is part of the united-diagnosis mechanism that will be discussed in section 2.D in detail.

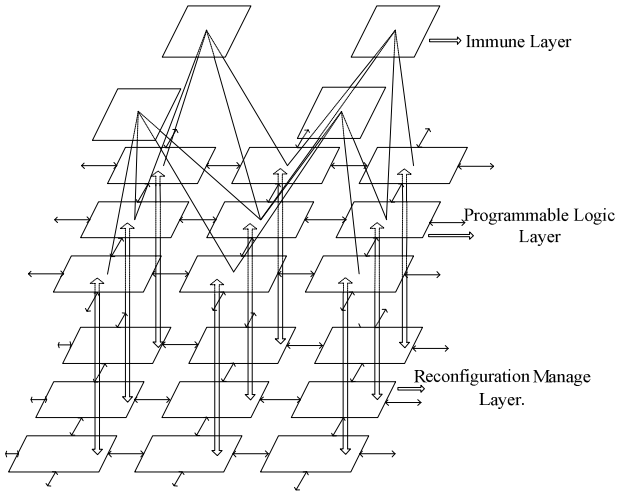

Fig. 1. Overall structure of the array

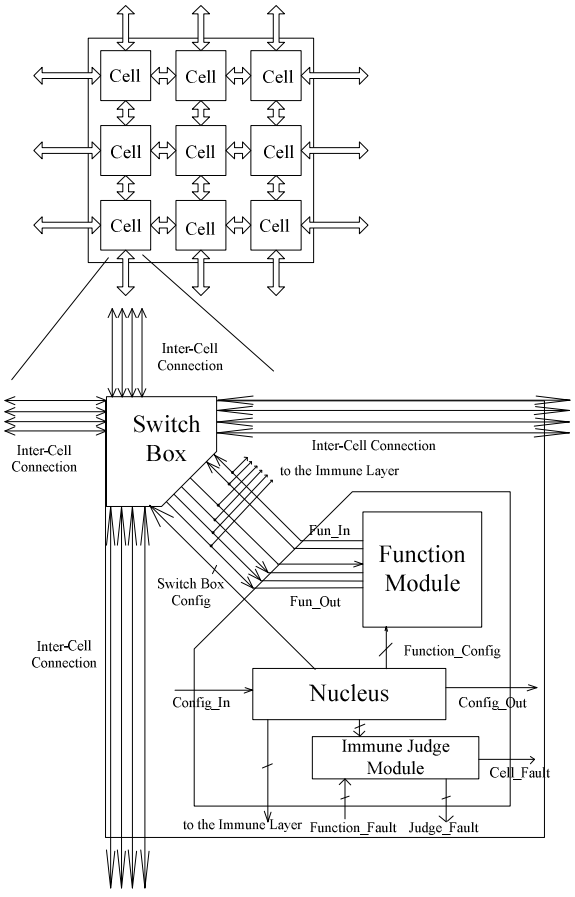

Fig. 2. Structure inside a cell

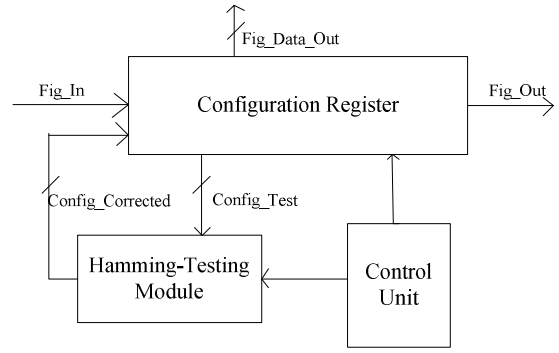

Fig. 3. Structure of the Nucleus

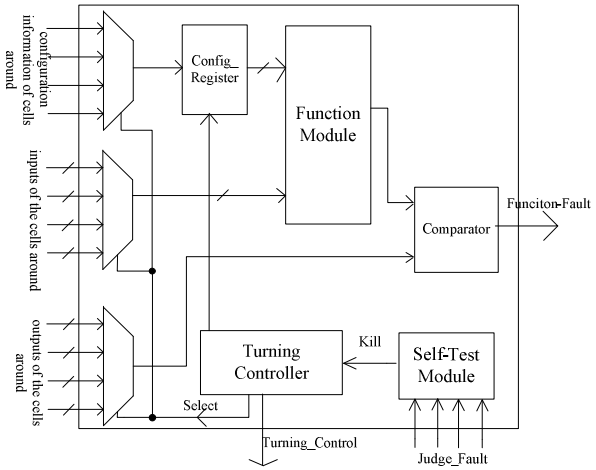

Fig. 4. Structure of the immune cell

C. *Immune Layer*

The immune system protects the body from invasions and maintains reliable operations even in the presence of bacteria and viruses [9]. The great diversity and large quantity of immune cells and materials are distributed throughout the body without any central control, so the process will not be stopped for single-point failures. Our supervision tasks are also distributed in the immune layer. Dynamic-turning approach is applied to immune cells to mimic the biology diversity. An immune cell is responsible for the nearest four cells around. It monitors one cell at a time and turns to another after a fixed period of time called a time slice. If we arrange the turning appropriately, the cells are under testing at any time [14]. Fig. 4 is the structure of the immune cell. It dynamically loads the cell's configuration bits to its Configuration Register and share the inputs of the cell. The Comparator will send a positive 'Function_fault' signal if the result does not match.

Before immune response acting，immune system dose several tests to identify the invasion. This increases the reliability largely. We take inspiration from immune system's multi identifications [14] to establish a united-judge mechanism. As mentioned above, a cell is under the supervision of four independent immune cells around it. After a full turning cycle, there will be four independent feedback 'Function_Fault' signals. As shown in Fig. 3, these signals go into the Immune Judge Module of the cell. Signal 'Cell_Fault' is generated to reconfiguration manage layer to indicate its own state by analyzing with the majority rule. Four 'Judge_Fault' are generated to the four immune cells

accordingly to indicate if the cell thinks the immune cells are in fault states. An immune cell is also under independent monitoring with the 'Judge_Fault' signals from the four cells around it. As shown in Fig. 4, these 'Judge_Fault' signals are connected by the Self_Test Module to generate a diagnosis signal 'Function_Fault' to indicate the state of this immune cell. A positive 'Function_Fault' signal will trigger the suicide process of the immune cell. Its Turing Controller stops turning and the relevant cells stop accepting its 'Function_Fault' signals as if the immune cell is transparent.

As described above, self-diagnosis is achieved through cooperation between immune cells and cells. This is what the 'united' means. The result is based on the analysis of multi independent monitoring signals to decrease the affection of instantaneous faults The bi-direction supervision between cells and immune cells make sure the immune mechanisms are fault-free themselves. With the majority rule, the killing of one immune cell will not affect the diagnosis of the cells around it or other immune cells sitting beside it largely, even if the elimination of two immune cells around one same cell at the same time will not.

D. Reconfiguration Manage Layer

When the immune system cleans up the sick, senescence and dead cells, the body replenishes the cells through the division of health cells nearby. Body's function maintains.

The organism can grow, reproduce and revive by absorbing nutrition; the electronic system cannot. However, with the unified structure of cells, different functionalities of the cell are only decided by configuration strings and the input data. We could insert several spare cells that initially do not have any specific functions in our Programmable Logic Layer to work as replacements. Fig. 5 shows two arrangement examples of spare cells in the array. 'S' represents the spare cell. 'N' represents the normal cell.

The custom circuit function is translated to connected logic unit that can be mapped into the array. Cells in the array are configured and connected with each other according to the logical and sequential relationships of these logic units. During recovery, we should not only find the substitution cell but also rearrange the connections of the fault cell to the replace one. In the development flow of FPGAs or ASICs, place and route needs sophisticated algorithms, massive computation and usually ample time. These are usually performed by powerful processors and software. Although the rerouting is a local action, it also needs careful arrangement. It is the most important problem should be solved to repair the system online.

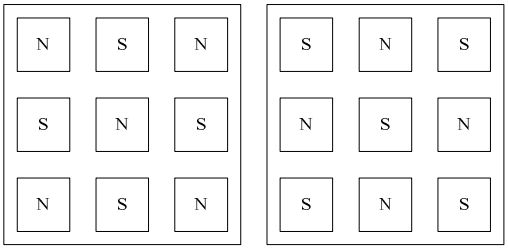

Fig. 5. Two distribution of spare cells in the array

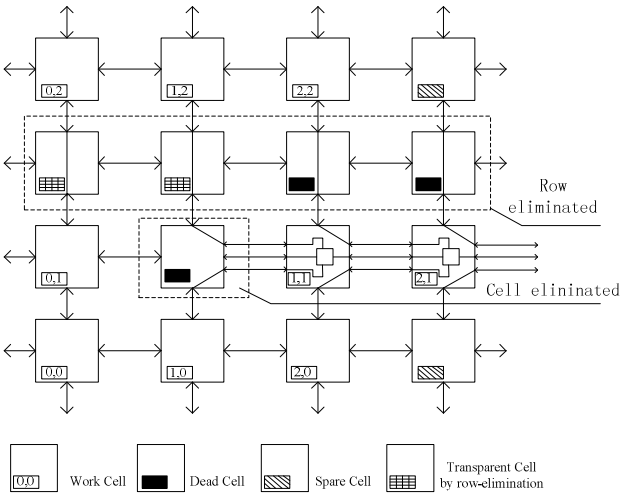

Fig. 6. Possible rerouting of cells

In prior works, several methods are studied to perform local reroute. The most used one is the fixed replace and reroute strategy [15]. The replacement has two stages. First: spare cell sits in the same row replace the faulty cells. When the number of fault cells in the same row exceeds the number of spare cells in that row, second: the whole row is eliminated and all the cells are logically shifted upwards. A spare row will take over the functionality of the faulty row. Fig. 6 presents the two rerouting method accordingly[15]. In cell elimination, the oblique paths are used to reroute the signals. In row elimination, the signals only need to pass through the transparent cells to reach the new row.

This process of replacement and routing is quite simple and friendly to hardware. But they do have shortcomings. If faults concentrate on the same row, with the right side shift replacement, large numbers of routing resources will be used for rerouting. So the number of faults can be tolerated in one row simultaneously is very limited even if there are adequate spare cells sitting in the same row. In addition, the row-eliminated scheme will waste large number of resources because most of the health cells in the row will be abandoned only for the repairing of the minority fault cells, especially when the scale of the circuit expands.

One of the major limiting factors for the development of evolvable hardware principles is the lack of simple and compact method in system dynamic routing strategies [16]. It is true that replace and reroute is a complex task. It needs to choose the most appropriate connections considering the length of the critical path, the rational use of routing resources, the time to recover and the consumption of extra hardware to rebuild the data paths. The Reconfiguration Manage Layer and a dedicated rerouting algorithm proposed in this paper aims to do the rerouting in an autonomous and flexible way, and to find the balance of these aspects.

We draw inspiration again from the concept of a living being's distributed mechanisms. The seeking of available signal paths, sophisticated algorithm to make the best rerouting choices and many other related works are all distributed in the Reconfiguration Manage Layer. As shown in Fig. 1, the Reconfiguration Manage Layer consists of homologous units. In rest of the paper, we will call them 'unit' to simplify. Each cell in the Programmable Logical Layer has a corresponding unit .There are data paths between them. The unit can read the connection information of the

cell by accessing the configuration bits of the Switch Box. These units undertake the concrete tasks in recovery. Fig. 7 shows the structure of the unit. It has an information register and a flow controller in every direction. The registers are used to store the inherent routing information and the temporary routing data created during the process. The flow controller makes local routing decisions in the rerouting algorithm. The Controller in the middle is in control of accessing the cell's inherent routing data and starting the recovery process when it receives the positive 'Cell_Fault' signal.

In human body, function recovery of the damaged cells are mostly performed by healthy cells sitting beside them, because they have similar functionalities and tight connections with those damaged cells. Considering the length of the new signal path, the complexity to make appropriate routing decisions, and the time and resource consumptions, this neighbor-replacement strategy inspires us to do the replace and route within a nearby and relatively small range at first. So, the units in the Reconfiguration Manage Layer are divided into blocks. Each block consists

nine cells' corresponding units. The replace and reroute process will be conducted within the same block of the fault cell at the beginning. Only if it fails, the replace and reroute search scope will expand as shown in Fig. 8.

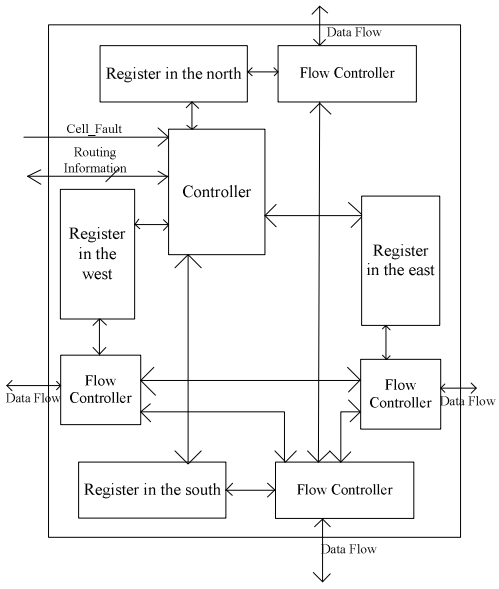

Fig. 7. Structure of the unit

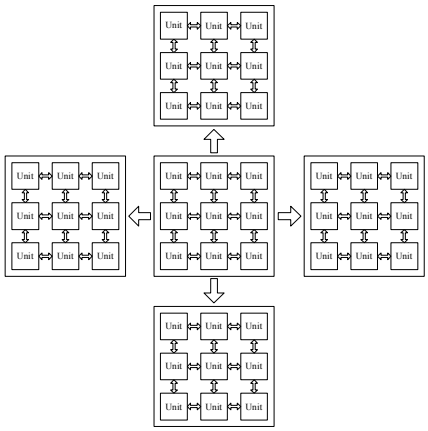

Fig. 8. Expansion of the searching scope

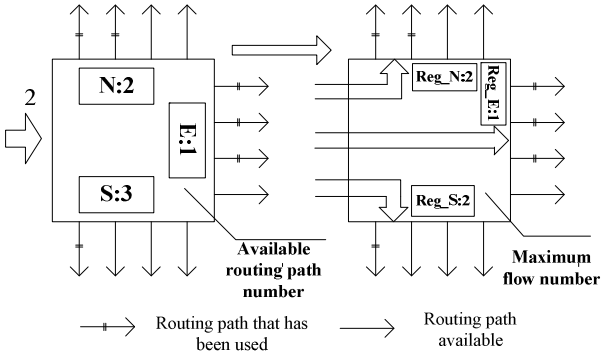

Fig. 9. Example of Stage 2

This nearest blocked researching can make best of the existing original signal paths and take advantages of the fault-cell-nearest position to find the shorter rerouting paths. The scope of searching is fixed so the choices of available replacement cells and signal paths are limited. This reduces the workload of the rerouting algorithm and makes it easier to implement on hardware.

As mentioned above, the traditional replace and reroute process demands a high computational load which hardware cannot afford. A new distributed dynamic maximum flow (DDMF) algorithm is well designed to match the blocked units to solve the problem efficiently on hardware.

Distributed: This is the main idea throughout the array design. The algorithm conducting is distributed in the array, each unit undertakes a part of the computation and decision-making.

Dynamic: The new signal paths are established in an incremental constructive process. The algorithm process starts from the fault cell and dynamically spreads in the form of data flow, which carries the information of the available data paths in the block. They are self-adaptive to different situations.

Maximum flow: All the signals connected to the fault cell should be rerouted to the replacement cell. Sometimes, the number of available paths to do the reconnection is greater. Making the reasonable decisions is the key point for maintaining unimpeded signals. As we have already considered the path length, the choices are made mainly to avoid the block up of the data path. Because the available routing resources are limited inside a block especially in some densely routed directions. Sometimes, some directions may be critical to a certain signal because without their help the signal cannot be connected to the target cell. If the critical path is used by other signals, the rerouting process could not be finished within the block. Maximum flow aims to push the signal flow to the widest direction, which has the most unused routing paths, to make sure the rerouting decision made currently will not block the signal in the following.

No matter which unit in the block receives a 'Cell_Fault' signal from the corresponding cell, the DDFM algorithm starts. There are four main stages in the recovery:

Stage 1: Find replacement cell in the block. This is similar to a breadth-first search. Search signals start from the fault cell's unit. The fault cell is tagged as the source cell. The unit will spread out the search signal if its corresponding cell is not a spare cell otherwise, broadcast the message that

it is a spare one in the block's local build-in bus and is tagged as the target cell, the search finishes.

Stage 2: Route-searching signal proceeding — find out all the possible paths linking the source unit and the target unit, and record them in the units' registers along the way. At this stage, one search signal is sent out at a time from one direction of the source cell. This signal travels as broadly as possible until the target cell or endpoint. It carries the maximum available data path number, which is called flux in the following, in this direction. When it comes in to one unit, the flow controller of this direction decides the out flux of other directions by analyzing this signal and the original routing paths. Fig. 9 shows an example of how it works. The search signal flows in from the west of the unit, and it has the flux of two. Assuming the available data paths are two, one and three in the north, east and south directions respectively. Then, the out flux of the east direction of this unit is one because no matter how many signals flow into the unit, the output flux will not be greater than the available path. The out flux is two in the north and two in the south in the same way. The process iterates until the searching signal proceeds to the target cell. And when all the directions of the source cell have been searched, all the possible paths from the source to the target unit are found and recorded.

Stage 3: Track backwards the appropriate path for all signals needed to reroute. According the processes above, only the paths that can reach the target cell have records in its unit's register. The tracking signal is sent out once a time from the target unit to decide the rebuilding path for one certain signal. The path in which the search signal flows through means a real signal path when recovery finishes. When it flows into a unit, the flow controller of this direction decides the output direction with the rule of choosing the max available path according to the previous recording in all registers to minimize the problem of key path using up. As the decision of where the tracking signal flows means a real rerouted signal path, the available routing resources recorded in related registers need to update accordingly. Until the tracking signal reaches the source unit, the rerouting path for this certain signal is established. The process iterates until all of the signals needed to reroute find their paths.

Stage 4: Update the configuration bits of the Switch Box of the cells along the rerouted paths.

If any of the stages cannot finish for the lack of routing resources, the research target will be parted and the research scope expands to the four directed neighbor blocks as shown in Fig. 8. As the algorithm is distributed within the block, the expanded blocks can work independently and concurrently. This parallel work largely reduces the time needed to finish the recovery. The time increases with the scale of the block and not the scale of the array. Therefore, the expansion of the circuit will not affect the recovery time.

The algorithm reroutes the signals according to the inherent routing situations and the decisions made previously. This self-adaptive method can handle different routing requirements flexibly and uses the resources rationally. As the algorithm is conducted within the block which has limited space the time consumption has a limitation. The structure of entities in all three layers does not change with the scale of the array. Only the number of entities increases with the expansion of the array, and there are no extra resource consumptions.

III. EXPERIMENTAL RESULTS

The behavior model of the reconfigurable array has been described using the Verilog language, and a fault-tolerance hardware platform is built based on it in the Xilinx Spartan6 FPGA demo board. In order to verify the features described in pervious sections, we mapped a 4bits parallel multiplier in the array with the scale of 7 cells×7 cells. The details of the placement and routing are shown in Fig. 10. X3~X0 and Y3~Y0 are inputs and P7~P0 are the outputs of the multiplier. 'S' represents the spare cell. The block boundaries are also tagged. Cell with a '+' gate represent a one-bit full adder. Lines between cells are signal paths. Immune cells are not presented.

First, logic function of the 4bits multiplier has been fully tested by the 256 possible combination of inputs. The array can work normally as a configurable device.

As most faults faced by hardware can be seen as a form of stuck-at faults [17], we mainly discuss the monitoring and recovery of stuck-at faults here. The faults are manually injected into the array.

A. Tests For the Fault Tolerance Abilities

Cell with the color gray in Fig. 10 is the one chosen to inject faults.

We randomly chose one immune cell around the gray cell to inject the error. It's 'Function_Fault' turned to '1' for the injected error. However, the other three immune cells were

- Fault tolerance ability of configuration bits

We randomly choose a positive bit in gray cell's configuration register and set it to negative manually to simulate a stuck-at '0' fault. The simulation wave of the ISE, software development environment of Xilinx in Fig. 11 shows that at the time of 700ns, a stuck-at '0'error was injected into the 'config_data' register, 'error' was detected in the next clock. The output of the adder turned to a wrong value. However, as the Hamming-Testing Module worked, in the next clock, a corrected value was written to the 'config_data'. It is the same as the one before we injected the error. Output of the adder turned right. The Hamming-Testing Module in nucleus works appropriately. It can detect the error in configuration register and correct one bit error.

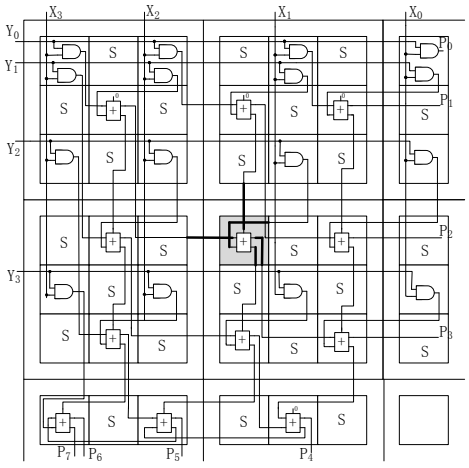

Fig. 10. Detailed placement and routing of the multiplier before fault injection

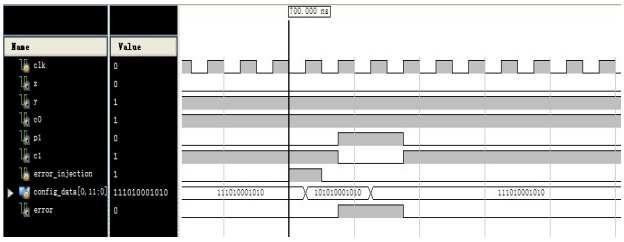

Fig. 11. Simulation wave of ISE software tool to verify the fault tolerant ability of configuration bits

- United detection of faults

According to the majority rule, the gray cell believed the fault was happening in the immune cell with injected error. As this immune cell has four cells around it, other three cells also sent back positive 'Judge_Fault'to it, the immune cell was sure of the fault of itself. The suicide process of this immune cell was triggered. After the first injection we again randomly chose another immune cell to inject error. And still the condition to make sure the error of cell is not satisfied. The error-injected immune cell suicided, the gray cell continued to function well.

- Recovery

This time error was injected to the gray cell's Function Module. Four immune cells all sent back positive 'Function_Fault' to the gray cell. Gray cell knew the fault was in its own structure, 'Cell_Fault' was set to '1'. Recovery process was triggered. In Fig. 10, the bolding black solid lines around the gray cell meant the signals needed to reroute. Fig. 12 is the detailed placement and routing after recovery. The gray cell in Fig. 12 is the replacement cell found for the gray cell in Fig.16. The bolding black dashed lines were one-to-one correspondence with the signals needed to reroute in Fig.10 After recovery, a full 256 inputs test was added to the recovered adder. The results were all correct, the recovery was successful. From the detailed placement and routing figure we can find that the rerouted paths are rationally distributed in the block, not concentrate on one direction to get the shortest paths. The data paths around the replacement cell are not blocked.

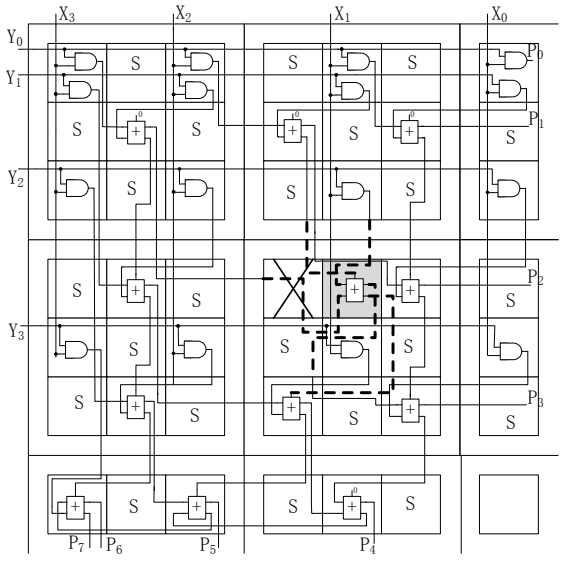

Fig. 12. Detailed placement and routing of the multiplier after recovery

B. Recovery time consumption analyzing

There are two main stages in analyzing the recovery time. First is the stage of united-fault identification, second is the fault recovery.

Stage1: United-fault identification. Before the identification, four time slices are necessary to collect the related signals. Therefore, the time consumed here is Tconfirm = 4×Nperiod×Tclk, 'Nperiod' is the clock number of one time slice, 'Tclk' is the clock cycle.

Stage2: Fault recovery.

Replacement researching: Replacement researching signal starts from the fault cell's unit. It extends one cell per clock cycle. The searching is inside the block so the worst situation to find a replacement within the block is Tfind=4×Tclk.

Routing: In forward searching of Algorithm DDMF, the most clock cycle used to traverse all the paths from one direction of the source cell to the target cell is Nergod = (4-1)×2+1 = 7. There are three directions in the worst situation, D=3 times traversals are needed. Forward finding time in total is Tergod = Nergod×D×Tclk. In the backwards tracking, clock cycle is the same as Nergod. Four inputs and two outputs signals of the function module need to reroute in the worst case, the time consumed then is Tback = Nergod×Nsignal×Tclk. 'Nsignal' is the number of signals that needed to reroute. The total is Troute = Tergod + Tback = Nergod×D×Tclk + Nergod×Nsignal×Tclk. This time consumption is only relevant to the scale of the block not the scale of the array. It will not increase with the massive extension of the circuit.

Information transfer: The configuration strings of the fault cell will be transferred to the replacement cell in this step. Ttrans=Nconfig×Tclk, 'Nconfig' represents the number if bits.

IV. CONCLUSION

In this paper, we present a new hardware architecture inspired by biological processes and structures to support the self-diagnosis and self-recovery of circuits. A dedicated algorithm fitting well with the proposed hardware structure is carefully designed to ease the physical implementation of an actual online recovery system.

The array is composed of three layers with homogeneous entities logically. All of the function, detection and recovery facilities are distributed in these entities. Cells in the Programmable Logic Layer are programmed to realize custom functions. The Immune Layer, working with the function cells to detect and locate faults, is inspired by human body's powerful immune system. Reconfiguration manage units are organized in blocks. The dedicated algorithm is customized for the blocked structure and makes it possible to implement a flexible self-adaptive online recovery with acceptable time and resource consumptions.

The verification performed on the fault-tolerant hardware platform built on the proposed structure have proven this bio-inspired reconfigurable array can diagnose faults and automatically recover from faults. The routing resources are rationally used in the rerouting process, and the time and resource consumptions are acceptable. It can be used as the base hardware platform to support fault tolerance systems.

REFERENCES

[1] C. Ortega, A.M. Tyrrell, "Biologically inspired fault-tolerant architectures for real-time control applications," *Control Engineering Practice*, 7, 673-678,1999.

[2] P. Bremner, Y. Liu, M. Samie, M. Samie, G. Dragffy, A.G. Pipe, G. Tempesti, J. Timmis, A.M. Tyrrell, "SABRE: a bio-inspired fault-tolerant electronic architecture, " *Bioinspiration & Biomimetics*, 8, 016003(16pp), 2013.

[3] D. Mange, M. Sipper, P. Marchal, "Embryonic electronics," *Biosystems*, 51,145-152, 1999.

[4] X. Zhang, G. Dragffy, A. Pipi, "Embryonics: a path to artificial life?" J. Artif. Life: 12, 313-332, 2006.

[5] X. Zhang, G. Dragffy, A. Pipi, "Bio-inspired reconfigurable architecture for reliable systems," *VLSI'03*, Int. Conf. on VLSI: 34-40, 2003.

[6] C. Ortega, D. Mange, S. Smith, A.M. Tyrrell, "Embryonics: a bio-inspired cellular architecture with fault-tolerant properties," *Genetic Program. Evolvable Mach*,1, 187-215, 2000.

[7] C.S. Szasz, V. Chindris, "Fault-Tolerant Embryonic Network Development for High Reliability Mechatronic Applications," *Int. Rev. Appl*. Sci. Eng: 1(1-2), 61-66, 2010.

[8] M. Samie, G. Dragffy, D. Gabriel, et. al, "Unicellular Self-Healing Electronic Array," 2nd International Through-life Engineering Services Conference: 400-405, 2013.

[9] D.W. Bradley, A.M. Tyrrell, "Immunotronics—Novel Finite-State-Machine Architectures with Build-in self-test using self-nonself differentiation," *IEEE Transactions on Evolutionary Computation*, vol. 6, no. 3, pp. 227-238, 2002.

[10] D.W. Bradley, A.M. Tyrrell, "Immunotronics: Hardware Fault Tolerance Inspired by the Immune System," *ICES 2000*:11-20, 2000.

[11] W. Barker, A.M. Tyrrel, "Hardware Fault-Tolernce Within the POEtic System," *ICES 2005*,LNCS 3637:25-36, 2005.

[12] C. Teuscher, "On the State of the Art of POEtic Machines," *Technical Report 01/375*, SwissFederal Institute of Technology Lausanne, 2001.

[13] Y. Thoma, G. Tempesti, E. Sanchez, Juan Manuel Moreno Arostegui, "POEtic: an electronic tissue for bio-inspired cellular applications," *Biosystems*: 76,191-200, 2004.

[14] D. Bradley, C. Ortega-Sanchez, A. Tyrrell, 2000. "Embryonics+ Immunotronics: A Bio-Inspired Approach to Fault Tolerance". *Nasa/dod Workshop on Evolvable Hardware*: 215-233.

[15] X. Zhang, G.Dragffy, A.G. Pipe, N. Gunton, 2003. "A Reconfigurable Self-Healing Embryonic Cell Architecture". *Proceedings of the International Conference on Engineering of Reconfigurable Systems and Algorithms*, Las Vegas, Nevada, USA.

[16] Moreno J.M. Arostegui, E. Sanchez, J. Cabestany, "An In-System Routing Strategy for Evolvable Hardware Programmable Platforms," Conference Evolvable Hardware Proceedings The Third NASA/DoD Workshop on: 157-166, 2001.

[17] Y. Zhang, "Research on the hardware fault-tolerance technology. Department of Measurement Technology and Instruments", Master's Thesis, Nanjing University of Aeronautics and Astronautics, 2009.