IEEE Access
Multidisciplinary | Rapid Review | Open Access Journal

Date of publication xxxx 00, 0000, date of current version xxxx 00, 0000.

Digital Object Identifier 10.1109/ACCESS.2017.DOI

A dynamical evolution approach to high placement performance and low fault-tolerant cost for bio-inspired self-repairing hardware

LIU XIUBIN[1,2], LI DUO[1,2], LI YUE[1,2]

[1]Science and Technology on Integrated Logistics Support Laboratory, National University of Defense Technology, Changsha 410073, China
[2]College of Intelligence Science and Technology, National University of Defense Technology, Changsha 410073, China

Corresponding author: Li Yue (e-mail: liyue@nudt.edu.cn).

This work was supported by the National Natural Science Foundation of China under Grant 51675523.

ABSTRACT Bio-inspired self-repairing hardware is a reconfigurable system characterized by high fault-tolerant ability. To further increase the placement performance and reduce the fault-tolerant cost, a dynamical evolution approach to multi-objective dynamic placement is developed. A primary challenge for our study is computational complexity. Based on group theory, the computing task was divided on the dimensions of time and space to increase the utilization of online computing resources. By introducing the multi-step placement transformation, a discrete dynamical system model was built and the multi-objective dynamic placement was converted into a stability problem of constrained systems. In order to satisfy the dynamical requirements, an artificial particle system model was built and its evolution laws were verified by dynamic analysis. By simulating the evolution laws, a dynamical evolution algorithm was proposed, which could avoid large-scale iterative calculation used in the conventional optimization search algorithms. Experiments have shown that the dynamical evolution method could offer high placement performance with low fault-tolerant cost. Besides, it also has the advantage of high design flexibility since there is no initial placement constraint.

INDEX TERMS Bio-inspired self-repairing hardware, dynamical evolution, placement performance, fault-tolerant cost.

I. INTRODUCTION

WITH the development of integrated circuit technology, the demand for fault-tolerant electronic systems with high performance and low cost is increasing steadily. Recent studies are focusing on the use of reconfigurable system to satisfy requirements [1]. Nevertheless, a primary concern of reconfigurable system is computational complexity [2]. Dynamic reconfiguration is a highly computationally intense task and in some cases, the computational complexity increases with the size of programmable resources by exponential growth [3]. In order to avoid the combination explosion problem, researchers mimicked the behavioral pattern of multicellular organisms and developed bio-inspired self-repairing hardware to solve complex computational tasks by local rules without any central control [4], [5]. As a reconfigurable, fault-tolerant system, bio-inspired self-repairing hardware is characterized by decentered architecture, distributed computing, and dynamic partial reconfiguration [6]. It uses multiple processing elements to increase the computing power and relocate configuration information segments (or function blocks) instead of generating the entire bitstreams to reduce the calculation amount.

In bio-inspired self-repairing hardware, function blocks can be migrated from fault regions and relocated to other regions by dynamic placement. During this process, there are two noteworthy parameters, including the reconfigurable hardware cost needed in fault tolerance (hereafter called Fault-tolerant Cost) and the functional circuit performance after dynamic placement (hereafter called Placement Performance) [7]–[10]. Currently, linear removal methods (e.g., Embryonics [11]) and neighbor replacement methods (e.g., Self-Repairing Digital System [12]) have high Placement

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3006121, IEEE Access

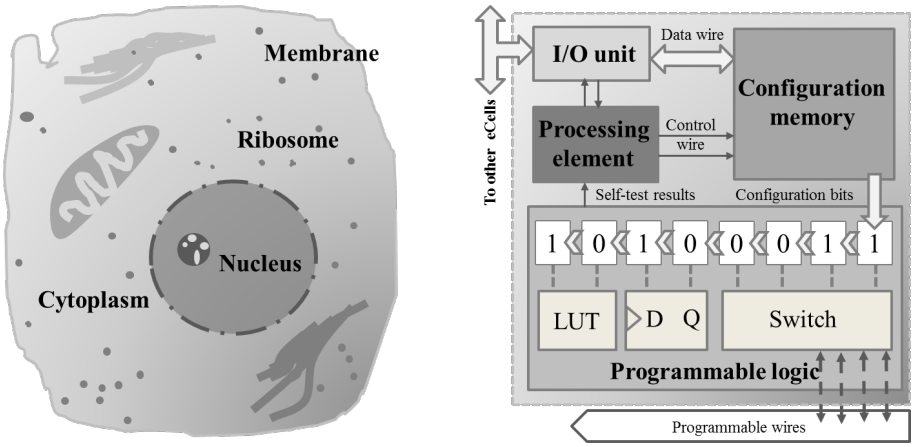


FIGURE 1: A comparison between a biological cell and an electronic cell. (a) A biological cell mainly consists of membrane, nucleus, ribosome, and cytoplasm. (b) An electronic cell mainly consists of an input/output unit, a configuration memory, a processing element, and a group of programmable logic.

Performance and high Fault-tolerant Cost; while global replacement methods (e.g., eDNA [13]) have low Placement Performance and low Fault-tolerant Cost. So far, however, there is still a lack of effective dynamic placement methods in bio-inspired self-repairing hardware that can achieve high Placement Performance and low Fault-tolerant Cost.

The goal of this paper is to develop a multi-objective dynamic placement approach to high Placement Performance and low Fault-tolerant Cost. In order to achieve this online computational task, the calculation amount should be reduced, and existing processing elements should be made full use of. By introducing the multi-step placement transformation to replace single-step transformation, bio-inspired self-repairing hardware was converted into a discrete dynamical system, and the computing resource is improved on the time dimension. Based on Lyapunov's stability theory [14], the multi-objective dynamic placement was described as a stability problem of constrained systems, and the calculation amount was reduced by avoiding extra information processing (e.g., routing information or timing information) that is often used in offline placement model. Based on the principle of dynamics, a self-repairing system was developed where all the processing elements were needed to take part, thus the computing resource is increased on the space dimension. Moreover, by simulating an artificial particle system that could satisfy dynamical requirements, a corresponding evolution algorithm was proposed, which could avoid large-scale iterative calculations used in the conventional optimization search algorithms. The following parts are organized as follows: section II presents the basic principle and related work, section III makes a model for the multi-objective dynamic placement; section IV develops a self-repairing dynamical system; section V exhibits experimental results; and section VI concludes the paper.

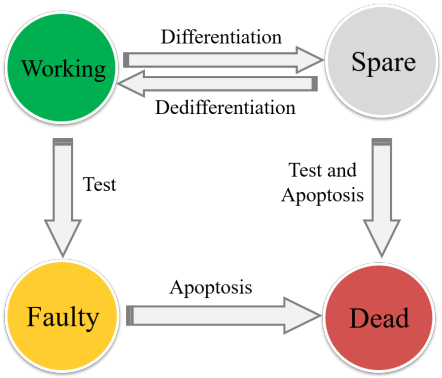


FIGURE 2: State transition of eCells.

II. BASIC PRINCIPLE AND RELATED WORK

Bio-inspired self-repairing hardware is a distributed, reconfigurable, fault-tolerant system, supporting a flexible implementation of function circuits on an artificial multicellular array [15]. As the basic structural and functional unit in analogy to the biological cell, every electronic cell (eCell) is composed of four parts (Fig.1):

1) an input/output unit, working as the cell membrane, supports the communication with other eCells;
2) a configuration memory contains a copy of the genome, in analogy to the nucleus;
3) a processing element, acting as the ribosome, is responsible for interpreting the genome;
4) a group of Programmable Logic (PL), working as the cytoplasm, is used for implementing the intended function block [16].

The working process of eCells can be modelled by a Finite State Machine (FSM) with four basic states, including:

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3006121, IEEE Access

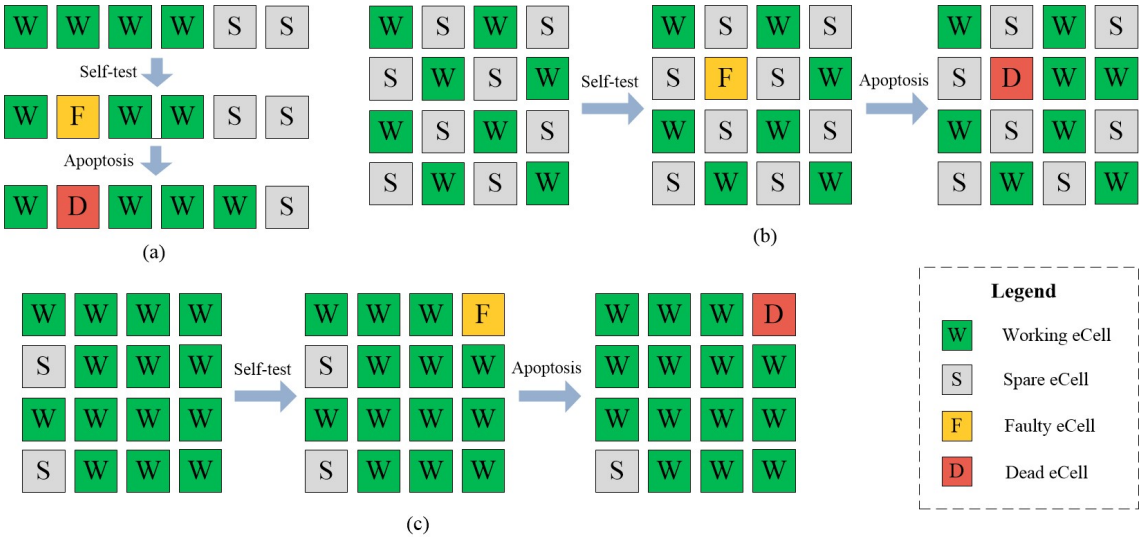


FIGURE 3: Current dynamic placement methods: (a) Linear removal method, (b) Neighbor replacement method, (c) Global replacement method.

1) $working$, the PL is fault-free and associated with a function block;
2) $spare$, the PL is fault-free and associated with no function block;
3) $faulty$, the PL is faulty and associated with a function block;
4) $dead$, the PL is faulty and associated with no function block.

The eCell state transition is shown in Fig.2. Cellular differentiation and dedifferentiation are a pair of reversible transitions, achieved by extracting the specific gene within the genome and implementing the corresponding block on the PL [17]. Nevertheless, the self-test and apoptosis are generally irreversible and unidirectional transitions. The self-test is performed on the spare and working eCells by monitoring behaviors, analyzing outputs, and identifying faults. If a fault is detected in a spare eCell, it will be immediately induced to apoptosis by stopping the functionality and isolating the PL from the array. If a fault occurs in a working eCell, it will turn into a faulty one at first, and then a dead one when its associated function block is removed.

The arrangement of function blocks is called a placement. The initial placement is made up of working eCell and spare eCells. Working eCells are used to implement the given function circuits and spare ones are for fault tolerance. However, with the increase of faults, the system will lose efficacy and have no ability to implement the function circuits. The placement at this moment is called the terminal placement. In the whole life between the initial and the terminal placements, bio-inspired self-repairing hardware works as a cellular automaton that repeatedly experiences two phases: dynamic placement and placement interval. The dynamic placement starts when a faulty eCell is identified and halts when all the faulty eCells are turned into dead ones. During this phase, all the PLs are deactivated. ECells automatically change their states or renew function blocks according to the environment information and the placement keeps changing by local rules without any central control [6]. By contrast, the placement interval is in a stable state without any faulty eCell. The PLs of working eCells are activated and function circuits can work normally.

In the early research, the linear removal method was a major dynamic placement approach. In Embryonics [11], eCells were connected sequentially in a linear chain. Initially, all the spare eCells were arranged at one end of the chain. If one working eCell was found faulty, its function block, together with its following eCells', would synchronously move by one eCell. Meanwhile, the function block of the last working eCell would be passed to the next spare one if it exists (Fig.3 (a)). The similar method could be generalized to a two-dimensional rectangular array, where the column (or row) containing the faulty eCell would transfer function blocks to the nearest right column, which would repeat the same action, and so on until a spare column was reached [18]. Since a column of spare eCells was used to mask one faulty eCell, 2D Embryonics had high Fault-tolerant Cost.

Thereafter, further improvements are mainly based on replacement rules, which are conducted by deactivating the faulty eCell and activating another spare one rather than a spare column for replacement. Inspired by the endocrine cellular communication, Yang et al. developed a self-repairing digital system where spare eCells were evenly embedded into working eCells in the initial placement (Fig.3 (b)) [12]. One faulty eCell can be replaced by any of its spare neighbors. Nevertheless, Fault-tolerant Cost was still large since every spare eCell provided fault tolerance in a small region.

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3006121, IEEE Access

TABLE 1: A comparison between current dynamic placement methods

	Main Performance Parameters		Computing Resource Utilization	
	Placement Performance	Fault-tolerant Cost	Transformation Step	Processing Elements Used
Linear Removal	High	High	One	Many
Neighbor Replacement	High	High	One	Few
Global Replacement	Low	Low	One	Few
Hybrid Replacement	Medium	Medium	One	Few
Conclusion	There is a lack of effective methods to achieve high Placement Performance and low Fault-tolerant Cost.		The computing resource utilization is low	

By contrast, based on efficient on-chip communication architecture (e.g., crossbar [19], [20], shared bus [21], and network on chip [22]), global replacement method was proposed and Fault-tolerant Cost was greatly reduced. For example, in eDNA [13], public channels were used for fault information broadcast. When a fault was detected, a spare eCell with the highest priority would be activated and the cellular differentiation would be conducted for replacement (Fig.3 (c)). Nevertheless, since some function blocks would be transmitted far away from their original positions, the function circuit timing performance (e.g., timing skew and transmission stability) declined and the routing complexity and total wire length increased, leading to low Placement Performance.

In addition, some dynamic placement methods with medium Fault-tolerant Cost and medium Placement Performance were proposed to provide more flexible choices. For example, Szasz et al. presented a hierarchical system with an extra cluster layer [23]. One cluster was composed of a switch matrix and some spare and working eCells. The replacement was limited to eCells among the same cluster. Kim et al. [24] proposed a hybrid system with two kinds of spare eCells, including redundant eCells and stem ones. In the self-repair, each working eCell had a corresponding redundancy to replace its function instantly. If the redundancy was used up, a stem eCell was immediately configured to the redundancy.

In conclusion, current methods are conducted based on a one-step placement transformation, and only a few processing elements are used to execute the calculation task (Table 1). Linear removal methods and neighbor replacement methods have high Placement Performance and high Fault-tolerant Cost; while global replacement methods have low Placement Performance and low Fault-tolerant Cost. However, there is a lack of effective methods that can achieve high Placement Performance and low Fault-tolerant Cost.

III. MULTI-OBJECTIVE DYNAMIC PLACEMENT MODEL

In this section, two indexes are proposed to make quantitative descriptions of Fault-tolerant Cost and Placement Performance. Based on group theory, the computing task is divided on the dimensions of time and space to increase the utilization of online computing resources. By introducing the multi-step placement transformation, a discrete dynamical system model is built, and the multi-objective dynamic placement is converted into a stability problem of constrained systems. Then the optimal performance that the system can achieve is analyzed by a constructive proof.

A. QUANTITATIVE INDEX

1) Fault-tolerant Cost index

Let $n_0(w)$, $n_0(s)$ denote the number of working and spare eCells in the initial placement. Let $n_T(w)$, $n_T(s)$, $n_T(f)$, $n_T(d)$ denote the number of working, spare, faulty, and dead eCells in the terminal placement. Then we have the following equations:

$$\begin{aligned} n_0(s) &= n_T(s) + n_T(d) \\ n_0(w) &= n_T(w) + n_T(f) \end{aligned} \tag{1}$$

Fault-tolerant Resource Utilization (FRU) can be defined as the fault-tolerant number (the number of dead eCells) in the terminal placement divided by spare eCell number in the initial placement, expressed as a percentage:

$$FRU = n_T(d) / n_0(s) \times 100\% \tag{2}$$

Fault-tolerant Resource Loss (FRL) can be expressed as the number of spare eCells unused in the terminal placement compared with the number of spare eCells initially existing in the system, expressed as a percentage:

$$FRL = n_T(s) / n_0(s) \times 100\% \tag{3}$$

Fault-tolerant Cost can be reduced by increasing the FRU value or decreasing the FRL value. According to Eq.(1), the sum of FRU and FRL is equal to 1.

$$FRU + FRL = 1 \tag{4}$$

2) Placement Performance index

Let B and E denote the function block set and the eCell set. The placement can be expressed as a mapping from function blocks to eCells, denoted by

$$\sigma : B \to E. \tag{5}$$

Assume that the initial placement, denoted by σ_0, has the best placement performance. When function blocks are transmitted away from their original positions, the function circuit timing performance (e.g., timing skew and transmis-

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3006121, IEEE Access

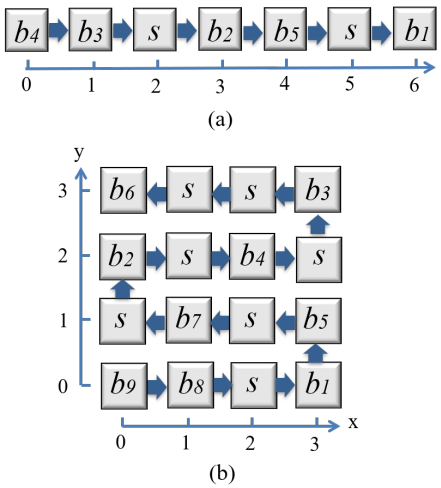


FIGURE 4: Placement sequences are constructed by recording blocks along (a) a unidirectional path starting from the origin in a linear chain; (b) a snaking path starting from the origin in a two-dimensional array. Thus, the placement sequences can be written by $\{b_4, b_3, s, b_2, b_5, s, b_1\}$ and $\{b_9, b_8, s, b_1, b_5, s, \cdots, b_6\}$, respectively.

sion stability) declines and the routing complexity and total wire length increase. For any placement σ_t, its Placement Performance Loss (PPL) can be expressed as the maximal displacement of function blocks

$$PPL = \max_{b \in B} d\left(\sigma_0\left(b\right), \sigma_t\left(b\right)\right) \tag{6}$$

where $d\left(x, y\right)$ is the Manhattan distance that uses the sum of the absolute differences of Cartesian coordinates between two points as the metric.

B. TASK DECOMPOSITION

Dynamic placement is a computationally intense task. The algebra structure of placement transformation inspires us to develop a task decomposition method to make full use of online computing resources.

Assume that spare eCells are associated with a special block, or spare block, denoted by s. The arrangement of blocks can be represented by a placement sequence, which is constructed by recording blocks along a continuous path. As shown in Fig.4, it is a unidirectional path in a linear chain or a snaking path in a two-dimensional array. For any two adjacent terms in one placement sequence, their corresponding eCells are still neighbors in the hardware system, but the reverse is not always true.

Let $P_{(B,E)}$ denote a set of placement sequences based on the function block set B and eCell set E. For any $p \in P_{(B,E)}$, it satisfies the following three properties:

1) The sequence length is equal to the number of eCells.
2) Let p_n denote the n^{th} term of p. For any natural number $n \leqslant |E| - 1$, $p_n \in \{s\} \cup B$.
3) For any function block $b \in B$, there exists one and only one term p_n in p such that $b = p_n$.

A metric can be defined on $P_{(B,E)}$. Given $p, q \in P_{(B,E)}$, their distance can be defined by the maximal displacement of function block symbols, denoted by:

$$D\left(p, q\right) = \max_{b \in B} f_b\left(p, q\right) \tag{7}$$

where $f\left(x, y\right)$ is the distance in a one dimensional sequence.

A bridge can be built between the PPL value and the sequence difference. Let p^* denote the sequence corresponding to the initial placement (σ_0). For any placement σ_t, assume that its corresponding sequence is q. Due to the snakelike mapping order (Fig.4(b)), the PPL value of σ_t is equal to the sequence distance in a one-dimensional eCell chain, and no less than that in a two-dimensional eCell array, denoted by

$$PPL \leqslant D\left(p^*, q\right) \tag{8}$$

Let us ignore constraint conditions and jump above implementation details. Then the dynamic placement can be denoted by a simple form:

$$q = gp \tag{9}$$

where $p, q \in P_{(B,E)}$ and g is a permutation identifying a specification of a way to reorder a sequence. For example, given $p = \{b_0, b_1, b_2, s\}$ and $q = \{b_2, b_1, s, b_0\}$, g can be written in cyclic form, that is, (143)(2), or more commonly, (143) to omit the unchanged element, meaning the first, fourth, and third elements are mapped to each other in a cyclic fashion, while the second element is fixed.

Let G be the set of permutations acting on the sequence. We can define a binary operation on G which behaves much like ordinary multiplication; that is, given a map $G \times G \to G$ that sends the pair (g_1, g_2) into $g_1 g_2$, satisfying the associative law, the existence of an identity element, and the existence of an inverse. Then G is a permutation group. [25]. One of the basic results on group theory says that any permutation is a product of disjoint cycles [26]. Such cycles commute with each other, and the expression of the permutation is unique up to the order of the cycles. In addition, any cycle can be expressed as a product of transpositions (2-element cycle). Given a cycle $g = (x_1 x_2 \cdots x_k)$, it is easy to see that

$$(x_1 x_2 \cdots x_k) = (x_1 x_k)\,(x_1 x_{k-1}) \cdots (x_1 x_2) \tag{10}$$

The product is not commutative and the transpositions are performed from right to left, nevertheless, the representation is not unique. From this analysis follows a conclusion that any permutation can be achieved by the composition of transpositions.

C. DYNAMICAL SYSTEM MODEL

Based on dynamical system theory, bio-inspired self-repairing hardware can be converted into a discrete-time dynamical system, denoted by a four-tuple $\left\{P_{(B,E)}, A, T_d, M\right\}$, including four ingredients: a non-empty state space $P_{(B,E)}$, a set of initial states $A \subset P_{(B,E)}$, a discrete time set

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3006121, IEEE Access

$T_d = \{0, 1, 2, \cdots\}$, a family of motions $M \subset G$ that is the set of the composition of disjoint transpositions [27]. The evolution of the system state is a sequence of discrete trajectory through the state space. For any $g \in M$, we have $g^0 p = p \in A$ and $g^n p \in P_{(B,E)}$ for all $n > 0, n \in T_d$ [28].

The aim of multi-objective dynamic placement is to increase the FRU value and to decrease the PPL value. It can be converted into a stability problem of constrained systems, with the following properties:

1) $p^* \in P_{(B,E)}$ is Lyapunov stable, that is, all the trajectories that start out near the point p^* stay near p^* forever [14];
2) $\left\{P_{(B,E)}, A, T_d, M\right\}$ evolves under the Degree of freedom (DOF) constraints, each of which can be denoted by an identical equation $p[x] \equiv s$, meaning the x^{th} term is equal to s.

The most characteristic feature of this dynamical system model is the emphasis on the evolution on the dimensions of time and space. On the space dimension, one permutation can be decomposed into commutative cycles that can be implemented on different regions in parallel, thus raising the utilization of online computing resources. On the time dimension, one cycle can be further decomposed into non-commutative transpositions that can be executed sequentially. Since complex dynamical behaviors can be achieved by iterations of simple evolution laws, the hardware design can be simplified. By contrast, current methods can be considered as a one-step transformation limited in a local region. For example, Embryonics is a one-step cycle, whereas replacement-based methods are one-step transpositions. With limited computing power, they can hardly satisfy the multi-objective dynamic placement requirements.

Then let us analyze the optimal performance that such a constrained system can achieve.

Theorem 1: Let $\left\{P_{(B,E)}, A, T_d, M\right\}$ *be a dynamical system and assume that* $p^* \in P_{(B,E)}$ *is Lyapunov stable. Assume that there exists a point* $q \in P_{(B,E)}$ *such that* $D(p^*, q) \leqslant k$ *for any* k *DOF constraints, where* $k \leqslant |E| - |B|$.

Proof: We will use an algorithm to construct a q from the p^*. Let the initial value of q be equal to p^*. In the first step, check elements in q in sequence and transfer an s to the sequence tail for k times. In the second step, transfer an s from the tail of q to the corresponding constraint position. This step continues until all the DOF constraints are used. At this moment, q can satisfy the given DOF constraints. Then we will calculate the distance between q and p^*. For any $b \in B$, it will move to the left in the first step and to the right in the second step, and the maximal displacement is no more than k. Thus $D(p^*, q) \leqslant k$ and the theorem is established.

According to Eq.(8), the PPL value is no more than its corresponding sequence distance. Thus, the optimal PPL value is no more than k when the fault-tolerant number is k. Besides, the number of DOF constraints is no more than the number of spare eCells. Thus, the maximal FRU value can reach 100%.

IV. DYNAMICAL SYSTEM DESIGN

In order to satisfy the requirements mentioned in the above section, an artificial particle system model is built and its evolution laws are verified by dynamic analysis. Relying on memory read/write control, a self-repairing dynamical system is developed to mimic the particle system. Based on the evolution laws, a corresponding dynamical evolution algorithm is proposed, which could avoid large-scale iterative calculation used in the conventional optimization search algorithms.

A. ARTIFICIAL PARTICLE SYSTEM

The artificial particle system is made up of particles 'living' on lattices [29]. A lattice is a discrete world composed of interconnected nodes [30]. Consider, for example, a square lattice with four neighbors at each node such that one node is associated with each link to the next neighbor. Let us develop the laws of motion in this artificial world. First, one particle can either stay on a node or hop from site to site, but at every tick of a clock, it can move at most one lattice unit and can never leave the lattice. Then assume that this system is governed by two forces.

One force is called coupling force. It exists between any two particles, and its final effect is to maintain the relative displacement that is determined by the initial configuration. Let α and β denote two particles. Let the displacement of β relative to α be $\Delta\mathbf{r}_\beta^\alpha$ in the initial configuration. Assume that α stays on the site $(\mathbf{r}_\alpha)$ in a new configuration. An irrotational force field will be generated in the lattice and the position of field source in the new configuration is $(\mathbf{r}_\alpha + \Delta\mathbf{r}_\beta^\alpha)$. The coupling force on β is 0 at the source, and points to the source at the other sites.

The other is repulsive force, which comes into action when particles meet in one site. Repulsive force is assumed to be far stronger than coupling force in a short range. Under its action, the situation that two particles stay in one site is avoided. For example, if one particle collides against another, it bounces back and the other stays still; if two or more particles arrive at the same site, only one randomly selected particle can stay at the current site and the others return to their original positions.

In general, the structure of the lattice is unchanged, unless a node is found faulty and becomes a hole. Accordingly, let us make an analysis for the motion of particles in unchanging and changing lattices.

1) Dynamic analysis in an unchanging lattice

It can be noted that coupling force is a conservative force, thus it can be represented by a potential energy function that satisfies the derivative condition:

$$F_c = -\nabla U \tag{11}$$

where U is the potential energy that depends only on the difference between the initial and current configurations [31]. In this system, particles move along the energy gradient direction until they collide into another one. The particle

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3006121, IEEE Access

system has less potential energy when coupling particles move close to each other. According to Lyapunov's second method, this system is stable in the initial configuration where the potential energy is 0.

2) Dynamic analysis in a changing lattice

Initially, the lattice is prepared so that each site is occupied by no more than one particle. Assume that a hole occurs at some moment and there is a particle, denoted by α, staying on this site. We can assume that α moves to its neighboring sites by an infinitesimal virtual displacement. For example, it moves to the left and there is exactly another particle on this site, denoted by β. Under the action of the repulsive force, β will continue to move to its neighboring sites until one particle, denoted by γ, move to a space site. At this moment, the virtual displacement of γ becomes a real displacement, then its following particles move in turn until α hops out of the hole. Thereafter, no particle is allowed to hop into this hole again.

In conclusion, this artificial particle system can keep stable in an unchanging lattice and hop out of holes in a changing lattice, satisfying the stability requirements and DOF constraints. Thus it can be used as a reference for the further design and implementation.

B. SELF-REPAIRING DYNAMICAL SYSTEM

As a discrete space system, the artificial particle system can be simulated by bio-inspired self-repairing hardware. The transformation is shown in Table 2, nevertheless, the action range of coupling force is limited to save routing resources and simplify system design.

TABLE 2: Transformation from artificial particle system to bio-inspired self-repairing hardware

Artificial particle system	$\rightarrow$	Bio-inspired self-repairing hardware
System evolution	$\rightarrow$	Dynamic placement
Lattice	$\rightarrow$	Distributed memory
Node	$\rightarrow$	Memory in the fault-free eCell
Hole	$\rightarrow$	Memory in the faulty eCell
Particle	$\rightarrow$	Function block
Force on particle	$\rightarrow$	Memory read
Particle motion	$\rightarrow$	Memory write
Coupling force	$\rightarrow$	Action decision
Repulsive force	$\rightarrow$	Conflict arbitration

The discrete lattice is equivalent in architecture to a distributed memory system. All the configuration memories are identical and arranged in a net structure, which can be denoted by a graph comprising vertexes and edges. One configuration memory can be considered as a hole if its corresponding eCell is faulty, and a node if fault-free.

Particle with memory effects can be simulated by function block comprising three bit fields. The first field is an identifier. Next is a configuration data field used for implementing the intended functionality on the PL unit. The last part is a coupling information field, which consists of the initial displacement data of its coupling function blocks.

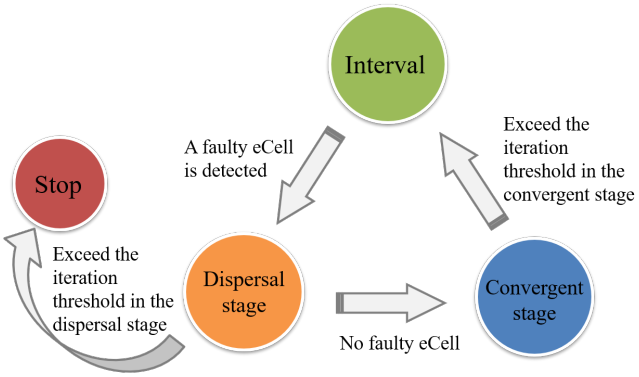


FIGURE 5: Evolution stage transition.

The force on particle and particle motion correspond to memory read and write. The read permission can only access the identifier, whereas the write permission deals with the entire information of function block. Since the particle system only supports the motion between neighboring sites, each eCell can write memories among its four adjacent eCells and itself. Besides, the memory read scope is restricted to a square region with a width of 5 to save routing resources. That is, assume that the eCell is located at (x_c, y_c), it has memory read permission of eCells located at $\{(x, y) \,|\, |x - x_c| \leqslant 2, |y - y_c| \leqslant 2\}$. Then the coupling pairs are restricted in the neighboring blocks in the initial configuration.

Coupling force and repulsive force can be achieved by action decision and conflict arbitration. According to the memory read results, the potential energy values at different sites are calculated, and a memory write request will be sent to the site with the minimum value. If two or more write requests are received, only one can be accepted by arbitration for data transmission, and the rejected ones have to wait for the next cycle.

C. DYNAMICAL EVOLUTION ALGORITHM

Bio-inspired self-repairing system is a synchronous system where the states are updated synchronously at discrete time levels. In one clock cycle, each function block can choose one among five possible motions, includes: staying at the current eCell, or moving one eCell to the left, right, up, and down side. Let MC, ML, MR, MU, MD denote these five motions. The necessary condition for the motions $\{ML, MR, MU, MD\}$ is that the next eCell is in the spare state; otherwise, the motion MC is used. In this system, the dynamic placement can be divided into three stages including interval, dispersal stage, and convergent stage (Fig.5).

The dispersal stage starts when one working eCell turns into a faulty one and continues until this eCell is dead. Let $\mathbf{r}_f$ denote the position vector of the faulty eCell. Given a function block $b \in B$, which stays on the site with the

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3006121, IEEE Access

position vector ($\mathbf{r}_b$), it will move in the direction against the faulty eCell. Let Ω denote all the optional position vectors. The optimal site can be calculated by:

$$\min_{\mathbf{x}_b \in \Omega} \left(|\mathbf{x}_b - \mathbf{r}_f| - |\mathbf{r}_b - \mathbf{r}_f| \right) \tag{12}$$

For example, assume the position vector of the faulty eCell is $(5, 5)$. For any function block, if it is located at $(2, 5)$, it will choose the motion ML; if located at $(2, 2)$, it will make a random choice between ML and MD. From a macro perspective, function blocks surrounding the faulty eCell move towards the peripheral spare eCells in a wave-like manner.

The convergent stage follows the dispersal stage and ends when exceeding the maximum number of iterations. Since every eCell can obtain limited environment information, the region potential energy, defined as the sum of displacement of coupling function blocks in a local placement, is used for calculation. Given a pair of coupling function blocks $b, k \in B$, let $\Delta\mathbf{r}_b^k$ denote the displacement of b relative to k in the initial placement. Assume the position vectors of b, k be $\mathbf{r}_b, \mathbf{r}_k$ in a new placement. When only k is taken into consideration, the potential energy of b can be calculated by:

$$U_b^k = d\left(\mathbf{r}_b, \mathbf{r}_k + \Delta\mathbf{r}_b^k\right) \tag{13}$$

When considering a set of coupling particles of b in a local placement, denoted by C, the region potential energy of b can be calculated by:

$$U_b^C\left(\mathbf{x}_b\right) = \sum_{k \in C} d\left(\mathbf{x}_b, \mathbf{r}_k + \Delta\mathbf{r}_b^k\right) \tag{14}$$

For any function block, it will calculate the region potential energy on different sites and choose a site with the minimal cost value:

$$\min_{\mathbf{x}_b \in \Omega} \left(U_b^C\left(\mathbf{x}_b\right)\right) \tag{15}$$

Dynamical evolution algorithm mainly consists of three steps. The first step is the eCell state transition according to Fig.2. Next is the evolution stage transition according to Fig.5. The last is memory read/write control, which is triggered in the dispersal and convergent stages (Algorithm 1). Thereafter, the system returns to the first step, and the cycle continues until the system stops.

Based on the principle of dynamics, dynamical evolution algorithm can make full use of online computing resources on the dimensions of time and space, and avoid large-scale iterative calculations used in the conventional optimization search algorithms. Besides, without the initial placement constraint, the algorithm can be used in a wider application range and the design process is more flexible.

Algorithm 1: Memory read/write control

For each $working$ or $faulty$ eCell, run these codes synchronously:

1: Read memory in a local region;
2: Choose a cost function according to the evolution stage;
3: Calculate the cost value;
4: Choose a $spare$ neighbor with the smaller cost value than itself;
5: Send Request messages to that neighbor;
6: Wait until receiving Response;
7: Transfer data to that memory.

For each $spare$ eCell, run these codes synchronously:

1: Wait until receiving Request messages;
2: Choose a Request randomly;
3: Make a Response to that Request;
4: Write the received data to memory.

V. EXPERIMENTAL RESULTS AND ANALYSIS

In this section, the design of simulation experiments is introduced. Based on the experimental results, the FRU and PPL values are counted, and Fault-tolerant Cost and Placement Performance are analyzed. Then a comparison is made between the linear removal method, global removal method, and dynamical evolution method.

A. SIMULATION EXPERIMENTS

The performance of dynamical evolution approach will be tested by simulation experiments under different cases (e.g., array size, spare eCell number, initial placement, and fault position). Based on Python 3.7.0, every eCell can be defined as an object comprising data members (e.g., address and function block) and methods (e.g., state transition method, evolution stage transition method, and memory read/write control method). Relying on a global clock, methods in different eCells can be executed concurrently. During the placement interval, one randomly selected working eCell can be turned into a faulty one to simulate fault injection. Then the self-repairing dynamical system restarts the dynamical evolution. If the system returns to the placement interval after the convergent stage, the maximal displacement of function blocks will be counted, and meanwhile, another fault will be inserted. Otherwise, the algorithm halts and the fault number is counted.

According to the array size and spare eCell proportion, our experiments can be classified into ten groups, and each group consists of 100 experiments with different initial placements. Among these groups, five are based on a 10×10 eCell array and another five on a 20×20 array. Let us denote the former five groups by $S\left(n\right)$ and denote the latter five by $L\left(n\right)$, where n is the number of spare eCells in the initial placement. The experiment conditions of different groups are shown in Table 3. There are 1000 experiments in total, and the results will be analyzed to evaluate Fault-tolerant Cost and Placement Performance.

B. FAULT-TOLERANT COST ANALYSIS

Fault-tolerant Cost can be evaluated by the FRU value (Eq.(2)). Every experiment corresponds to one FRU value,

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3006121, IEEE Access

TABLE 3: The experiment conditions of different groups

	Array Size	Spare eCell Number [a]	Working eCell Number [a]	Spare eCell Proportion [a]	Experiment Number
$S(5)$	10×10	5	95	5%	100
$S(10)$	10×10	10	90	10%	100
$S(20)$	10×10	20	80	20%	100
$S(30)$	10×10	30	70	30%	100
$S(40)$	10×10	40	60	40%	100
$L(20)$	20×20	20	380	5%	100
$L(40)$	20×20	40	360	10%	100
$L(80)$	20×20	80	320	20%	100
$L(120)$	20×20	120	280	30%	100
$L(160)$	20×20	160	240	40%	100

[a] This data comes from the initial placement.

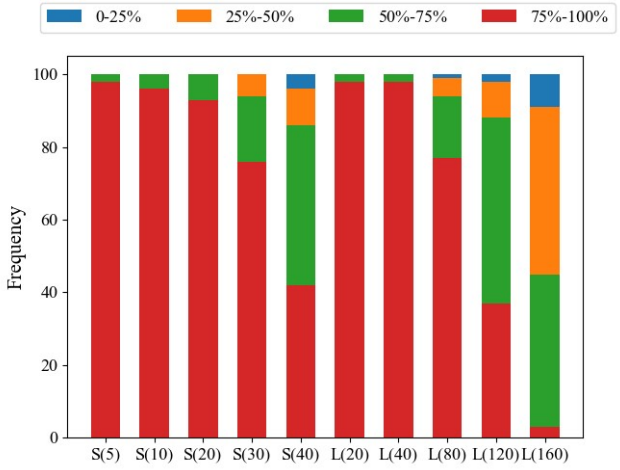


FIGURE 6: A stacked bar graph to compare FRU between different groups. The FRU values of each group are classified into four categories: $0-25\%$, $25-50\%$, $50-75\%$, and $75-100\%$, represented by different colors.

thus there are 100 values in one group. As is shown in Fig.6, a stacked bar graph is made to exhibit all the data by dividing FRU values in four categories: $0-25\%$, $25-50\%$, $50-75\%$, and $75-100\%$. For example, there are 96 FRU values exceeding 75% and 4 values varying from 50% to 75% in the group $S(10)$. Since the system may lose efficacy before an expected life when all the movements are blocked by dead eCells, FRU varies randomly and no group can achieve the 100% FRU value for all the experiments.

The average fault-tolerant number and the average FRU value are counted in Table.4. The average fault-tolerant number goes in the opposite direction from the average FRU value. When comparing groups with the same array size, the average FRU shows a small decrease at first, and then declines dramatically with the increase of spare eCell number. When comparing groups with the same spare eCell proportion, there is no significant difference between groups with the proportion equal to 5% and 10%. Nevertheless, with the increase of spare eCell proportion, the FRU difference gradually increases.

TABLE 4: A comparison of fault-tolerant number and FRU value

	Spare eCell Number [a]	ECell Number	Spare eCell Proportion [a]	Fault-tolerant Number [b]	FRU [b]
$S(5)$	5	100	5%	4.83	96.6%
$S(10)$	10	100	10%	9.51	95.1%
$S(20)$	20	100	20%	18.81	94%
$S(30)$	30	100	30%	25.2	84%
$S(40)$	40	100	40%	28.1	70.25%
$L(20)$	20	400	5%	19.64	98.2%
$L(40)$	40	400	10%	37.91	94.8%
$L(80)$	80	400	20%	66.79	83.5%
$L(120)$	120	400	30%	81.4	67.8%
$L(160)$	160	400	40%	77.79	48.6%

[a] This data comes from the initial placement.
[b] This data is an average value.

It implies that the dynamical evolution algorithm has greater advantage in Fault-tolerant Cost when the proportion of spare eCells is small. With the increase of spare eCell proportion, the fault-tolerant ability increases slowly; while the FRU value declines dramatically.

C. PLACEMENT PERFORMANCE ANALYSIS

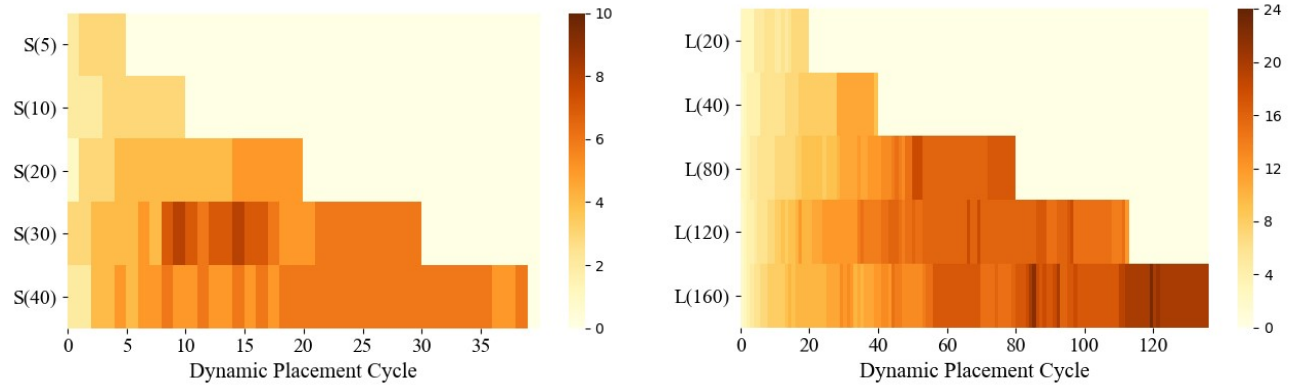
Placement Performance can be evaluated by the PPL value (Eq.(6)). Compared to FRU, PPL has larger data size since it is counted at the end of each dynamic placement cycle rather than each experiment. All the PPL values are divided into different data sets, and one data set consists of PPL values that belong to the same group and the same dynamic placement cycle. The maximal PPL value in each data set is exhibited in a heat map (Fig.7). These values are no more than 8 in the $S(n)$ groups and no more than 23 in the $L(n)$ ones. PPL shows a gradually increasing trend with the increase of dynamic placement cycles and finally tends to be stable. Nevertheless, there is no significant difference along the vertical axis (spare eCell proportion) in Fig.7. Then a further analysis is made at the 15^{th}, 35^{th}, 75^{th}, and 105^{th} cycles. All the data sets related to these cycles in the $L(n)$ groups are chosen to make a boxplot (Fig.8). In the 15^{th} cycle, the box medians increase in order of spare eCell proportion. The other cycles have similar properties.

It can be found that small spare eCell proportion has better Placement Performance than large one. Besides, the PPL value is usually less than its corresponding dynamic placement cycle, implying that dynamical evolution algorithm can achieve the optimal performance that is described by *Theorem 1*.

D. METHOD COMPARISON

Let us make a comparison between the linear removal method, global removal method, and dynamical evolution method. All the methods are repeated 100 times based on the same simulation environment, where the array size is set to 10×10 with 10 spare eCells in the initial placement. In the linear removal method, there is an additional constraint on

This article has been accepted for publication in a future issue of this journal, but has not been fully edited. Content may change prior to final publication. Citation information: DOI 10.1109/ACCESS.2020.3006121, IEEE Access



(a) (b)

FIGURE 7: A heat map of the maximal PPL values. Figure (a) is a comparison between $S(5)$, $S(10)$, $S(20)$, $S(30)$, and $S(40)$. Figure (b) is a comparison between $L(20)$, $L(40)$, $L(80)$, $L(120)$, and $L(160)$. Each colored block represents the maximal PPL value in the corresponding data set. The value with dark color is greater than that with light one.

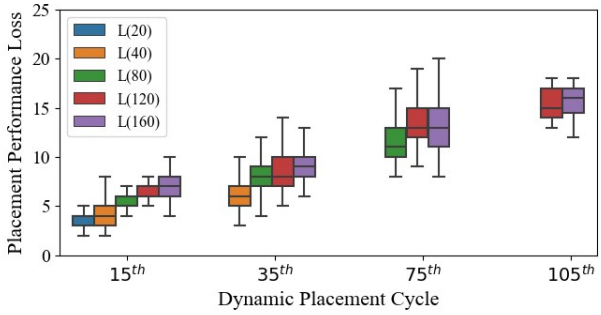


FIGURE 8: A boxplot of data sets at the 15^{th}, 35^{th}, 75^{th}, and 105^{th} dynamic placement cycles. There are 5 short horizontal line in a box, representing the lower extreme, lower quartile, median, upper quartile, and upper extreme of one data set. For example, in the data set of the $L(160)$ at the 15^{th} cycle, the five numbers are 4, 6, 7, 8, and 10, respectively.

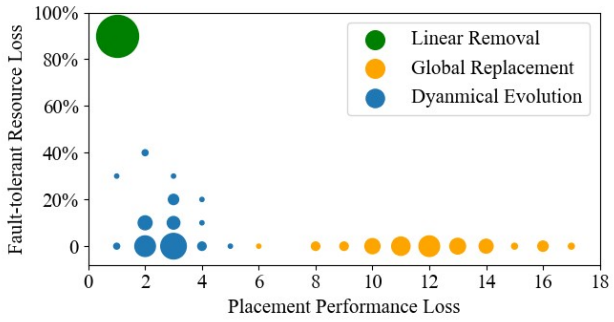


FIGURE 9: A comparison between the linear removal method, global removal method, and dynamical evolution method. The area of one bubble represents the amount of data at the corresponding position.

the initial placement, that is, spare eCells should be initially arranged on the rightmost column. In the global removal method, the priority of spare eCells is specified in random order and the initial placement is generated in the same way as that in the dynamical evolution method. In each experiment, a randomly-selected working eCell will be marked as a faulty one in the placement interval until the system loses fault-tolerant ability. The remaining spare eCell number and the maximal displacement of function blocks in the terminal placement will be counted. Then the FRL value and the PPL value will be calculated by Eq.(3) and (6).

There are 100 experiments in one method and every experiment corresponds to one FRL value and one PPL value. All the experiment results are plotted on a bubble chart, where the PPL value increases from left to right and the FRL value increases from bottom to top. As is shown in Fig.9, three methods are isolated from each other without overlapping points. In the linear removal method, there is only one point on the upper-left corner. In the global replacement method, there are 11 points located along a horizontal line on the lower-right corner. In the dynamical evolution method, there are points in various sizes located in a small region on the lower-left corner. As far as FRL is concerned, the average FRL value of global replacement method ($FRL = 0$) and dynamical evolution method ($FRL = 4.9\%$) is far less than that of linear removal method ($FRL = 90\%$). Nevertheless, the average PPL value of linear removal method ($PPL = 1$) and dynamical evolution method ($PPL = 2.65$) is far less than that of global replacement method ($PPL = 12$).

In conclusion, compared with the global replacement method and linear removal method, dynamical evolution method can offer high Placement Performance with low Fault-tolerant Cost. Besides, it also has the advantage of

high design flexibility since there is no initial placement constraint. It should be noted that the proportion of spare eCells is a major parameter in applications. Small spare eCell proportion can lead to better Placement Performance and less Fault-tolerant Cost.

VI. CONCLUSION

The goal of this paper was to develop a dynamical evolution approach to high Placement Performance and low Fault-tolerant Cost for bio-inspired self-repairing hardware. The study on algebraic structure of the placement transformation has inspired a task decomposition method to raise the utilization of computing resources. By converting bio-inspired self-repairing hardware into a discrete-time dynamical system, the multi-objective dynamic placement has been described as a stability problem of constrained systems. Based on an artificial particle system model, a self-repairing dynamical system has been developed and a dynamical evolution algorithm has been proposed. Experiments have shown that the dynamical evolution method can offer high Placement Performance with low Fault-tolerant Cost. Further research is required to explore the dynamic routing approaches and to verify the dynamical evolution approach on a bio-inspired hardware platform.

REFERENCES

[1] Victor Dumitriu, Lev Kirischian, and Valeri Kirischian. Run-time recovery mechanism for transient and permanent hardware faults based on distributed, self-organized dynamic partially reconfigurable systems. 65(9):2835–2847, 2016.

[2] A. Ahmadinia, C. Bobda, S. P. Fekete, J. Teich, and J. C. Van, der Veen. Optimal free-space management and routing-conscious dynamic placement for reconfigurable devices. IEEE Transactions on Computers, 56(5):673–680, 2007.

[3] Wuxi Li, Shounak Dhar, and David Z. Pan. Utplacef: A routability-driven fpga placer with physical and congestion aware packing. 37(4):869–882, 2018.

[4] Kasem Khalil, Omar Eldash, Ashok Kumar, and Magdy Bayoumi. Self-healing hardware systems: A review". Microelectronics Journal, 93:104620, 2019.

[5] Tao Wang, Jinyan Cai, and Yafeng Meng. A novel embryonics electronic cell array structure based on functional decomposition and circular removal self-repair mechanism. Advances in Mechanical Engineering, 9(9), 2017.

[6] L. Xiubin, Q. Yanling, F. Xiangli, Z. Qingqi, and L. Yue. Dynamic placement optimization for bio-inspired self-repairing hardware. IEEE Access, 8:11007–11018, 2020.

[7] Zhu Sai, Cai Jinyan, and Meng Yafeng. Partial-dna cyclic memory for bio-inspired electronic cell. Genetic Programming & Evolvable Machines, 17(2):83–117, 2016.

[8] Meng Yafeng, Zhu Sai, and Han Chunhui. Analysis of hardware consumption of embryonic bio-inspired self-repairing system. Modern Electronics Technique, 040(6):129–132, 2017.

[9] K. Khalil, O. K. Eldash, and M. Bayoumi. A novel approach towards less area overhead self-healing hardware systems. In 2017 IEEE 60th International Midwest Symposium on Circuits and Systems (MWSCAS), pages 1585–1588, 2017.

[10] Zhai ZHANG, Yao QIU, Xiaoliang YUAN, Rui YAO, Yan CHEN, and Youren WANG. A self-healing strategy with fault-cell reutilization of bio-inspired hardware. Chinese Journal of Aeronautics, (7):1673–1683, 2019.

[11] Daniel Mange, Eduardo Sanchez, André Stauffer, Gianluca Tempesti, Pierre Marchal, and Christian Piguet. Embryonics: a new methodology for designing field-programmable gate arrays with self-repair and self-replicating properties. Very Large Scale Integration Systems IEEE Transactions on, 6(3):387–399, 1998.

[12] I. Yang, Jung Sung Hoon, and Cho Kwang-Hyun. Self-repairing digital system with unified recovery process inspired by endocrine cellular communication. IEEE Transactions on Very Large Scale Integration (VLSI) Systems, Very Large Scale Integration (VLSI) Systems, IEEE Transactions on, IEEE Trans. VLSI Syst, 21(6):1027 – 1040, 2013.

[13] Michael Reibel Boesen, Pascal Schleuniger, and Jan Madsen. Feasibility study of a self-healing hardware platform. In Proceedings of the 6th International Conference on Reconfigurable Computing: Architectures, Tools and Applications, ARC'10, page 29–41, Berlin, Heidelberg, 2010. Springer-Verlag.

[14] Anthony N. Michel and L. Hou . Stability theory of discrete-time dynamical systems involving non-monotonic lyapunov functions. Communications in Applied Analysis, 17(3):159–165, 2013.

[15] Isaak Yang, Sung Hoon Jung, and Kwang Hyun Cho. Self-repairing digital system based on state attractor convergence inspired by the recovery process of a living cell. IEEE Transactions on Very Large Scale Integration Systems, 25(2):648–659, 2017.

[16] Qingqi Zhuo, Yanling Qian, Li Yue, Nantian Wang, and Tingpeng Li. Embryonic electronics: State of the art and future perspective. In 2013 IEEE 11th International Conference on Electronic Measurement Instruments, volume 1, pages 140–146, 2013.

[17] Christof Teuscher, Daniel Mange, André Stauffer, and Gianluca Tempesti. Bio-inspired computing tissues: towards machines that evolve, grow, and learn. Bio Systems, 68(2-3):235–244, 2003.

[18] D. Mange, M. Sipper, A. Stauffer, and G. Tempesti. Toward robust integrated circuits: The embryonics approach. Proceedings of the IEEE, 88(4):516–543, 2000.

[19] James Alfred Walker, Martin A. Trefzer, Simon J. Bale, and Andy M. Tyrrell. Panda: A reconfigurable architecture that adapts to physical substrate variations. IEEE Transactions on Computers, 62(8):1584–1596, 2013.

[20] Javier Soto, Juan Manuel Moreno, and Joan Cabestany. A self-adaptive hardware architecture with fault tolerance capabilities. Neurocomputing, 121(dec.9):25–31, 2013.

[21] Tingpeng Li, Yanling Qian, and Li Yue. Bus-structure-based embryonic bio-inspired hardware technology. In 2012 4th International Conference on Intelligent Human-Machine Systems and Cybernetics, volume 2, pages 157–161, 2012.

[22] Michael Reibel Boesen and Jan Madsen. Edna: A bio-inspired reconfigurable hardware cell architecture supporting self-organisation and self-healing. In Proceedings of the 2009 NASA/ESA Conference on Adaptive Hardware and Systems, AHS '09, page 147–154, USA, 2009. IEEE Computer Society.

[23] Csaba Szasz and Adrian Cioloca. Two-layer coarse-fine-grid network model for bio-inspired computing systems development. In 2013 17th International Conference on System Theory, Control and Computing (ICSTCC), pages 515–520, 2013.

[24] S. Kim, H. Chu, I. Yang, S. Hong, S. H. Jung, and K.-H. Cho. A hierarchical self-repairing architecture for fast fault recovery of digital systems inspired from paralogous gene regulatory circuits. IEEE Transactions on Very Large Scale Integration Systems, 20(12):p.2315–2328, 2012.

[25] Yen Lin Huang and Chin Lung Lu. Sorting by reversals, generalized transpositions, and translocations using permutation groups. Journal of Computational Biology, 17(5):685–705, 2010.

[26] Gareth A Jones. Primitive permutation groups containing a cycle. Bulletin of the Australian Mathematical Society, 89(1):159–165, 2014.

[27] Anthony N. Michel and Ling Hou. Stability of dynamical systems with discontinuous motions:. Sice Journal of Control Measurement & System Integration, 1(6):411–422, 2011.

[28] S. V. Emel'yanov, Yu. E. Danik, M. G. Dmitriev, and D. A. Makarov. Stabilization of nonlinear discrete-time dynamic control systems with a parameter and state-dependent coefficients. Doklady Mathematics, 93(1):121–123, 2016.

[29] Hongsheng Chen, Zheng Zhong, Zhiwei Chen, and Xiaotao T. Bi. Simulation of flow and heat transfer around a heated stationary circular cylinder by lattice gas automata. Powder Technology, 290:72–82, 2015.

[30] Haoming Wang, Haibo Zhao, Zhaoli Guo, Yongxiang He, and Chuguang Zheng. Lattice boltzmann method for simulations of gas-particle flows over a backward-facing step. Journal of Computational Physics, 239:57–71, 2013.

[31] D. Halliday, R. Resnick, and J. Walker. Fundamentals of physics. Physics Today, 25(4):53–54, 2003.