Received March 17, 2020, accepted March 28, 2020, date of publication April 2, 2020, date of current version April 16, 2020.

Digital Object Identifier 10.1109/ACCESS.2020.2984950

Hardware Evolution Based on Improved Simulated Annealing Algorithm in Cyclone V FPSoCs

QIANYI SHANG, (Student Member, IEEE), LIJUN CHEN, (Member, IEEE), JING CUI, (Member, IEEE), AND YAO LU, (Member, IEEE)
Equipment Simulation Training Center, Army Engineering University of PLA, Nanjing 210000, China

Corresponding author: Lijun Chen (lijunchen2001@163.com)

This work was supported in part by the National Natural Science Foundation of China under Grant 61271152, Grant 61602505, and Grant 57377170.

ABSTRACT Evolvable hardware (EHW) is an emerging area of research that uses evolutionary algorithms (EAs) to construct circuits without manual intervention. However, this technique confronts two major issues: the evolution efficiency of structures and the computational efficiency of EAs. To address these issues, we construct a novel virtual reconfigurable circuit (VRC) based on artificial neural network (ANN) architecture and develop a promising EA based on improved simulated annealing (ISA) for the research of EHW. Here, ISA escapes the local optimization through an inner loop and expands new search space through an outer loop. Furthermore, several strategies are proposed for the representation, perturbation and selection of solutions (individuals) to reduce the computational burden throughout the entire evolution process. The use of an Intel Cyclone V field programmable system-on-chip (FPSoC) further accelerates the execution of the algorithm and provides designers with much more high-level processing power than that provided by soft-cores. The obtained results show that the proposed method achieves high-speed processing and improves computational efficiency. The results also show that the method has good flexibility and scalability.

INDEX TERMS Evolvable hardware, artificial neural networks, improved simulated annealing, algorithm design and analysis, FPSoC.

I. INTRODUCTION

Evolvable hardware (EHW) [1] is a novel technology for the design of circuits that uses inspiration from biology to perform self-reconfiguration that enables devices to dynamically adapt to changing environments. A large number of EHW research studies involve programmable architectures (or reconfigurable devices) [2] and evolutionary algorithms (EAs) [3], [4].

As one of the key factors of EHW, the most commonly used reconfigurable devices in the past fewrecent decades are field programmable gate arrays (FPGAs) [5]. However, an important limitation of the dedicated reconfiguration of commercial FPGAs is that the configuration bit stream is complicated. To eliminate limitations of the evolution platform on a specific FPGA, a virtual reconfigurable circuit (VRC) is presented. In the VRC method, a virtual reconfigurable hardware layer is developed upon FPGA, forming a programmable architecture for EHW [6], as shown in Fig. 1.

However, due to the complexity and variety of evolution circuits in practical applications, it is difficult to build a general and flexible VRC. Artificial neural networks (ANNs) can emulate the physical structures used in brains to process information based on powerful parallel operations [7]. Their low-powered, highly parallel, fault-tolerant computing and extensibility may help the evolution structure.

In general, a typical platform for FPGA-based evolvable hardware systems requires a computer as a processor. With the rapid development of FPGAs, it has become possible to construct complete hardware evolution directly on a single-chip digital system based on a field programmable system-on-chip (FPSoC) [8], [9]. FPSoCs open the door for the application of EHW due to these powerful embedded processors (mainly ARM cores).

As another essential factor of EHW, the EA is a significant aspect that influences the evolution efficiency and the speed

The associate editor coordinating the review of this manuscript and approving it for publication was Haruna Chiroma.

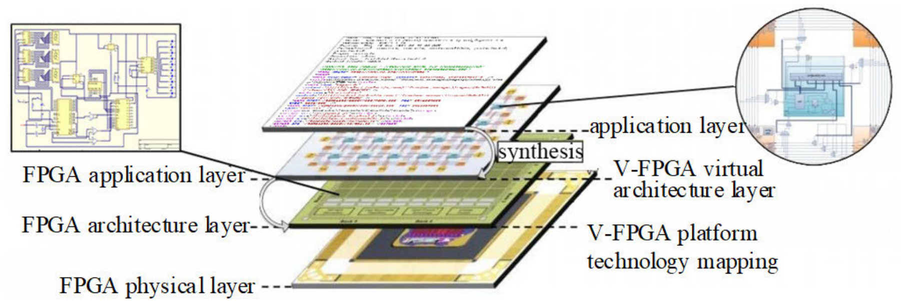


FIGURE 1. Layer model of the VRC approach.

of convergence and thus has become one of the focuses in EHW research. EHW uses bio-inspired methods, e.g., genetic algorithms (GAs) [10], genetic programming (GP) [11], and evolution strategies (ESs) [12] etc., to design hardware. These algorithms are formed by a combination of simple rules restricting and randomness that mimic natural phenomena, also known as meta-heuristic algorithms.

The simulated annealing (SA) inspired by physical phenomena is also a type of meta-heuristic algorithm. SA is easy to implement, provides excellent solutions over meta-heuristic approaches and has very good local-based exploration. These characteristics have also motivated researchers to adopt SA in other optimization problems. However, SA needs to accurately calculate objective function values and requires a large amount of computation time, which limits its application in EHW.

The main focus of this paper is to present a flexible reconfigurable architecture based on ANN architecture using an improved SA (ISA) algorithm for the implementation of the evolution of combinational logic circuits. ISA achieves fast search through an inner loop and an outer loop. Furthermore, several improved strategies have been proposed in this paper. First, ISA uses a new representation method to represent a configuration bit string. Then, to speed up the local search in the inner loop, we add three perturbation operators to generate a new configuration bit string. To reserve the optimal individual in the evolution process, we incorporate the elitist strategy to accelerate the evolution of the ISA. Finally, a perturbed operator is used in the outer loop to explore different search spaces, thereby achieving a global search. ISA has been implemented and tested on a new evolution platform called Intel Cyclone V FPSoC. Experimental results show that ISA can realize circuit evolution based on FPSoC for EHW research. Overall, our proposed methods can effectively improve the evolution efficiency problem and increase the successful rates of evolution.

II. RELATED WORKS

Generally, the EHW refers to a hardware system that implements real-time dynamic reconstruction of programmable logic devices by mimicking biological evolution mechanisms. It can be simply expressed as: EHW = reconfigurable device + EA, where the reconfigurable device includes the reconfigurable platform and the reconfigurable architecture.

A. RECONFIGURABLE PLATFORM

Generally, the reconfigurable platforms used for EHW research include field programmable transistor array (FPTA), field programmable analog array (FPAA), and the most commonly used FPGA [13]. The platform has an important impact on the scale and efficiency of evolution.

The categories of EHW can be extrinsic EHW [14] or intrinsic EHW [15], depending on whether the configuration bit stream is evaluated on its hardware or offline with a simulator. The internal EHW can be used away from the PC and does not require external manual intervention, which is more conducive to its application. The EA is implemented to generate candidate solutions directly on the platform and find the best candidate to directly configure the reconfigurable core. Typically, the EA can be run in an embedded processor (ARM, MicroBlaze, PowerPC, etc.), and these powerful embedded processors enable FPGAs to evolve from simple programmable to very powerful FPSoC platforms.

FPSoC implementations offer many significant advantages for EHW's single-chip implementations, such as reduced power consumption and improved communication efficiency and reliability. In [16], the authors analyzed the Zynq-7000 FPSoC platform, which contains an XC7Z020 device, to evaluate real-world systems. Another application used Zynq-7000 as a new evolution platform, such as the study of reconfiguration architecture in [17], and the high performance of evolution in [18].

However, EHW has only a few works on this topic due to the need for researchers with extensive expertise (knowledge) and practice in both the software and hardware domains, and most of these works address Xilinx Zynq-7000 devices [19]. For such reasons, we adopted Cyclone V FPSoC as a new evolution platform for evaluating the performance of the same application in EHW.

B. VRC-BASED RECONFIGURABLE ARCHITECTURE

Currently, the main way to modify the reconfiguration for EHW research is based on a VRC [1]. Designers achieve complete control of the reconfigurable architecture model by building a virtual layer on top of the device fabric. Common VRC-based circuit evolution consists of a number of basic processing elements (PEs) that are interconnected in a specific manner.

Initially, the virtual evolvable hardware FPGA bridge was proposed by Haddow and Tufte [20] as a genotype/phenotype mapping. This representation method opened up a new area for EHW research. Later, the real VRC-based evolution was proposed and implemented by Sekanina [21]. The VRC evolution model is mainly composed of m × n PE units composed of a matrix array. The VRC, described using Verilog HDL, is depicted, synthesized and downloaded to form a virtual layer on the programmable device.

In practice, a VRC designs the required matrix structure for the needs of the given application. For example, a VRC builds a multi-layer array structure through configurable functional elements (FEs), with the outputs of the previous layer in each layer being the inputs to the next layer, where the FE is implemented by 4 kinds of logic functions [22]. The connections and inputs between the FEs are determined by the genome string generated by the EA.

Another VRC architecture uses a 4 × 6 array structure with a total of 25 PEs, each of which is implemented by multiplexers and function generators [23]. A more recent report on VRC is in [24], which discusses the problem of synthesis of combinational logic circuits at the gate-level. It is also important to mention that these reports are still based on FPGA from the Xilinx Company.

The main problems with the methods mentioned above are that the interconnection topologies may still be too complex, thus reducing the flexibility and scalability of the fabric and slowing down the reconfiguration. An easy way to obtain better results involve integrating an ANN into the research of reconfigurable structures. The main attribute of the ANN is parallel processing, which is very beneficial for improving the evolution efficiency and the scale of evolution. In [25], two cascaded FPAAs were used to create an ANN layer for the hardware evolution of in-situ robotic fault-recovery. Recently, research on ANN-based VRC was discussed in [13], mainly using an improved VRC to optimize evolutionary fault tolerance methods. This report was based on the extrinsic evolution of a Xilinx Virtex SRAM-based FPGA.

C. EVOLUTIONARY ALGORITHMS

Most of the EAs currently in use show quick progress at the beginning, but this progress is slowed down at the end [26]. For example, the most classic and commonly used GA is easy to converge prematurely, especially for complex problems. Once it occurs, GA stays at the local optimum. Therefore, finding algorithms or methods that prevent premature convergence or jumping out of the local optimum may improve the efficiency of the EA.

SA was first presented by Kirkpatrick *et al.* [27] as a method for complex problems. After decades of research, because of its breadth of applicability, ease of implementation. In the past decades, SA has been widely reported and has made significant achievements in solving complex optimization problems, such as TSP [28], the preventive maintenance problem [29], and the optimization of high-dimensional problems [30]. The main novel concept in our study is to use optimization algorithms for the simplification problem in analog circuits. One of the novel concepts in this paper is to use the improved SA to develop an evolutionary system for the design of combined circuits, which solves the problems that require a large number of computations and evaluations during the search process.

III. ANN-BASED RECONFIGURABLE ARCHITECTURE AND HARDWARE IMPLEMENTATION

A. ANN

The ANN is a special complex architecture that simulates the nerve cell of the brain with a very simple structure and functions as the basic unit (called node or neuron) to form a network structure with high reliability and high plasticity through a high degree of interconnection. Its highly parallel, low-powered, fault tolerant computing and extensibility are very attractive for evolutionary architecture applications. Integrating the ANN structure into the evolutionary architecture may offer good results.

Among different categories of ANN, feed-forward neural network is the most common. It is organized in layers and which consists of multiple neurons, each of which is connected to the previous layer with various weights [31], as shown in Fig. 2. Numerical input signals enter the input layer on the left side of the network and propagate through the middle layers to the right and output to the output layer on the right. This is similar to our evolutionary structure. Mapping and combining the ANN architecture with the reconfigurable structure can make the evolutionary structure more flexible and control the illegal connections, so it is a popular platform for both research and deployment [32].

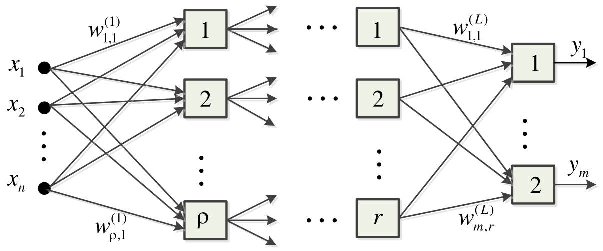


FIGURE 2. An example of a feed-forward neural network.

B. ANN-BASED EVOLVABLE ARCHITECTURE

1) BASIC UNITS OF ANN-BASED RECONFIGURABLE ARCHITECTURE

The programmable element (PE) unit is the main programmable unit of the evolutionary model. A PE unit usually includes two elements, an input unit (IU) and a function unit (FU), as shown in Fig. 3. The elements of an IU or FU have multiple inputs and one output. The number of inputs

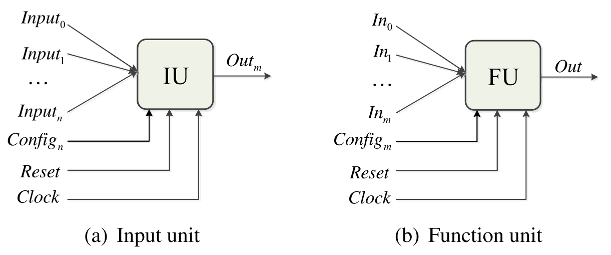


FIGURE 3. Two components of the ANN-based VRC evolutionary model.

depends on the input of the evolution circuit and the size of the evolution matrix. In the structure of an IU or FU, the function of each signal is as follows: configuration signal *Config* is used for actually writing configuration data; configuration clock signal *Clock* is used for writing configuration data in the same clock; configuration reset signal *Reset* is used to reset the configuration register. The IU determines the number of inputs and outputs by the specific circuit and selects the input signal through the multiplexer. The FU determines the sequential logic and combinational relationships that multiple inputs must realize. In this article, each FU implements a simple logical function on the output of IU, such as addition, subtraction, XOR, identity, constant value, etc., and then uses the output as the input of the next hidden layer or the final output.

2) THE VRC UNIT OF ANN-BASED RECONFIGURABLE ARCHITECTURE

The connection between multiple PE units constructs an ANN-based VRC array. Each layer receives the system input signal or the output signal of the previous layer unit, performs some logic processing on it, and then outputs the result to the next layer unit, as shown in Fig. 4. According to the structure of the PE unit, the multiplexers *mulA*, *mulB* and *mulY* are used in the application for signal selection. F_k is the type of logical function. The number of F_k and the type of each F_k can be designed autonomously through the function of the evolutionary circuit, which also has a great impact on the evolution efficiency. However, for a virtual array, it should be noted that the connection weights between the network nodes of each layer of the ANN-based structure reflect only the connection relationship of the nodes, and the weight is only 0 or 1 (0 means disconnected; 1 means connected).

C. FPSOC-BASED IMPLEMENTATION

The overall architecture of the evolution system is shown in Fig. 5. The entire system is completed on a single FPSoC chip and is completely free of reliance on the PC. The entire system consists of two parts, the Hard Processor System (HPS) and the ANN-based VRC structure. The HPS includes a dual-core ARM Cortex-A9, on-chip RAM, ROM, internal interconnect and bus interface and writes configuration data through the GPIO interface to implement reconfiguration of the VRC array. First, the ISA software is executed

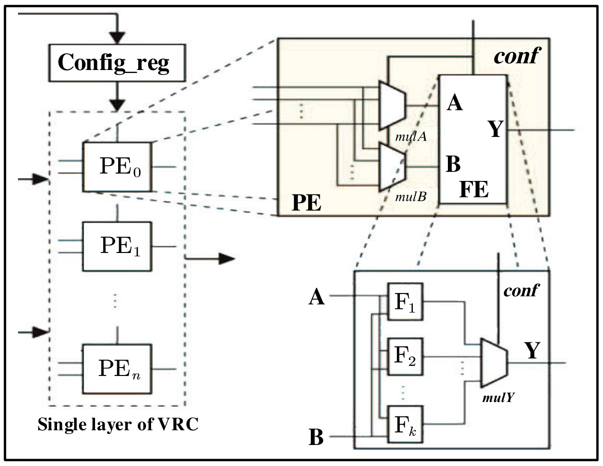


FIGURE 4. A single virtual layer architecture.

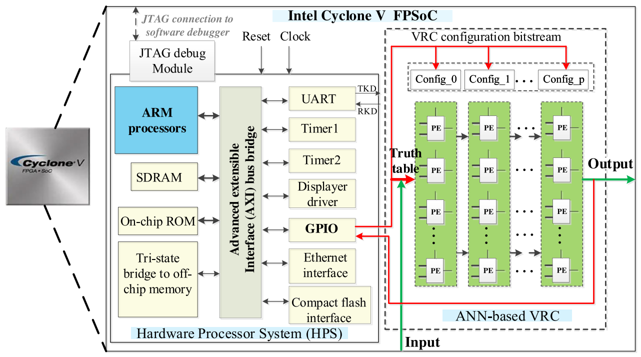


FIGURE 5. System block diagram of the evolution system.

on the ARM processor and generates multiple candidate solutions (chromosomes). To calculate the fitness value of each candidate solution, the processor configures the ANN-based VRC model using new configuration bits created based on the candidate solution through the FPSoC internal bus. Then, the desired input vector is input into the completed configuration circuit, and the response is compared with the truth table output. Finally, regenerate new neighborhood candidate solutions to find better candidate solutions. Iteratively run the ISA until the termination criterion is reached.

IV. THE THEORETICAL FUNDAMENTALS OF SIMULATED ANNEALING ALGORITHM

In attempts to find the optimal solution using EAs, it is seen that the demanded generation number of evolution increases very quickly when the number of searches increases, and the advantage of the EA will not continue to evolve because it becomes trapped in a local optimal value. SA is a stochastic global optimization method that simulates the process of physical annealing with solids [30]. The solid is cooled slowly after heating to a certain temperature. When heated, the internal structure of the solid becomes disordered with increasing temperature, and the internal energy increases, while the internal structure gradually becomes orderly when cooled slowly and finally reaches an equilibrium state at normal temperature.

SA establishes the connection between thermodynamic behavior and the search optimization algorithm and chooses to accept or reject inferior solutions through an acceptance probability criteria. The probability p of accepting an inferior solution gives the SA algorithm some ability to escape from local optimum. The most commonly used probability function is shown as follows:

$$p = \begin{cases} e^{\left(\frac{-\Delta f}{T_k}\right)}, & \text{if } \Delta f > 0 \\ 1, & \text{otherwise} \end{cases} \quad \text{for} \quad T_k > 0 \tag{1}$$

where T_k is an adjustable parameter representing the temperature at each iteration, Δf is the difference between the energy function or fitness function of two neighbors' solutions in a given function or application. X is the current solution, and its fitness value is denoted by $f(X)$. After the update, a new candidate solution X' is found, and its fitness value is $f(X')$. The relative change in fitness value a between X and X' is defined as:

$$\Delta f = f(X) - f(X') \tag{2}$$

Moreover, the function *random* is used to generate a uniform random number r between 0 and 1. If p is greater than r, the new solution will be accepted to replace the former one.

Generally, SA is mainly composed of six parts: 1) objective function, 2) initial condition, 3) choice of neighborhood, 4) acceptance function, 5) cooling schedule and 6) termination condition. The procedure of SA algorithm is shown in Algorithm 1.

Algorithm 1 A Pseudocode of SA Algorithm

```
1: Initialize the parameters: temperature $T_0$, initial solution,
   cooling rate $\alpha$, time step $k$.
2: Generate an initial solution $X$ randomly.
3: while the termination criterion is not meeted do
4:     Generate a new solution: $X'$
5:     if $f(X')>f(X)||p = exp(-\Delta f/kT) > r$
6:         $X = X'; f_{new} = f(X')$
7:     end if
8:     Reduce the temperature using $T_k = (\alpha^k)T_0; k = k + 1$
9:     Update $X_{best}$
10: end while
11: Return $X_{best}$
```

SA maintains excellent robustness and efficiency. It is easy to implement and can also run with lower memory requirements. These characteristics have motivated the use of SA in several engineering problems [33]. Although the SA algorithm is effective in solving optimization problems [34], its slow convergence rate and the requirement for too much computational effort have also limited its implementation in applications that require high real-time performance such as EHW. In EHW, using the pure SA algorithm, evolution will stall when the search process approaches an approximate optimal solution. To this end, the SA algorithm needs to be improved and then applied to circuit evolution.

V. IMPROVED SIMULATED ANNEALING FOR THE EHW PROBLEM

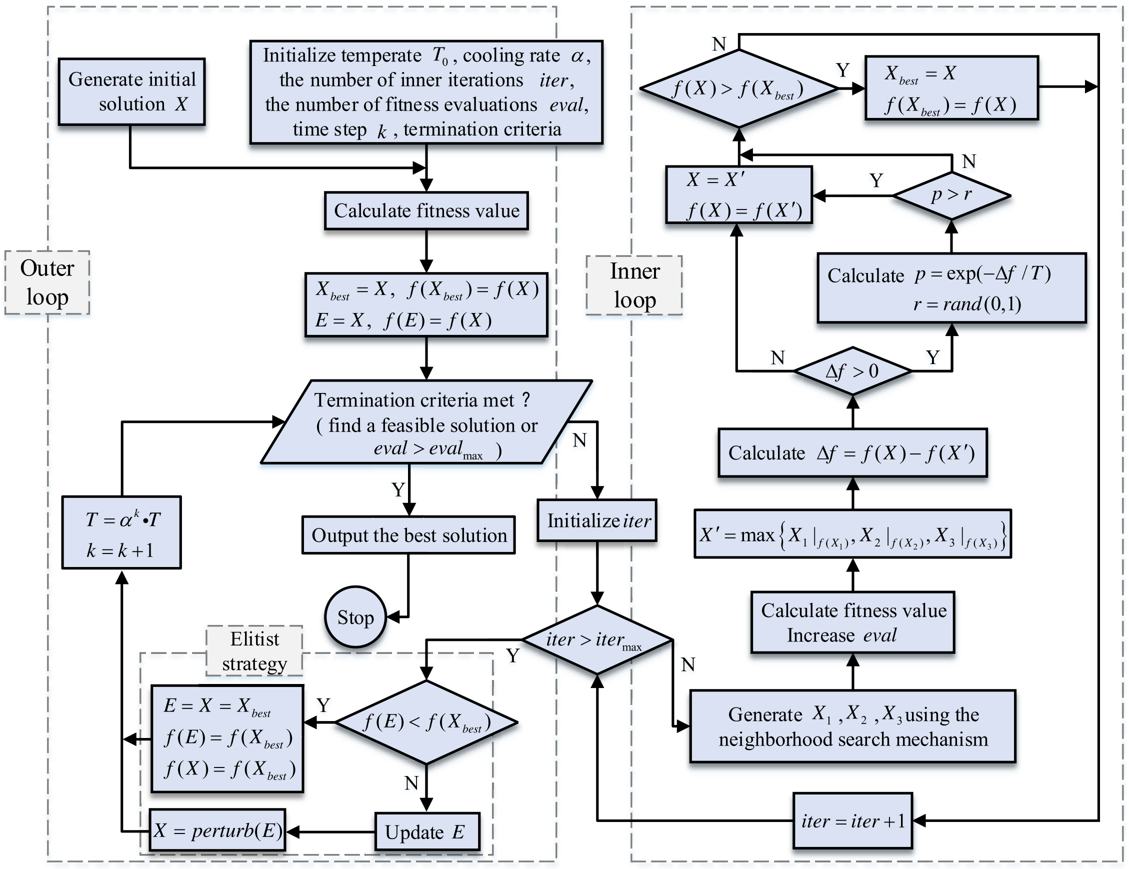
In this article, we are committed to developing an effective ISA approach for the EHW design of combinational logic circuits using Cyclone V FPSoCs. The algorithm proposed here was run in a single chip powerful hard-processor cores, using the two-loop (i.e., outer loop and inner loop) iteration to accelerate the evolution. In this algorithm, the solution representation, the neighborhood search, and the selection of the best solution are innovated. In this regard, it can be considered as a new EA. The pseudo-code of ISA is described in Algorithm 2, and its flowchart is provided in Fig. 6.

Algorithm 2 Pseudocode for the Proposed ISA Algorithm

```
1: Initialize the parameters: temperature $T_0$, initial solution, maximum iteration $iter_{max}$, maximum evaluation
   $eval_{max}$, cooling rate $\alpha$, time step $k$.
2: Generate an initial solution $X$ randomly.
3: $X_{best} = X; f(X_{best}) = f(X)$;
   $E = X; f(X_{best}) = f(X)$
4: while the termination criterion is not satisfied do
5:     Initialize $iter$
6:     while $iter \leq iter_{max}$ do
7:         Create $X_1, X_2, X_3$ using the neighborhood search
           mechanism
8:         $X' = max\left\{X_1|_{f(X_1)}, X_2|_{f(X_2)}, X_3|_{f(X_3)}\right\}$
           // Select the candidate solution with the highest
           fitness value as $X'$
9:         if $f(X')>f(X)||p = exp(-\Delta f/kT) > r$
10:            $X = X'; f(X) = f(X')$
11:        end if
12:        Update the best solution $X_{best}$    // Elitist strategy
13:        Increase increment $iter$
14:    end while
15:    Return $X_{best}$
16:    if $f(E) < f(X_{best})$
17:        $E = X = X_{best}; f(E) = f(X) = f(X_{best})$
18:    else
19:        Update $E$    // Updated to E used in the former
       outer loop. This is also an elitist strategy
20:        Set $X = perturb(E)$
21:    end if
22:    Reduce the temperature using $T_k = (\alpha^k)T_0$
23:    Increase increment $k$
24: end while
25: Output the best solution
```

The main steps of the ISA algorithm for circuit evolution are explained as follows:

1) Initialize the parameters of ISA and generate an initial solution. These parameters are initial temperature (T_0),



Outer loop
Generate initial solution X
Initialize temperate T0, cooling rate α, the number of inner iterations iter, the number of fitness evaluations eval, time step k, termination criteria
Calculate fitness value
X_best = X, f(X_best) = f(X)
E = X, f(E) = f(X)
Termination criteria met ? (find a feasible solution or eval > eval_max)
Y
N
Output the best solution
Stop
T = α^k•T
k = k+1
Elitist strategy
E = X = X_best
f(E) = f(X_best)
f(X) = f(X_best)
f(E) < f(X_best)
X = perturb(E)
Update E
Initialize iter
iter > iter_max
Inner loop
f(X) > f(X_best)
X_best = X
f(X_best) = f(X)
p > r
X = X'
f(X) = f(X')
Calculate p = exp(−Δf / T)
r = rand(0,1)
Δf > 0
Calculate Δf = f(X) − f(X')
X' = max{X1|f(X1), X2|f(X2), X3|f(X3)}
Calculate fitness value
Increase eval
Generate X1, X2, X3 using the neighborhood search mechanism
iter = iter + 1

FIGURE 6. Flowchart of the proposed ISA.

cooling rate (α), maximum number of inner iterations ($iter_{max}$), maximum number of fitness evaluations ($eval_{max}$), time step (k), and the perturbed probability (p_m) used in the perturbed operator. An initial solution (X) is generated by the objective function. The provisional optimal solution explored by the outer loop is E.

2) Calculate the fitness value of the current solution X and save it as the best solution, $E = X_{best} = X$.
3) Inner loop. In the Inner loop, the current solution X is perturbed in the vicinity by the neighborhood search mechanism to to yield X_1, X_2 and X_3. Three neighborhood search methods proposed for circuit evolution can quickly generate new candidate solutions, and choose the solution with the highest fitness value as the new solution X'. ISA accepts the new candidate solution X' if it has a better fitness than the current one or decides whether to adopt the inferior solution by acceptance probability criteria of a bad solution.
4) If the fitness value of X turns out to be better than that of X_{best} (i.e., $f(X) > f(X_{best})$), X_{best} will be updated with X.
5) Outer loop. This is also an important step for ISA, which can extend the best solution (X_{best}) obtained in the inner loop to a new search space, thereby compensating for possible missing "gene fragments" on the "chromosome". If the fitness value of E turns out to be lower than X_{best}, then both E and X are replaced by X_{best}. If not, update E to the solution used in the former outer loop, and then perform a perturbed operator on E.
6) The temperature is cooled, and the previous step is continued to obtain a more accurate solution, while the probability that the acceptance function accepts an inferior solution is also lower due to the decrease in temperature.
7) As long as the computation time is long enough, the objective function can always converge to the globally optimal solution.

In the rest of this section, we will introduce the components of ISA for the EHW problem.

A. SOLUTION REPRESENTATION

High quality performance of a meta-heuristic can be intensely associated with selecting a suitable solution representation [35]. To further improve the speed and performance of evolution, we designed a binary matrix group encoding method based on the proposed evolvable system.

1) THE LOGICAL FUNCTION CONFIGURATION OF EACH PE

In the ANN-based structure, what kind of logic function the PE implements is represented by a binary column vector. For example, the j^{th} logical functions of the p^{th} layer are encoded using the column vector $\left[l_{1,j}{}^{[p]}, l_{2,j}{}^{[p]}, \cdots l_{i,j}{}^{[p]}, \cdots l_{t,j}{}^{[p]}\right]^T$, where $t = \lceil log_2 M\rceil$, M is the amount of logical functions, and p is the number of layers of the ANN-based VRC array. The function codes are arranged to form a matrix, which is expressed as follows:

$$L^{[p]} = \begin{bmatrix} l_{1,1}{}^{[p]} & l_{1,2}{}^{[p]} & \cdots & l_{1,j}{}^{[p]} & \cdots l_{1,s}{}^{[p]} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ l_{i,1}{}^{[p]} & l_{i,2}{}^{[p]} & \cdots & l_{i,j}{}^{[p]} & \cdots l_{i,s}{}^{[p]} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ l_{t,1}{}^{[p]} & l_{t,2}{}^{[p]} & \cdots & l_{t,j}{}^{[p]} & \cdots l_{t,s}{}^{[p]} \end{bmatrix}$$
$$1 \le i \le t \quad 1 \le j \le s \quad \forall l_{i,j}{}^{[p]} \in \{0, 1\} \qquad (3)$$

where s is the number of nodes (i.e., the number of PE) in the current layer,

2) THE CONNECTION CONFIGURATION OF EACH PE

In the ANN-based structure, the connection relationship between the j^{th} PE of the p^{th} layer and the output of the $(p-1)^{th}$ layer array uses a connected column vector $\left[w_{1,j}{}^{[p]}, w_{2,j}{}^{[p]}, \cdots w_{i,j}{}^{[p]}, \cdots w_{s',j}{}^{[p]}\right]^T$, where s' is the number of nodes in the previous layer. The connection codes are arranged to form a matrix, which is expressed as follows:

$$W^{[p]} = \begin{bmatrix} w_{1,1}{}^{[p]} & w_{1,2}{}^{[p]} & \cdots & w_{1,j}{}^{[p]} & \cdots w_{1,s}{}^{[p]} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{i,1}{}^{[p]} & w_{i,2}{}^{[p]} & \cdots & w_{i,j}{}^{[p]} & \cdots w_{i,s}{}^{[p]} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{s',1}{}^{[p]} & w_{s',2}{}^{[p]} & \cdots & w_{s',j}{}^{[p]} & \cdots w_{s',s}{}^{[p]} \end{bmatrix}$$
$$1 \le i \le s' \quad 1 \le j \le s \quad \forall w_{i,j}{}^{[p]} \in \{0, 1\} \qquad (4)$$

3) THE CONFIGURATION OF EACH LAYER

To facilitate storage and operation, the logical function column vectors of each PE are spliced together with the connection column vectors to uniquely determine the type of function that each PE can implement and its connection relationship with the previous layer array. Then, the configuration codes of each PE in the layer (i.e., $P^{[p]}$) are arranged to form a matrix, which is expressed as follows:

$$P^{[p]} = \begin{bmatrix} l_{1,1}{}^{[p]} & l_{1,2}{}^{[p]} & \cdots & l_{1,j}{}^{[p]} & \cdots l_{1,s}{}^{[p]} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ l_{t,1}{}^{[p]} & l_{t,2}{}^{[p]} & \cdots & l_{t,j}{}^{[p]} & \cdots l_{t,s}{}^{[p]} \\ w_{1,1}{}^{[p]} & w_{1,2}{}^{[p]} & \cdots & w_{1,j}{}^{[p]} & \cdots w_{1,s}{}^{[p]} \\ \vdots & \vdots & \vdots & \vdots & \vdots \\ w_{s',1}{}^{[p]} & w_{s',2}{}^{[p]} & \cdots & w_{s',j}{}^{[p]} & \cdots w_{s',s}{}^{[p]} \end{bmatrix} \qquad (5)$$

By sequentially encoding each layer, a binary matrix group encoding $P = \{P^{[1]}, P^{[1]}, \cdots, P^{[p]}\}$ can be obtained, where P is the solution encoding, p is the number of layers of the ANN-based VRC array. For example, Fig. 7 shows the $P^{[p]}$ used in the evolution of 2-bit multiplier circuit design. As aforementioned in encoding method, the first two lines of the matrix performs functional configurations, the other rows are the connection configuration.

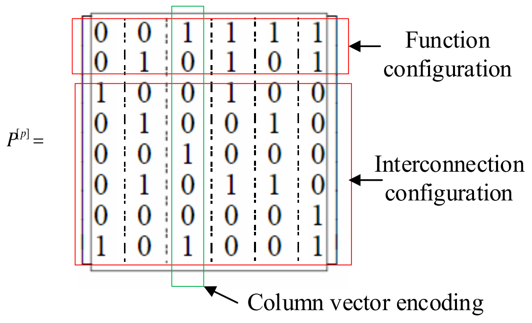


FIGURE 7. An example of encoding method.

The advantages of describing the ANN-based VRC structure representation as matrix group encoding mainly include the following two aspects:

1) Easy to store and operate. Adopting the matrix group encoding method is conducive to improving the programming implementation of the ISA and the implementation of perturbation in the neighborhood.
2) Efficient search in search space. By perturbing a row or a column in the matrix, a wide range search in search space can be achieved, which is beneficial to improving the diversity and feasibility of candidate solutions.

B. OBJECTIVE FUNCTION

Our goal is to evolve combinational logic circuits with 100% functionality, i.e., to find a feasible solution that satisfies the requirements of fitness value. After finding a feasible circuit, the ISA algorithm completes the configuration of the circuit and stops evolution.

C. FITNESS EVALUATION

In our work, we used simple truth table matching as a measure of the fitness value of our chromosome. After the VRC structure is configured with the chromosome, enter the input value of the desired function. If the actual output matches the corresponding output of the truth table, the value is increased by one. This is done in turn on all the inputs listed in the truth table to obtain the fitness value for the chromosome. The fitness function is shown in Eq. (6).

$$f(X_i) = \sum_{j=1}^{n}(f_j), (f_j \in \{0, 1\}). \qquad (6)$$

where $f(X_i)$ is the fitness of the i^{th} chromosome (solution), $i = 1, 2, \ldots,$ N (population size used in GA), and n is the number of entries of the truth table. f_j is the comparison between the actual output and the expected value, the same is 1, otherwise it is 0.

D. ITERATION LOOPS

The optimization framework of ISA shown here mainly includes two loops: outer and inner loops. In the proposed ISA, a cooling schedule, an elitist strategy and a perturbed operator are introduced into the update of the best solution in the outer loop to solve the problem of global search. Moreover, three neighborhood structures, an elitist strategy and an acceptance criterion are introduced into the inner loop to achieve fast convergence.

1) OUTER LOOP

The starting point of the outer loop is to explore new areas of the search space through perturbation (perturbed operator), so as to realize global search. If a better solution X is obtained in the inner loop, the optimal solution E is updated and continued. At the same time, the probability of accepting an inferior solution in the inner loop is reduced by a cooling schedule, thereby maintaining the directionality of search. If not, by randomly perturbing the solution E, the search space can be jumped to a new search region. Furthermore, the use of elitist strategy can ensure that when the inferior solution is searched during the outer loop, it is restored to the solution E of the previous loop.

2) INNER LOOP

The inner loop accepts new solutions mainly in accordance with acceptance criterion. The introduction of the acceptance probability criterion can accept the inferior solution probabilistically, so as to jump out of the local optimal solution. In addition, the addition of the elitist strategy can better retain the searched optimal solution, so it is easier to achieve fast search. Moreover, in the neighborhood structure, four neighborhood search methods are proposed for circuit evolution. These methods are very effective in generating new legal circuit configurations. Due to large perturbations in the early stages of SA manages the local search procedure [36], the use of matrix group encoding method can generate large perturbations to explore new areas of the search space. By choosing the solution with the highest fitness among the newly generated solutions (i.e., X_1, X_2, X_3) as the new solution X, the search efficiency can be further improved.

3) PERTURBED OPERATOR

The function of the perturbed operator is to perturb the current solution E to produce a new solution. This perturbation method uses the idea of mutation method in GAs to generate new gene segments through mutation. Here, the basic bit mutation operation is performed on bit-by-bit basis, according to the mutation probability (p_m), to obtain a new encoding string by mutating one or several configuration bits. It should be noticed that the value of p_m cannot be set too high, otherwise the outer loop will be reduced to random search. Conversely, a low value of p_m may make the effect of the mutation operation less obvious, which also loses the significance of the perturbed operator in navigating the search space to the new search region.

4) NEIGHBORHOOD STRUCTURE

The ISA adopts a probabilistic perturbation operation in the Monte-Carlo hill-climbing searching process to jump out of the local optimal trap and reach a global optimum. There are many ways to generate neighborhood solutions of the current solution. In this paper, the target of neighborhood structure generation is to generate a neighboring solution for its current solution using new operators such as shift, swap, and reverse. Fig. 8 shows examples for each of three mechanisms. Using the shift operator, a row is randomly selected for the change (gene segment C_4 moved from third to fifth). Using the swap operator, randomly select a segment of the chromosome to reverse its order (the order of gene segments on the chromosome changed from $N_2N_3N_1$ to $N_1N_3N_2$). In a reverse operator, the randomly selected segment itself is reversed to produce a new segment (gene segment on chromosome reversed from C_3 to P_3, from N_3 to P_3, from C_2 to P_2).

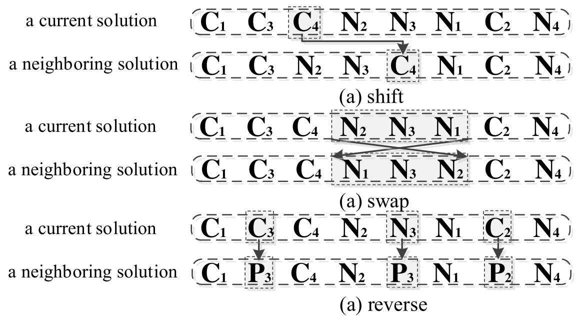


FIGURE 8. **Example of solution generation operators: (a) shift operator, (b) swap operator, (c) reverse operator.**

In each iteration of the inner loop, each perturbation operation (shift, swap, and reverse operator) is used to generate three new solutions (i.e., X_1, X_2, X_3) in turn from the current solution. Then, select the solution with the highest fitness of the three new solutions as the neighboring solution.

E. COOL SCHEME

The performance of the SA is important relative to the cooling schedule [30]. Here, the cooling schedule of the ISA is used to control the speed at which the algorithm converges. The common geometric cooling method is applied in this paper, and the formula of the cooling schedule is:

$$T_k = (\alpha^k)T_0 \tag{7}$$

where T_0 is the initial temperature, k is the time step, and α is the cooling rate ($0.8<\alpha<1$).

F. ELITIST STRATEGY

In inner loop, a current solution X is saved as an optimal solution at the beginning of the iteration loop, and if a modification exhibits better fitness, the optimal solution X_{best} is replaced. After the end of the inner loop, determine whether to replace the provisional optimal solution E in the outer loop with the best solution X_{best} obtained in the inner loop, or update E to a historical record value, and restart the iteration loop after updating temperature.

VI. EXPERIMENTAL RESULTS

A. PLATFORM OF THE EHW EXPERIMENT

The performances of the proposed methods have been tested on a Terasic DE1-SoC board including an Intel Cyclone V FPSoC (5CSEMA5F31C6) [37]. The HPS processor frequency was set to 925 MHz. The proposed ANN-based VRC structure successfully modeled with Verilog HDL. In these experiments, all algorithm software is written in C and compiled and linked using the ARM DS-5 Intel FPGA edition. Functional integrity testing of the feasible circuits is simulated and verified in ModelSim.

B. METRICS

In the EHW system, the metrics for quantifying reconfigurable architectures and EAs are diverse. Since most of the previous researches used Xilinx FPGAs, this article innovatively uses Intel FPGA. Therefore, the differences in evolutionary platforms make it impossible to simply compare results with previous researches in terms of typical performance metrics such as speed or quality of the solutions found. In addition, the main idea of this work is to present the ISA algorithm as a tool for the design of combinational logic circuits.

To inspect the performance of the ISA algorithm, the most commonly used genetic algorithm (GA) and pure SA in the EA have been taken as a reference algorithm for the purpose of comparison. The metrics of interest will be as follows:

1) Success rate: the number of times the method finds a feasible circuit within the termination condition.
2) Average fitness: average of the highest fitness in multiple trials.
3) Evaluations: the average number of fitness function evaluations required to find a feasible circuit.
4) Convergence analysis: The variation of the fitness function evaluations with the average fitness of the solution is pursue as a standard of quantifying the convergence.

C. SIMULATION AND RESULTS

To investigate the effectiveness of the ANN-based structure and proposed algorithm, we used the system to evolve basic combinational logic circuits for different benchmark Boolean problems. The four benchmark Boolean questions presented in this paper are shown in Table 1. The ADD1 and ADD2 are 1-bit and 2-bit binary full-adders respectively. The MUL2 is a 2-bit binary multiplier. The CHE is a 4-bit even parity checker that consists of four inputs and one output.

The performance of the optimization algorithm depends to some extent on the values of its parameters, which are the setting of the initial value, the selection of the probability value and the termination condition. The selection of parameter values is difficult to obtain through a uniform principle or formula. To determine these values, massive experiments and comparisons are required.

TABLE 1. Benchmark boolean problems. N_i, N_o, N_r and N_e denote the numbers of inputs, outputs, rows, and entries of the truth table, respectively.

Name	N_i	N_o	$N_r = 2^{N_i}$	$N_e (= N_r \times N_o)$
CHE	4	1	8	8
ADD1	3	2	8	16
MUL2	4	4	16	64
ADD2	5	3	32	96

In the following experiments, we implemented the same experiment with the standard GA and the pure SA to compare the superiority of the proposed algorithm. The parameters of the proposed ISA and SA are presented in Table 2. The parameter settings for GA are listed in Table 3. It should be mentioned that all trials were performed 20 times and the results were averaged, each trial being a different random number seed.

TABLE 2. Parameter values of ISA and SA.

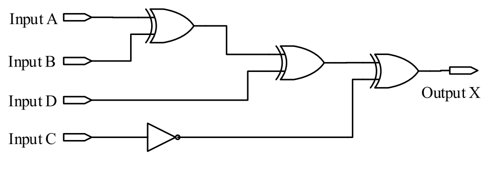
Parameters	ISA	SA
Initialization	Bit random	Bit random
Initial temperature T_0	1000	1000
Cooling rate α	0.99	0.99
Perturbation probability p_m	0.02	
Maximum number of inner iterations $iter_{max}$	100	
Maximum number of fitness evaluations $eval_{max}$	800000	800000

TABLE 3. Setting the parameters of GA.

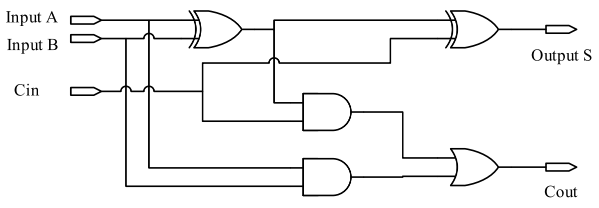
GA parameters	Experimental settings.
Cross probability	0.85
Mutation probability	0.001
Population size	30
Initialization	Bit random
Mutation strategy	Binary swap mutation
Cross way	Single point cross
selection method	Roulette-wheel selection

The circuit diagram of a feasible 4-bit even checker, 1-bit full adder, 2-bit multiplier and 2-bit full adder successfully evolved is shown in Fig. 9. Results reported in Fig. 10 shows the simulation result of the evolved circuit using the ISA algorithm. *Input*[] is the input data of the circuit's truth table, and *result*[] is the actual output data of the successful evolution.

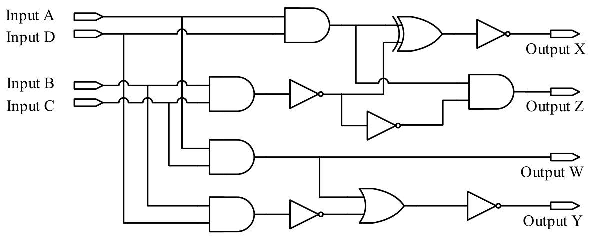
The average evolution results of 20 runs in each individual value of every algorithm are shown in Table 4. In the case of the 4-bit even parity checker and 1-bit binary full-adder, note that the ISA, SA and GA found a feasible circuit to reach their best fitness value, but ISA required significantly fewer fitness function evaluations than GA and SA. In the case of the 2-bit binary multiplexer and 2-bit binary full-adder, ISA was still able to find a feasible circuit to reach the best fitness value



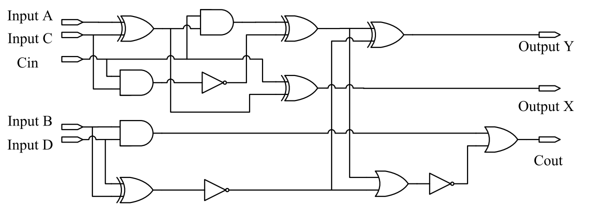
(a) 4-bit even parity checker



(b) 1-bit binary full-adder



(c) 2-bit binary multiplexer



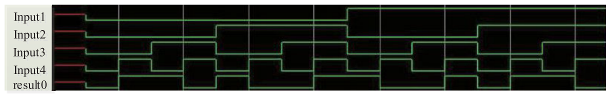
(d) 2-bit binary full-adder

FIGURE 9. Evolved circuit for 4-bit even parity checker, 1-bit full adder, 2-bit multiplier and 2-bit full adder.

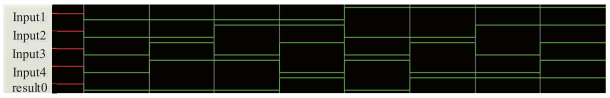
TABLE 4. Comparison of results between a binary ISA, GA and SA for the different target circuits.

Target circuit	Algor-ithm	Success rate	Average fitness	Evaluations
CHE	GA	100%	16	9511
	SA	100%	16	9381
	ISA	100%	16	7578
ADD1	GA	100%	16	103434
	SA	100%	16	114692
	ISA	100%	16	37309
MUL2	GA	23.1%	57.8	410410
	SA	73.4%	62.8	178694
	ISA	100%	64	43648
ADD2	GA	42.86%	74.9	612930
	SA	61.4%	90.2	405873
	ISA	100%	96	213683

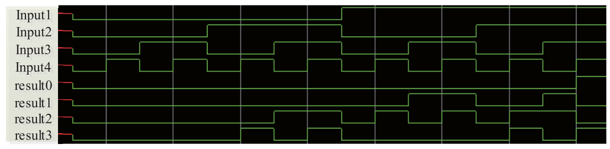
100% of the time, but a feasible circuit was found by the GA only 23.1% of the time and 42.86% of the time, respectively, a feasible circuit was found by the SA only 73.4% of the time and 61.4% of the time, respectively. Also note that, in this



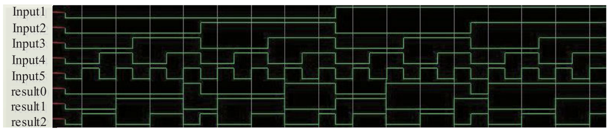
(a) 4-bit even parity checker



(b) 1-bit binary full-adder



(c) 2-bit binary multiplexer



(d) 2-bit binary full-adder

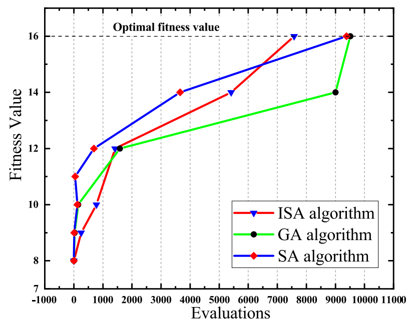
FIGURE 10. The simulation results.

case, the use of the ISA significantly decreased the average number of evaluations required to find a feasible circuit.

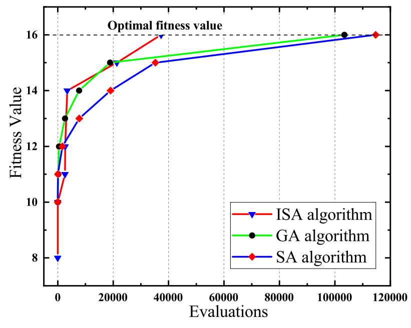
Characterizing convergence speed is one of the most important research contents in the research of EAs for hardware evolution. In this paper, we further analyze the convergence speed of the ISA algorithm in the evolution process and compare it with the other two algorithms. The variation curve of the fitness value with the times of fitness evaluation is shown in Fig. 11 ((a)-(d)). In each image, the abscissa shows the average of the times of fitness evaluation independently executed 20 times, and the ordinate shows the optimal fitness value found in multiple experiments. In Fig. 11 (a) and (b), the optimal fitness value of the evolutionary circuit is 16, that is, the current solution is a feasible solution when the fitness reaches 16. In Fig. 11 (a), although ISA has no clear advantage in the early stage convergence rate, it evolves faster in the later stage, and the number of evaluations required for successful evolution is also the least. In Fig. 11 (b), compared with SA and GA, the ISA algorithm has a faster convergence rate. In Fig. 11 (c), the optimal fitness value is 64. The evolution results show that the convergence speed is very fast in the early stage, but the performance of ISA is more advantageous in the later stage of evolution. In Fig. 11 (d), it should also be noted that the convergence speed of GA and SA is faster in the early stage, while the convergence speed of ISA is faster in the later evolution stage, and the number of evaluations required for its successful evolution (with a fitness value of 64) is also the least.

D. CONTRASTIVE ANALYSIS OF ISA AND STANDARD SA

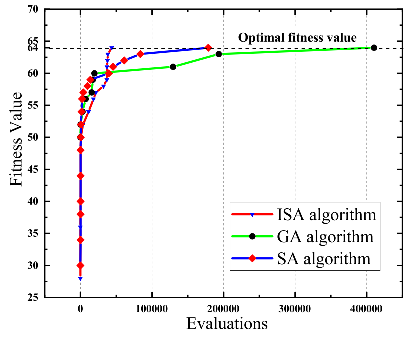
In order to further evaluate the effectiveness and efficiency of the proposed algorithm, the ISA is compared with the standard SA by changing the temperature during the evolution.



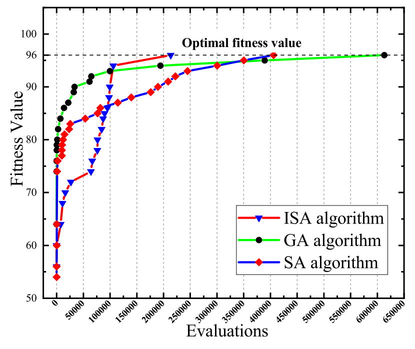
(a) 4-bit even parity checker



(b) 1-bit binary full-adder



(c) 2-bit binary multiplexer



(d) 2-bit binary full-adder

FIGURE 11. The convergence graphs of ISA, SA and GA.

Taking the 2-bit multiplier as an example, we correspond to the optimal value in the evolution process (i.e. the average value of the current optimal fitness value in multiple experiments) with temperature, and its change represents the searching process of the optimal fitness value, as shown in Fig. 12.

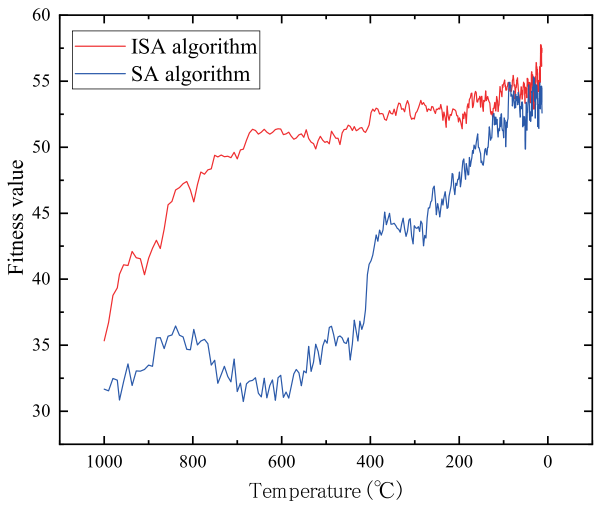


FIGURE 12. Fitness value changing process of each algorithm.

Analysing the results, in the temperature decreasing, we observe that the red line corresponding to the ISA rises more quickly than the blue line related to the standard SA. In addition, during the ascent process, the ISA is more stable, while the SA fluctuates greatly, which indicates that ISA is easier to obtain high-quality solutions than standard SA. It should also be noticed that the average value of the fitness value of the ISA at the stopping temperature is higher than that of the standard SA. These phenomena show that ISA can optimize search results more effectively. In addition, the higher optimization value of the ISA indicates that it is more effective, it is easier to overcome local convergence, and to improve the success rate of evolution.

VII. CONCLUSION

EHW brings together hardware design, fault tolerance, artificial intelligence and autonomous systems to make it provide a very interesting topic and a new field of research. To address the complexity and efficiency of evolution circuits in practical applications, we introduced ANN-based evolution architecture for EHW, and built a complete evolution system in a Cyclone V FPSoC device for simulation and testing. In addition, by introducing two main loops, elitist strategy, chromosome representation and multiple perturbation schemes, a new improved SA is used to achieve rapid evolution of combinational logic circuits. In this design, the high speed of the circuit evolution is maintained because the ANN-based architecture is still implemented in hardware. ISA, as an evolutionary algorithm, runs on the ARM processor and has better flexibility. The experimental results show that our method effectively accelerates the rate of hardware evolution and reduces the average number of evaluations. It also shows that the proposed method can provide a promising solution in real-world applications of EHW.

Finally, with the increase of the scale of circuits, the evolution will stagnate. At this time, simply improving the EA or reconfigurable architecture has little effect on the efficiency of the evolution. Therefore, it may be more effective to adopt new evolutionary models, such as decomposition strategy. We will conduct research in this area in future work. Moreover, the proposed method also seems to be very suitable for EHW fault-tolerant technology, and that will probably be a path of research that we will explore in the near future.

ACKNOWLEDGMENT

The authors would like to thank the editors and anonymous reviewers for all their efforts on our article.

REFERENCES

[1] M. A. Trefzer, A. M. Tyrrell, *Evolvable Hardware: From Practice to Application*. Berlin, Germany: Springer, 2015.

[2] R. Yao, P. Zhu, J. Du, M. Wang, and Z. Zhou, "A general low-cost fast hybrid reconfiguration architecture for FPGA-based self-adaptive system," *IEICE Trans. Inf. Syst.*, vol. E101.D, no. 3, pp. 616–626, 2018.

[3] I. Zelinka, "A survey on evolutionary algorithms dynamics and its complexity—Mutual relations, past, present and future," *Swarm Evol. Comput.*, vol. 25, pp. 2–14, Dec. 2015.

[4] X. Huang, N. Wu, X. Zhang, and Y. Liu, "An evolutionary algorithm based on novel hybrid repair strategy for combinational logic circuits," *IEICE Electron. Express*, vol. 12, no. 22, 2015, Art. no. 20150765.

[5] R. Salvador, "Evolvable hardware in FPGAs: Embedded tutorial," in *Proc. Int. Conf. Design Technol. Integr. Syst. Nanosc. Era (DTIS)*, Apr. 2016, pp. 1–6.

[6] Q. Shang, L. Chen, and P. Peng, "On-chip evolution of combinational logic circuits using an improved genetic-simulated annealing algorithm," *Concurrency Comput., Pract. Exper.*, Aug. 2019, Art. no. e5486.

[7] W. Chine, A. Mellit, V. Lughi, A. Malek, G. Sulligoi, and A. M. Pavan, "A novel fault diagnosis technique for photovoltaic systems based on artificial neural networks," *Renew. Energy*, vol. 90, pp. 501–512, May 2016.

[8] R. F. Molanes, J. J. Rodriguez-Andina, and J. Farina, "Performance characterization and design guidelines for efficient processor–FPGA communication in cyclone v FPSoCs," *IEEE Trans. Ind. Electron.*, vol. 65, no. 5, pp. 4368–4377, May 2018.

[9] R. F. Molanes, M. Garaj, W. Tang, J. J. Rodriguez-Andina, J. Farina, K. F. Tsang, and K. F. Man, "Implementation of particle swarm optimization in FPSoC devices," in *Proc. IEEE 26th Int. Symp. Ind. Electron. (ISIE)*, Jun. 2017, pp. 1274–1279.

[10] D. E. Golberg, *Genetic Algorithms in Search, Optimization, and Machine Learning*. New York, NY, USA: Springer, 1989, p. 411.

[11] S.-J. Chang, H.-S. Hou, and Y.-K. Su, "Automated passive filter synthesis using a novel tree representation and genetic programming," *IEEE Trans. Evol. Comput.*, vol. 10, no. 1, pp. 93–100, Feb. 2006.

[12] R. K. Belew and L. B. Booker, "A survey of evolution strategies," in *Proc. 4th Int. Conf. Genetic Algorithms*, 1991, pp. 2–9.

[13] G. Jian and Y. Mengfei, "Evolutionary fault tolerance method based on virtual reconfigurable circuit with neural network architecture," *IEEE Trans. Evol. Comput.*, vol. 22, no. 6, pp. 949–960, Dec. 2018.

[14] Y. Tao and Y. Zhang, "An extrinsic EHW system for the evolutionary optimization and design of sequential circuit," in *Proc. Artif. Intell. Cloud Comput. Conf. (AICCC)*, 2018, pp. 174–180.

[15] V. Thangavel, Z. Song, and R. F. Demara, "Intrinsic evolution of truncated puiseux series on a mixed-signal field-programmable SoC," *IEEE Access*, vol. 4, pp. 2863–2872, 2016.

[16] R. Dobai and L. Sekanina, "Towards evolvable systems based on the Xilinx Zynq platform," in *Proc. IEEE Int. Conf. Evolvable Syst. (ICES)*, Apr. 2013, pp. 89–95.

[17] R. Dobai and L. Sekanina, "Low-level flexible architecture with hybrid reconfiguration for evolvable hardware," *ACM Trans. Reconfigurable Technol. Syst.*, vol. 8, no. 3, pp. 1–24, May 2015.

[18] O. Garnica, K. Glette, and J. Torresen, "Comparing three online evolvable hardware implementations of a classification system," *Genetic Program. Evolvable Mach.*, vol. 19, nos. 1–2, pp. 211–234, Jun. 2018.

[19] L. Costas, R. Fernandez-Molanes, J. J. Rodriguez-Andina, and J. Farina, "Characterization of FPGA-master ARM communication delays in Zynq devices," in *Proc. IEEE Int. Conf. Ind. Technol. (ICIT)*, Mar. 2017, pp. 942–947.

[20] P. C. Haddow and G. Tufte, "Bridging the genotype-phenotype mapping for digital FPGAs," in *Proc. 3rd NASA/DoD Workshop Evolvable Hardw. (EH)*, Jul. 2001, pp. 109–115.

[21] L. Sekanina, "Evolutionary functional recovery in virtual reconfigurable circuits," *ACM J. Emerg. Technol. Comput. Syst.*, vol. 3, no. 2, p. 8-e, Jul. 2007.

[22] X. Yang, Y. Li, C. Fang, C. Nie, and F. Ni, "Research on evolution mechanism in different-structure module redundancy fault-tolerant system," in *Proc. 7th Int. Symp. Intell. Comput. Appl. (ISICA)*, 2016, pp. 171–180.

[23] P. N. Kumar, S. Anandhi, and J. R. P. Perinbam, "Evolving virtual reconfigurable circuit for a fault tolerant system," in *Proc. IEEE Congr. Evol. Comput.*, Sep. 2007, pp. 1555–1561.

[24] J. Wang and C.-H. Lee, "Virtual reconfigurable architecture for evolving combinational logic circuits," *J. Central South Univ.*, vol. 21, no. 5, pp. 1862–1870, May 2014.

[25] D. Berenson, N. Estevez, and H. Lipson, "Hardware evolution of analog circuits for *in-situ* robotic fault-recovery," in *Proc. NASA/DoD Conf. Evolvable Hardw. (EH)*, 2005, pp. 12–19.

[26] Y. Liang and L. Wang, "Applying genetic algorithm and ant colony optimization algorithm into marine investigation path planning model," *Soft Comput.*, pp. 1–12, Oct. 2019.

[27] S. Kirkpatrick, C. D. Gelatt, and M. P. Vecchi, "Optimization by simulated annealing," *Science*, vol. 220, no. 4598, pp. 671–680, 1983.

[28] B. Chopard and M. Tomassini, *An Introduction to Metaheuristics for Optimization* (Natural Computing Series). Cham, Switzerland: Springer, 2018.

[29] C. M. La Fata and G. Passannanti, "A simulated annealing-based approach for the joint optimization of production/inventory and preventive maintenance policies," *Int. J. Adv. Manuf. Technol.*, vol. 91, nos. 9–12, pp. 3899–3909, Aug. 2017.

[30] Z. Ye, K. Xiao, Y. Ge, and Y. Deng, "Applying simulated annealing and parallel computing to the mobile sequential recommendation," *IEEE Trans. Knowl. Data Eng.*, vol. 31, no. 2, pp. 243–256, Feb. 2019.

[31] H. R. Ansari, M. J. Zarei, S. Sabbaghi, and P. Keshavarz, "A new comprehensive model for relative viscosity of various nanofluids using feed-forward back-propagation MLP neural networks," *Int. Commun. Heat Mass Transf.*, vol. 91, pp. 158–164, Feb. 2018.

[32] P. Colangelo, O. Segal, A. Speicher, and M. Margala, "Artificial neural network and accelerator co-design using evolutionary algorithms," in *Proc. IEEE High Perform. Extreme Comput. Conf. (HPEC)*, Sep. 2019, pp. 1–8.

[33] K. Amine, "Multiobjective simulated annealing: Principles and algorithm variants," *Adv. Oper. Res.*, vol. 2019, pp. 1–13, May 2019.

[34] X. Han, Y. Dong, L. Yue, and Q. Xu, "State transition simulated annealing algorithm for discrete-continuous optimization problems," *IEEE Access*, vol. 7, pp. 44391–44403, 2019.

[35] E. Yadegari, A. Alem-Tabriz, and M. Zandieh, "A memetic algorithm with a novel neighborhood search and modified solution representation for closed-loop supply chain network design," *Comput. Ind. Eng.*, vol. 128, pp. 418–436, Feb. 2019.

[36] A. I. Hammouri, M. S. Braik, M. A. Al-Betar, and M. A. Awadallah, "ISA: A hybridization between iterated local search and simulated annealing for multiple-runway aircraft landing problem," *Neural Comput. Appl.*, pp. 1–21, Dec. 2019.

[37] (2015). *Altera Corporation: Cyclone V Hard Processor System Technical Reference Manual*. [Online]. Available: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-v/cv_5v4.pdf

QIANYI SHANG (Student Member, IEEE) received the B.E. degree in electric power systems and automation from the Ordnance Engineering College of PLA, Hebei, China, in 2013, and the M.S. degree in computer technology from the Equipment Simulation Training Center, Army Engineering University of PLA, Jiangsu, China, in 2019. He has participated in Mathematical Modeling Contest actively and carried off many prizes. His current research interests include embedded systems, software engineering, genetic algorithms, evolutionary computation, and so on.

LIJUN CHEN (Member, IEEE) received the M.S. and Ph.D. degrees from the Department of Computer Engineering, Ordnance Engineering College of PLA, Hebei, China. Since 2017, he has been with the Equipment Simulation Training Center, Army Engineering University of PLA, Jiangsu, China, where he is currently a Professor. His current research interests include embedded systems, software engineering, neural networks, and fuzzy logic.

JING CUI (Member, IEEE) received the Ph.D. degree in information technology from Xi'an Jiaotong University, in 2017. She is currently an Associate Professor with the Equipment Simulation Training Center, Army Engineering University of PLA, Jiangsu, China. Her main research interests include embedded systems, software engineering, and artificial intelligence.

YAO LU (Member, IEEE) received the B.E. degree in electronic science and technology from Heilongjiang University, Heilongjiang, China, in 2010, and the M.S. degree in computer technology from the Equipment Simulation Training Center, Army Engineering University of PLA, Jiangsu, China, in 2019. His current research interests include artificial intelligence and network security.