Journal of Electronic Testing (2022) 38:547–565
https://doi.org/10.1007/s10836-022-06027-6

Self Healing Controllers to Mitigate SEU in the Control Path of FPGA Based System: A Complete Intrinsic Evolutionary Approach

S Deepanjali[1] · Noor Mahammad Sk[1]

Received: 4 July 2022 / Accepted: 18 September 2022 / Published online: 28 September 2022

Abstract
Single event upsets (SEU) are the transient errors that occur during the operation of the circuit. High radiation in the space environment and its invasion of the nanoelectronics can result in a bit-flip in the combinational circuits and may cause a stuck-at fault in the sequential circuit. Faults are unacceptable for any application, especially SEU in the control path, which is crucial and imperative since it can lead to functional or even mission failure. As a result, this paper proposes a bio-inspired technique based on a modified heuristic-guided genetic algorithm to mitigate error at the Finite State Machine (FSM), which is the controller's behavior model. The proposed architecture performs an optimized functional level evolution of the FSM intrinsically on ProAsic3e FPGA boards without the help of System-on-Chip (SoC). Due to this, the delay caused by extrinsic and hybrid evolution has been reduced. The proposed heuristic-guided genetic algorithm recovers the fault in the control circuit with less convergence time when compared to the standard genetic algorithm. The resource utilization of the proposed evolvable hardware system has reduced costs compared to traditional functional evolution.

Keywords SEU · FPGA · Fault Tolerant Evolvable Hardware · Genetic Algorithm · Bio-inspired electronics · Adaptive Hardware · Transient Errors

1 Introduction

The control path of the Hardware is a vital part of any application as it decides how and which data path elements have to operate. The need for deploying the controller on FPGA is high in most critical and adaptive Hardware, such as Unmanned Aerial Vehicle (UAV) [4], reusable launch vehicles, and ground support equipment, due to its advantage of dynamic and run-time configuration to mitigate on-chip and off-chip errors, as well as to reduce mission cost by reconfiguring the same FPGA chips for multiple applications [3].

There is a great demand for making the control path of FPGA boards Anti-SEU [5]. However, the downside of this design approach is the FPGA-based systems' subjection to the harsh environment. In such conditions, the board can face transient faults termed single-event errors.

The control circuit of the FPGA-based system is modeled using FSM. Given any state machine for a controller, the hardware design comprises a combinational circuit to compute the output (control signals) and next state based on the Boolean expression reduced by K-Map from the state transition table and a memory component (Register or Flip-Flop) to provide the next state as the current state at the end of a clock cycle, as shown in Fig. 1.

The transient errors that happen in FSM can cause a single or multiple-bit flip in the following two cases:

- **Case 1: SEU on Input bits** SEU can impact the combinational circuit's input bits while transiting from the current state to the next, leading to a wrong transition.
- **Case 2: SEU in Memory Component** The error can happen in the register or a flip-flop, which stores the next state encoding. As a result, bit-flip occurs in the state encoding of the next state, which is sent as input in the corresponding clock cycle. This bit encoding error in the

Responsible Editor: V. D. Agrawal

✉ Noor Mahammad Sk
noor@iiitdm.ac.in

S Deepanjali
cs21d0001@iiitdm.ac.in

[1] Department of CSE, Indian Institute of Information Technology, Design and Manufacturing, Kancheepuram, Chennai, India

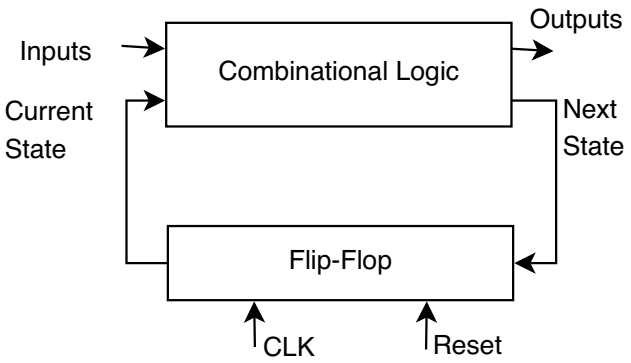

Fig. 1 Hardware representation of finite state machine

memory component can lead to a wrong transition and thereby induce function failure.

Any error recovery procedure has to tackle these error models to ensure an absolute and robust control circuit. For instance, consider the control circuit of a robot performing object search and retrieval task [16] comprising three main tasks such as environment inspection (I), object seizing (S), and reaching Target position (T), as shown in Fig. 2. The main operations of the robot are modeled as a finite set of states {I, S, T} respectively. The transition between the states happens on accepting two-bit binary input = {00,01,10,11}. The input of each state is encoded using segmented binary encoding. The first bit in the input represents whether an object is recognized in the camera, and the second bit represents the odometer value. The X in the transition arc represents the input, which can be either 0 or 1.

In Fig. 2 consider a transition function T (reaching target position) from object seizing defined as $S \times 0X \rightarrow T$. If a single event upset happens while reading the first bit of the input (0X). The transition function becomes $S \times 1X \rightarrow I$, which makes the control circuit send the wrong control

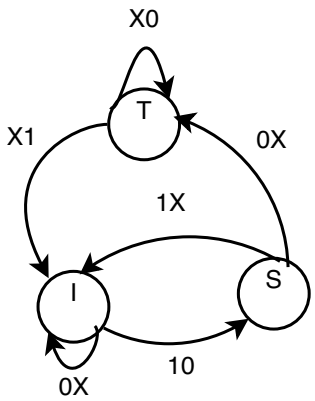

Fig. 2 FSM of Object Search and Retrieval robot

signal for the data path elements and perform a different function than intended. Consider the encoding of state I-environment inspection represented in binary as *00* and the occurrence of a single event upset in the register can cause the flip in the first bit, resulting in *10* denoting the operation of navigating to the target position. In this case of fault occurrence, the control circuit communicates the wrong control signal to undesired data path elements. Hence, it is essential to develop a mitigation technique/mechanism to ensure the correct functioning of the FSM.

In general, the fault tolerance in digital electronics is performed via redundancy based methods, such as: Hardware [7, 19, 21], Time [1] and Information [6, 12]. For deep space exploration and satellite rovers, the use of FPGAs has increased. For such a huge number of FPGA-based designs, the redundancy can increase the size of the overall Hardware in terms of size and power. To overcome the overall increase in size, the types of faults on the Hardware were analyzed priorly, and partial redundancy on selected components was applied [20, 22]. However, autonomous or self-adaptive Hardware does not require redundancy at any level. For instance, evolvable Hardware, a bio-inspired fault-tolerant electronics, utilizes the same faulty component and an evolutionary algorithm module to mitigate the error.

The primary reconfiguration technique in evolvable Hardware is focused on bitstream level evolution. The bitstream is the collection of the configuration data obtained by the FPGA-specific software after the place and route operation of the digital application. It is stored in the configuration memory of the FPGA and describes the routing information and contents of the LUT, CLB, and clock signals. Access to this configuration data is made possible by using FPGA-specific Dynamic Partial Reconfiguration (DPR) tools.

Access to the configuration data is restricted in multiple military-grade FPGA and FPGAs with high security. Furthermore, the bit streams are also encrypted. To confront this challenge, an application-level mimicry of the configuration memory has been developed with the help of Virtual Reconfigurable Circuit (VRC) [23]. The VRC comprises the configuration register, which contains the select line values of $m \times n$ array multiplexer. These multiplexers are routed and function based on the select line values in the configuration register. In contrast to the bitstream evolution, the circuit level evolution is performed by applying the configuration register content as a chromosome. As a result, different routing and functionalities of circuits are verified.

The greatest challenge in VRC is high resource overhead since it is a multiplexer-based structure. For a simple full adder circuit, approximately 89% of the resources are utilized for the multiplexer. Hence, there is a vital requirement for optimizing the VRC array structure. This paper presents an optimized VRC array structure with a heuristics-guided

genetic algorithm for accelerated fault tolerance and increased scalability. The additional delay in processor-based evolution is reduced by deploying the evolutionary algorithm on the same FPGA. The proposed heuristics-guided genetic algorithm's design mainly utilizes the control circuit's deterministic nature. The faults are simulated at the VRC's functional and routing capabilities, and the effectiveness of the fault tolerance is reported.

The rest of the paper is organized as follows. Section 2 explains the preliminaries of Evolvable Hardware (EHW) and related work. Section 3 presents the proposed intrinsic evolvable hardware system for SEU error correction and detection and explains it with different applications. Section 4 details the implementation methods adopted to verify the proposed design, Sect. 5 details the results and discussion, followed by Sect. 6.

1.1 Contribution of this Paper

- The bitstream evolution and standard function level evolution are replaced with optimized function level evolution. As a result, resource utilization is reduced compared to standard function level evolution.
- The deployed control circuit as an optimized functional evolution array is represented as a chromosome and evolved using a heuristic-guided genetic algorithm to reduce the chromosome length and accelerate the convergence rate.
- The proposed function level evolution array and heuristic-guided genetic algorithm are deployed as digital circuits on the same FPGA, and complete hardware level evolution is performed to eliminate the dependability and delay caused by extrinsic and hybrid evolution.
- The proposed EHW system is tested for efficacy for different types of faults and its improvement in resource utilization and convergence rate is presented.
- The proposed solution uses a military-grade FPGA, which does not possess specialized support such as JTAG bits software and processor for the implementation of EHW as utilized in related work.

2 Background

The application of characteristics found in bio-organisms to electronics for self-organizing and self-adaptiveness is called bio-inspired fault tolerance in electronics. In general, redundancy-based fault mitigation methods can increase the cost of the entire mission in terms of area, power, and delay. Hence an intelligence-based methodology is required at the circuit level to adapt itself to the changing environment. The bio-inspired fault tolerance is categorized into three dimensions based on the POE model where P stands for ***P****hylogeny* - The evolution capabilities of species inspire the next generation of electronics. Evolvable Hardware is an example of this dimension. The ***O****ntogeny* in the POE model is motivated by the multi-cellular division of the zygote. Embryonics [18] is an example of this category. A fault in a component is mitigated by replacing the faulty cell with a neighboring spare cell. The ***E****pigenesis* adapts the learning behaviour of the species to electronics called Immunotronics [27]. This vertical of the POE model is adapted from the human immune system's inbuilt characteristics of identifying Self/Non-self. Among these dimensions of bio-inspired electronics, our proposed work focuses on phylogeny to design a self-healing control path.

EHW is the application of biological concepts to electronic Hardware with the help of evolutionary algorithms. The field of evolvable Hardware was pioneered by Thompson et al. [32] in 1996. The field of research of EHW has been distinguished under two motivations: where EHW is used for optimized hardware design and the design of adaptive Hardware. The autonomous design of the Hardware focuses on designing the electronic circuit from scratch based on parameters like area, power, and delay. Various combinational circuits are self-designed by EHW under these categories, as shown in Table 1. The latter research area focuses on making hardware design adaptive to the changing environment, especially in applications where the hardware design is prone to harsh environments like space and nuclear reactors. EHW is best suited for adaptive Hardware as it finds the fault-tolerant solution quickly in search space autonomously

Table 1 Recent works of Evolvable Hardware for autonomous design of digital circuit

Reference	Reconfigurable Fabric	Bio-Inspired Algorithm	Approaches	Application
[38]	Zynq-7000 SoC	Cartesian Genetic Programming	Hybrid	Design of 2 bit multiplier and 8-bit parity
[36]	Virtex 6 (XC6VLX240T) Ml605	Genetic and Memetic Algorithm	Hybrid	Design of 2 and 4-bit adder and multiplier and 6-bit parity generator
[25]	Spartan6 XC6SLX45-CSG484-3.	Embryonic and Genetic Algorithm	Extrinsic	Design and self rectification of BCD Decoder
[30]	SoC-based FPGA	Cartesian G enetic Programming	Extrinsic	Design of 2-bit Multiplier
[40]	Intel Cyclone V-SoC	Genetic Algorithm	Extrinsic	Design of 4-to-1 even parity generator and 2 bit adder and multiplier

when the Hardware is in operation. This EHW consists of two components: reconfigurable Hardware and an evolutionary algorithm.

2.1 Reconfigurable Hardware

The reconfigurable Hardware is the platform on which the evolutionary algorithm is applied. Any application deployed on this reconfigurable platform is initially written using a hardware description language, followed by routing the configuration bits (bitstream) on the Hardware. The bitstream describes the system behavior of the application, and it is placed in the configuration memory of the reconfigurable Hardware. The advancement in the usage of hardware platforms commenced with programmable array logic (PAL) [29, 31] to complex programmable architectures such as Field Programmable Gate Array (FPGA) [33, 35], Field Programmable Analog Array (FPAA) [11, 41], Field Programmable Transistor Array (FPTA) [10, 13]. Although in recent times, FPGA-based EHW has become mainstream.

2.2 Evolutionary Algorithm

An evolutionary Algorithm (EA) is an iterative algorithm initiated with a chromosome representation. The hardware characteristics are represented in the form of genes. The characteristics used in the chromosome representation can differ for each application. For instance, a robotic arm chromosome is structured with the number of joints, position, and number of fingers. Hence, it has to be carefully decided by the designer. The commencement of the evolutionary algorithm is an initial random population where random bits are generated in the length of the chromosome. These random bits are termed the population of the evolutionary algorithm. Each chromosome from the population is applied to the fitness calculation, and it is associated with the fitness score. The designer specifies the objective function and threshold value based on the application. For instance, the fitness score for evolving antennas is to obtain a signal power above 2 decibels. The selection operation controls the passing of chromosomes to subsequent generations. Based on the fitness score at each generation, the chromosome is selected using roulette wheel selection, rank selection, or tournament selection. The allele is flipped (mutation) among the selected individual genomes to generate new offspring. The crossover operator is responsible for pairwise recombining two selected individuals (Parent 1, Parent 2). These two operators are responsible for generating new offspring with variation and promoting the reproduction of the selected or fittest chromosome. The algorithm converges to the desired solution when the algorithm is iterated for a specified number of generations or when the desired result is attained.

Figure 3 depicts the evolutionary process performed by the Standard Genetic Algorithm (SGA) [9]. Apart from SGA, the other evolutionary algorithms include evolutionary strategies [2], Cartesian Genetic Programming [17]. The mentioned variants of evolutionary algorithms differ from each other based on genetic operation. For instance, SGA, Cartesian Genetic Programming, and Genetic Programming are distinguished based on the chromosome representation, where SGA follows segmented binary encoding, Cartesian Genetic Programming represents the genotype as a directed acyclic graph, and Genetic Programming considers the hardware description language as a parse tree for the genotype representation. The evolutionary strategy is represented as the $(1 + \lambda)$ method, where 1 represents that reproduction is done at an individual level, and λ represents the number of offspring generated.

2.3 Related Work

The research in the field of evolvable Hardware is explored in different categories such as the granularity of evolution, type of evolution based on location, reconfigurable Hardware used, reconfiguration methodology, and evolutionary algorithm used. The hardware platform used for evolution has developed from simple programmable logic comprised of 1000 gates to a commercial FPGA with millions of gates to accommodate complex designs. As mentioned earlier, the FPGA has been the most commonly used platform for evolution in recent times. FPGAs like the XC6200 series, Virtex series, and Spartan are widely accepted FPGAs for EHW. Also, the choice of SRAM-based FPGA is higher when compared to other FPGA technologies like Flash-based or Antifuse-based FPGA since the reconfiguration speed is high compared to others [37]. In the initial research, the bitstream format of FPGA was available to the public to promote the research. For instance, XC6200 by Xilinx was the first commercial FPGA for which the bitstream format and mapping of control logic blocks were documented and made accessible. However, FPGA manufacturers have moved towards employing more robust encryption for bitstream to prevent reverse engineering and side-channel attacks. Since 2015, FPGA manufacturers have used Data Encryption Standard (DES), and Microsemi FPGA has introduced Advanced Encryption Standard (AES) for encrypting the bitstream of the FPGAs used in military and space applications. Higher encryption standards have challenged the bitstream evolution greatly as they require more time to decrypt the bitstream. However, some work has shown evolution at the configuration bit level by using application interfaces provided by the vendor, such as JTAG bits [14] and ICAP (Internal Configuration Access Port) controller. This ICAP controller facilitates Dynamic Partial Reconfiguration (DPR), where reconfiguration is

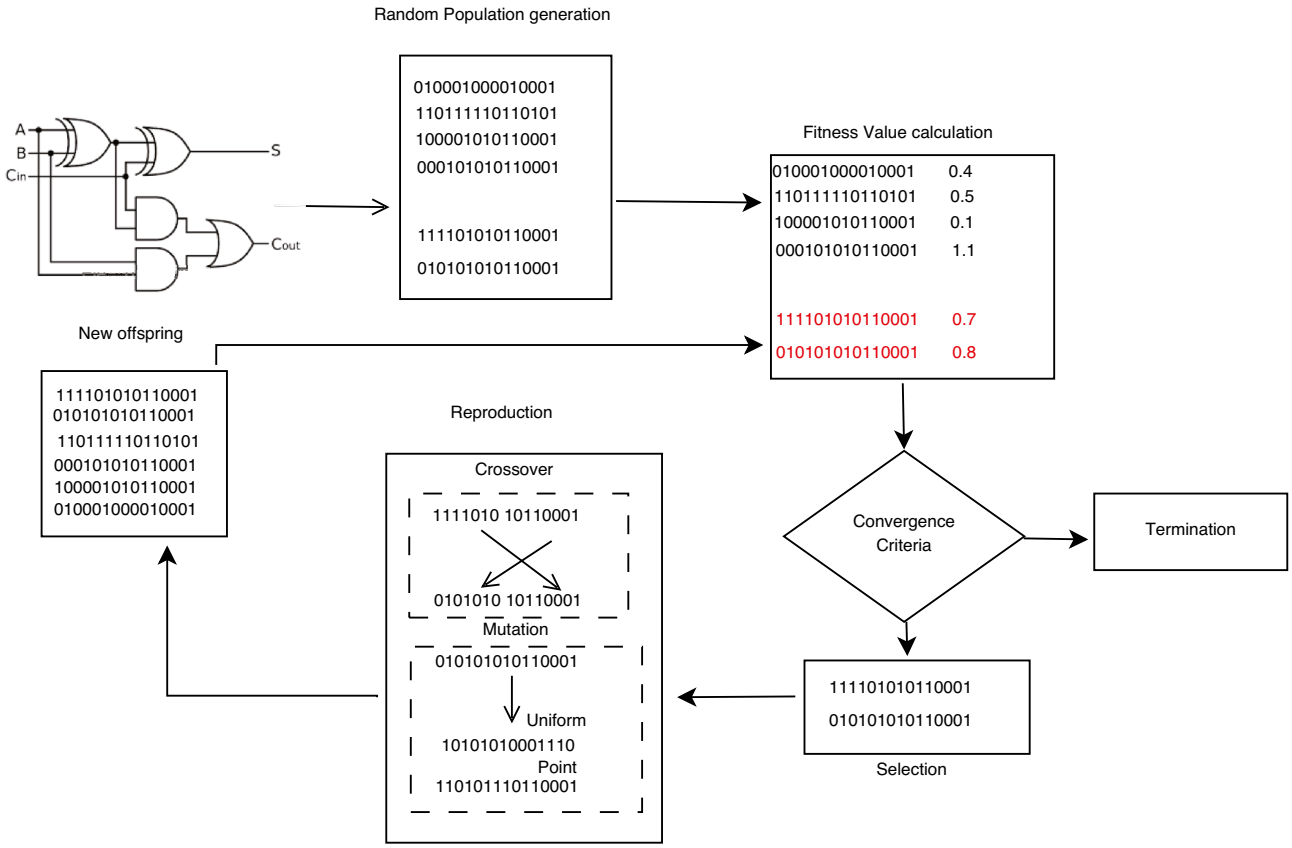

Fig. 3 Block diagram of Evolutionary algorithm

addressed for a portion of the circuit in run-time. The VRC is an alternative option for the reconfiguration mechanism in FPGA where DPR is unavailable. It was introduced by Sekanina in his work [23]. A VRC is a reconfigurable circuit built for an application on top of the same FPGA, and it performs function-level evolution. It is a two-dimensional array of PE (programming elements), where each PE comprises the implementation of all possible functions which can be selected using a multiplexer. The array of PE is instantiated with the input signal, which is accepted in the first column of the array. The subsequent column of PE obtains its input from the previous column and propagates the final circuit's output to the last column of PE. The select line used in selecting the inputs and functions of each PE is stored in the configuration register. The information present in this register acts as the chromosome in VRC, whereas in DPR, the bit stream generated by the design flow of the FPGA acts as the chromosome. For multiple FPGAs used in space, military, and other mission-critical applications, the bitstream is encrypted, or the availability of API and ICAP is non-existent. Hence, the VRC-based reconfiguration methodology is quietly accepted by EHW researchers. The challenge in the VRC-based methodology is that each PE is multiplexer based and implements all possible functions, increasing power consumption and resource utilization. However, as an advantage, VRC is fast since the delay only involves switching between the select lines to change the configuration. Among various characteristics of EHW, the location of evolutionary algorithm implementation plays a significant role in the EHW distinction. Because the design considered for evolution was simple, and there was no processor on the FPGA, pioneer research in EHW hosted the evolutionary algorithm in the external PC referred to as *Extrinsic evolution*. Later, many SoC-based FPGAs were manufactured, allowing the evolutionary algorithm to be implemented on the processor in the FPGA. The AXI bus, or the other system bus architecture, assists in loading the evolved circuit in the FPGA. This type of evolution is called *Hybrid evolution* where the genotype is evaluated on the Hardware, and the evolutionary algorithm is hosted as software present in the SoC of the FPGA. Complete *Intrinsic* evolution involves genome modeling, evaluation, and implementation of genetic operations on the same FPGA. Intrinsic evolution is considered the fastest type of evolution as there is no delay caused in transferring the genome between the FPGA and the processor. A comparison of hybrid and intrinsic evolvable Hardware is studied in [8]. This work has proved that the complete

hardware evolution works 7.74 times faster than the hybrid approach, with 0.198 lower power consumption and fewer resources. In this comparison work, a typical application of evolving category detection of sonar signals is considered where the genotype of length 480 bits is evolved. Regardless of these advantages, research in the field of complete hardware evolution [34] is scarce.

The work by NASA AMEs research centre [15] proposed a dual-board hybrid EHW on the Virtex 4 FPGA. The redundancy is applied at the board level, where board B performs the designated operation of board A while it undergoes evolution. This work converged to a fault-free chromosome after 1000 generations, accounting for 45 hours. The genetic algorithm was hosted on the JBIT software provided as an IP core by Xilinx FPGA vendors. This work posed the challenge of migrating the proposed methodology to complex logic and suggested accelerating the evolution cycle by implementing EA directly on Hardware. Similarly, in 2017, a self-repairing control circuit was implemented for a brushless DC motor using a hybrid EHW on a Virtex 6 ML605 FPGA [42]. The design employed the genetic algorithm on the MicroBlaze soft-core processor and used the ICAP controller for reconfiguring the bitstream. After 10,000 generations, the algorithm reached a fault-free controller. Also, in this above work, they have expressed that the repair time for the controller is high in terms of generation during which external service can be interrupted. The influence of genetic operators on the success rate and fault recovery time in terms of clock cycles is studied in [26]. A hybrid evolution of the BCD decoder is performed on the Spartan 6 FPGA, where the genetic algorithm is hosted on the MicroBlaze soft core processor. This work summarizes a comparison of selection, crossover, and mutation. Compared to the tournament-based selection, this work demonstrated that roulette wheel selection has a 0% success rate and a longer fault recovery time. A hybrid evolvable hardware [39] is implemented on a MicroBlaze soft-core processor of spartan 6 FPGA to mitigate errors in the 8-bit parity checker and 3-bit multiplier. A configuration library is maintained to compare obtained and anticipated results. The average fault recovery time is accounted for 1.83 seconds with 94% accuracy. Similarly, 1-bit mutation and 1-bit crossover have a higher success rate with less recovery time when compared to uniform mutation and crossover. The common challenges identified in cutting-edge work are

- **Low Scalability:** The state-of-the-art work faces difficulty in considering applications of higher size since bitstream level evolution is considered. The circuit size considered for evolution is low because the circuit size and bitstream length are directly proportional. For instance, the methodology in [39] encodes the chromosome of length 288 bits for an 8-bit parity checker and 441 bits for a 3-bit multiplier. Even though the circuit size is smaller, the total chromosome length is vast. The bit level evolution is further not possible in highly secured military grade FPGA, as the access to the bitstream is constrained.
- **Dependability:** The evolution process is implemented on the SoC present in the FPGA in [15, 26, 39, 42]. Implementing the evolutionary algorithm in the processor and evaluating the evolved chromosome in Hardware can create an additional delay. Hence, in [15], it is suggested to perform the evolution directly on the FPGA.
- **Convergence Rate:** The convergence rate of the EA plays a massive role in the fault recovery rate. If a genetic algorithm without any optimization is applied, the evolution can take several hours. For instance, the technique used in [15] required 45 hours for evolution. Hence, better understanding and selection of suitable parameters have to be decided for respective genetic operators, especially for adaptive Hardware where fault mitigation is required with high efficiency and speed.
- **Redundancy Rate:** The authors of [15] and [42] have suggested redundancy at the FPGA board level to eliminate the halt in the mission. Dual FPGA boards are utilized in [15].

3 Proposed Intrinsic Evolvable Hardware System

The hybrid EHW system discussed in Sect. 2.3 comprises three main components, such as a *Microprocessor* to host the evolutionary algorithm in its native high-level language; an *Evaluation Module* which is a copy of the target circuit to evaluate the fitness function of the generated chromosome by the EA; and an *AXI bus* to mediate the chromosome in between the evaluation module and the processor. Furthermore, the ICAP controller is used as an API for accessing the configuration bits if the evolution happens at a bitstream level.

The architectural difference with hybrid EHW is depicted in Fig. 4. The processor in the related works is substituted by the GA (Genetic Algorithm)_Module deployed as a digital circuit on the same chip. This module is responsible for all genetic operations in Fig. 3. In addition to the GA module, the proposed system design includes an error detection circuit to identify the SEU in the target circuit (control circuit). If the detection circuit recognizes the fault, the GA status signal is set to 1, which instantiates the GA_Module. The status signal 0 indicates that SEU is non-existent in the control circuit. Hence, the control signal and next state are forwarded to other data path elements and the control circuit's memory element (flip-flop).

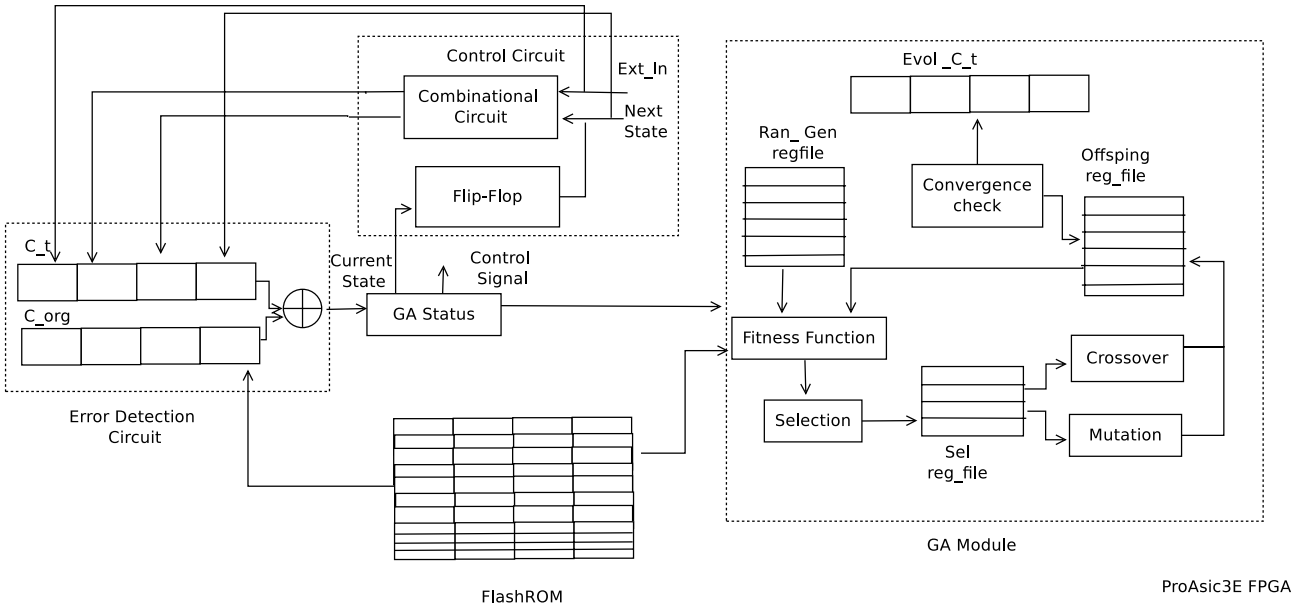

Fig. 4 Block diagram of proposed intrinsic evolvable hardware system

3.1 Optimized Function Level Representation of Control Circuit

In related work, the configuration bitstream level evolution is followed, which encompasses gate level granularity, thus making the evolution process time-consuming. As a result, Function level Evolution (FE) [24] is used in our proposed EHW system. The FE requires a two-dimensional array of programming elements (PE) designed on the same FPGA fabric. Each PE has a MUX-based implementation to select one function among the set of predefined functions. The selected function is applied to the inputs of PE, and the output is propagated to the PE of the next column. Each PE's input, output, and function information are stored as a configuration bit in the configuration register. In the previous works of FE, the number of functions realized in each PE is generic; therefore, the hardware overhead and configuration bit to select the function are high. The number of functions operated in each PE is constrained to produce an optimized function level evolution. This restriction is only possible due to the deterministic nature of the control circuit. Using K-Map reduction, any control circuit represented as a state transition table can be realized as a boolean expression. This expression is expressed as a combinational circuit in Fig. 1. Hence, the established functionality in the expression of the combinational circuit is only realized in each PE.

3.1.1 Example

Consider the control circuit of the brushless DC motor formulated as the state transition Table 2. The DC motor contains three positioning signals, T0, T1, and T2, and, on completion of each rotation, based on the values of the positioning signals, the driving signals Q0, Q1, Q2, Q3, Q4, and Q5 are activated. The input and output signals combination is described in the Truth Table 2.

From the truth Table 2, the Boolean expression enclosing the relation between the input variable and output variable is obtained by K-Map simplification and formulated in the Eqs. (1-6)

Table 2 State transition table: Brushless DC motor control circuit

Position Signal			Driving signal					
T0	T1	T2	Q0	Q1	Q2	Q3	Q4	Q5
1	0	0	0	1	0	0	0	1
1	1	0	0	1	0	1	0	0
0	1	0	0	0	1	1	0	0
0	1	1	0	0	1	0	1	0
0	0	1	1	0	0	0	1	0
1	0	1	1	0	0	0	0	1

$$Q0 = \overline{T1}.T2 \quad (1)$$

$$Q1 = T0.\overline{T2} \quad (2)$$

$$Q2 = \overline{T0}.T1 \quad (3)$$

$$Q3 = T1.\overline{T2} \quad (4)$$

$$Q4 = \overline{T0}.T2 \quad (5)$$

$$Q5 = T0.\overline{T1} \quad (6)$$

The Eqs. (1-6) shows that the driving signal is obtained by only AND and NOT functions. Hence, the proposed optimized functional evolution array consists of an array structure, as shown in Fig. 5. The driving signal is derived from the last column of the array. The first column accepts the external input signal T0, T1, T2, and other columns receive the input from their preceding column. Each PE comprises three selection signals, as shown in Fig. 6, where Sel 1 and Sel 2 of length 3 bits select the input signals, and Sel 3 of length 2 bits enables the function performed by each PE. The combination of these three select signals constitutes the configuration bit in the configuration register. In a standard function level evolution, the number of gates and length of Sel 3 signals would drastically increase the hardware resources. Also, multiple multiplexers would have been used to select the function. Hence, our proposed optimized functional level evolution will decrease the complexity of the proposed EHW system.

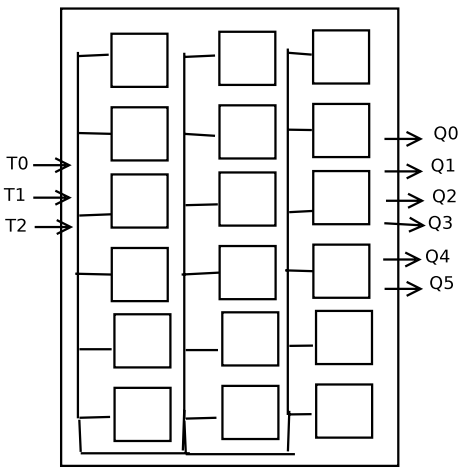

Fig. 5 Representation of functional PE array for brushless DC motor

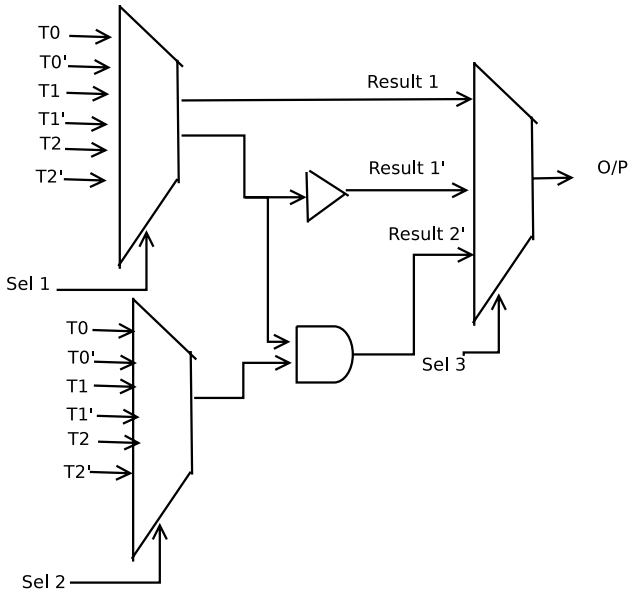

Fig. 6 Representation of single PE for brushless DC motor

3.2 Error Detection Circuit

The error detection circuit perpetually operates in synchronous with the control circuit. The control circuit, accepting external input, produces the control signal and the next state. These signals and the input signal and current state are checked for SEU. The memory-based training sample assists in the identification of the SEU. The possible input and output signals obtained from the state transition table are stored as content addressable memory in the FlashROM of the FPGA fabric. As a result, when the control circuit receives the input-positioning signal at time *t*, the corresponding driving signal is checked against the training example in the FlashROM. Any deviation from the desired output indicates that an error has occurred, which triggers the GA status signal to 1, and the GA module is instantiated. Instead, when SEU is undetected, the control signal is propagated to the data path elements. Figure 4 performs the error detection process where C_t represents the combination of input, current state, next state, and output control signal at time *t* generated from the control circuit, and C_{org} is the training example combination of all the above for the corresponding input. The bitwise XOR operation identifies the deviation between the obtained and expected results. The GA status signal either instantiates the GA module for fault recovery or forwards the control signal when the fault is undetected.

3.3 Heuristics-guided GA Module

The GA module is the crux of the EHW system. In a hybrid EHW system, the module is hosted in the microprocessor of the FPGA. But in our proposed EHW system, the operation of the genetic algorithm is deployed as a digital circuit using a hardware description language. The representation of the digital circuit as a genotype is the initial phase accomplished ahead of all the genetic operations. In our proposed system, we have analyzed the deterministic nature of the control circuit and applied the heuristic in chromosome encoding to reduce the genotype length. The chromosome structure contains three main elements: *External Input* supplied to FE array, *Programming elements (PE_i)* information such as Sel 1, Sel 2, Sel 3 of programming element activated at each column, where i represents the column of the FE array and the final output (control signal) generated as shown in Fig. 7. The total chromosome length is given by Eq. 7, where I represents the number of bits to represent the input signal, N represents the number of columns in the FE array, O represents the output control signal, and n1, n2, n3 represents the number of bits to represent the Sel 1, Sel 2, Sel 3.

$$Chromosome\ Length = I + N * (n1 + n2 + n3) + O \quad (7)$$

3.3.1 Population Initialization

The population P_i, of each i^{th} generation is collection of chromosome C_x, where x ϵ to [1, pop-size]. The primary population (P_0) applied as input to the GA algorithm is generated by random sampling. The choice of sampling method is crucial to prevent the algorithm from premature convergence. In our related works [15, 26, 39, 42] the initial population generated was unbounded since heuristic about the target circuit was not considered. The proposed EHW system is focused on utilizing the deterministic nature of the control circuit, so we have designed a restricted random sampling in our EHW system. The control circuit has a set of permissible external inputs in contrast to the other combinational circuits. For instance, Table 2 shows that external input is 3 bits and, among 8 (2^3) combinations, only six combinations of positioning signals have valid driving signals. Hence, the chromosome deriving from 000,111 can be restricted for population generation. This restriction prevents the time being utilized for invalid chromosome evaluation.

3.3.2 Fitness Calculation

The fitness function and value of the GA algorithm are distinct for each target circuit. In our previous works [15, 26, 39, 42] two methodologies where followed to establish the fitness function.

- **Comparison based** approach was designed. The expected circuit response was stored in the memory and compared with the obtained result. For a complex circuit, memory latency and occupancy can be concerning. This methodology is highly reliable, assuming no error has occurred in the stored bits.
- **Objective Function based:** The fitness value and function were predicted by interpolation of the training sample. The fitness value of the current sample for the same objective function was recorded, and its deviation from the threshold fitness value was analyzed. This methodology is suitable when an EHW system is used in the application, such as the autonomous design of the circuit. In fault recovery-based EHW systems, this approach poses a challenge when the current sample either under fits or overfits the function curve. For instance, the fitness function for a brushless DC motor was devised using the Lagrange interpolation method as $-0.5x^3+4.796x^2-8.111x+60.333$ with a regression error of 1.089 due to which the fitness calculation for fault free chromosome representation was miscalculated with a lesser fitness value. Hence, in our proposed method, a comparison-based approach was used. The expected result for each input combination with programming element details is stored in the FlashROM. The generated chromosome C_i at each generation is compared with the expected chromosome C_{org}. The number of bits deviated is calculated from the compared result as shown in the algorithm 1.

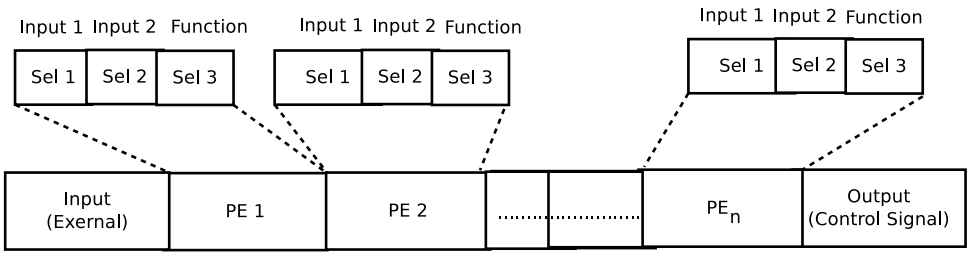

Fig. 7 Genotype representation for proposed GA algorithm

Algorithm 1 Fitness Value Calculation

Require: C_i
Ensure: Fitness Value $_i$
1. C_i is generated at generation g, where i ϵ [1,popsize]
2. The input field in C_i access the matching C_{org}
3. Comp = $C_i \oplus C_{org}$
4. For each bit: $0 \rightarrow$ n, where n = len(comp)
   if bit==1
   count=count+1
   Fitness value$_i$ = $\frac{100-(count*10)}{100}$

3.3.3 Genetic Operators

The genetic operators comprise functions such as selection, crossover, and mutation. The selection operator chooses the fittest chromosome at each generation for offspring reproduction using crossover and mutation. A rank-based selection method is chosen to select the best fitting chromosome based on its fitness value. Each chromosome C_i has associated fitness value$_i$ which is compared with each other and ranked. At least four chromosomes with the highest fitness value are selected and applied for crossover and mutation. The crossover operator helps in generating all variants of input, whereas the mutation operator supports the convergence of the fittest chromosome. Uniform mutation and point-wise crossover are chosen as genetic operators to yield the offspring for the next generation.

The produced offspring is subjected to subsequent generations until the convergence of the EA algorithm. The elitism is used in our algorithm for a chromosome whose input segment is identical to the target circuit's current input (position signal in the case of a brushless DC motor). This strategy in our proposed system has enhanced our convergence speed by applying selective pressure on the target circuit's current input. The proposed EA algorithm is converged when the elite chromosome has reached a fitness value of 100%. The following section demonstrates the working of our proposed intrinsic EHW system and heuristics-based EA algorithm by considering four control circuits.

Table 3 Encoding for Genotype representation: Brushless DC Motor

Input	Encoding	Function	Encoding
T0	001	Input	00
T0'	010	NOT	01
T1	011	AND	10
T1'	100		
T2	101		
T2'	110		
No input	000		

3.3.4 Example: A Genetic Evolution of Brushless DC Motor

The brushless DC motor realized in Fig. 5 shows that the array structure contains 18 PE with six rows and three columns. Each PE contains three selection lines for selecting Input 1, Input 2, and its function. The binary segmented encoding is followed as shown in Table 3. For instance, the chromosome 110-001-000-01-110-000-01-001-110-10-010001 represents that when the input (position signal)-110 is supplied to the FE array, PE in column 1 contains the configuration bit 001-000-01, PE in column 2 contains the configuration bit 110-000-01, and PE in column 3 contains the configuration bit 001-110-10, producing the output (driving signal) as 010001. Hence, the proposed genotype representation depicts the FE array absolutely for a given input signal. The total chromosome length for the brushless DC motor is 3+3*(3+3+2)+6 = 33 bit. The proposed heuristic-based chromosome encoding has reduced the genotype length compared to [42] by 36.66 % for the same application.

The initial zeroth generation is randomly populated following the restriction as mentioned in Sect. 3.3.1. Each generation contains *n* chromosomes fixed by the designer. For instance, if 100 is configured as the population size, the chromosome is generated randomly following the binary encoding in the range [0-2^{33}], where 33 is the chromosome length for the brushless DC motor. Consider two chromosomes generated,C_i and C_j among the population at some ith generation as shown in Fig. 8

110-000-000-10-110-010-10-101-111-01-101001

001-101-011-11-011-001-00-111-010-01-110001

Fig. 8 Two example chromosomes C_i and C_j generated at any ith generation

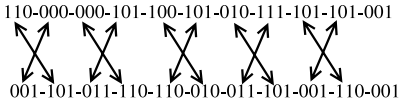

Fig. 9 Example of 3*3 crossover on chromosomes C_i and C_j

The fitness value of the above chromosome is calculated based on the comparison approach, where the first three bits in C_i and C_j are used as the index value to access the golden chromosome. The expected chromosome represented as C_{org1} and C_{org2} are compared with C_i and C_j respectively as shown in the Algorithm 1. The fitness value of C_i and C_j is recorded as 0.78% and 0.55%. Hence the above two chromosomes are selected based on the ranking method. A 3×3 crossover methodology is applied to chromosomes, and as a result of recombination, four new chromosomes Res[1-4] are produced, as shown in Fig. 10. The recombination is performed by segmenting the chromosome at each 3rd bit and reshuffled with the other chromosome as shown in Fig. 9.

The fitness value of the resultant chromosome Res [1-4] as shown in Fig. 10 will be calculated in the subsequent generations as 0.52%, NA, 100%, and 0.36%, denoting that the chromosome with positioning signal 110 (Res 3) has evolved to fault free convergence. The NA in the fitness calculation represents that the positioning signal 000 does not apply to the target circuit- a brushless DC motor. To demonstrate the mutation process assume that chromosome C_i is generated as 110-001-000-00-110-000-01-001-110-10-010001 with fitness value 0.96%. A uniform mutation is applied to the chromosome with a 0.03 mutation rate. The total number of the chromosome produced for C_i after mutation is 33 chromosomes, where each bit is mutated in C_i. For instance, as shown in Fig. 11 Res[1-33] are produced. The Res10 chromosome is evaluated as 100% fitness value, and other chromosomes are calculated as 0.95% in the next generation. The adopted mutation ensures that when a chromosome with 0.96% is applied for mutation, Then one of its child chromosomes will converge in the next generation. The proposed genetic algorithm follows elitism. The evolutionary process is terminated if the current input signal is converged.

3.4 Applications: Intrinsic EHW System for Control Circuit

This section details the proposed EHW system for mitigating the faults with four chosen control circuits. The

Res 1 101-000-110-10-101-010-11-011-111-10-101001
Res 2 000-101-101-11-010-101-01-111-011-01-110001
Res 3 110-000-000-10-110-010-10-101-111-01-101001
Res 4 001-110-011-00-011-010-10-101-000-11-010001

Fig. 10 Resultant chromosomes after crossover process

Res 1 010-001-000-00-110-000-01-001-110-10-010001
Res 2 100-001-000-00-110-000-01-001-110-10-010001
Res 3 111-001-000-00-110-000-01-001-110-10-010001
....
....
....
Res10 101-001-000-10-110-010-10-101-111-01-101001
....
....
Res 33 101-001-000-10-110-010-10-101-111-01-101000

Fig. 11 Resultant chromosomes after mutation process

chosen control circuit is utilized in various applications like space, robotic navigation, and consumer electronics. The case study profile explains the proposed EHW system's design phases, like hardware representation of the FSM, optimized functional evolution array, and genetic parameters. The control circuits analyzed in the section are the following:

1. Control circuit of Quadrature Decoder
2. Control circuit of Robotic straight line navigation.
3. Control circuit of RISC-V processor
4. Control circuit of Brushless DC Motor (BLDC) as explained in Sect. 3.3.4

3.4.1 Control Circuit of Quadrature Decoder

A Quadrature Decoder (QD) is used as a case study in [15] for testing their proposed evolutionary strategy. The QD is an incremental encoder for counting the entities passing to and from the light beam. The FSM of QD is tabulated as a state transition Table 4. The input of the FSM is two bits, which indicates the on/off condition of two-channel sensors (A and B), and the output is a two-bit combination that indicates the four directions of the wheel rotation. From the table, the output bits - S_1, S_2 is obtained using the Eq. 8. This Eq. (8) and Table 4 is the foundation for applying the heuristic to the optimized functional evolution array and the proposed heuristic-guided genetic algorithm. The single PE of the QD should incorporate the three multiplexers with three selection lines, as shown in Fig. 12. The FE array of the QD consists of 4×4 programming elements with input A, B, and output S_1, S_2. The first column of the

Table 4 State transition table of quadrature decoder

Channel A	Channel B	Direction(S_1,S_2)
1	0	00
1	1	01
0	1	10
0	0	11

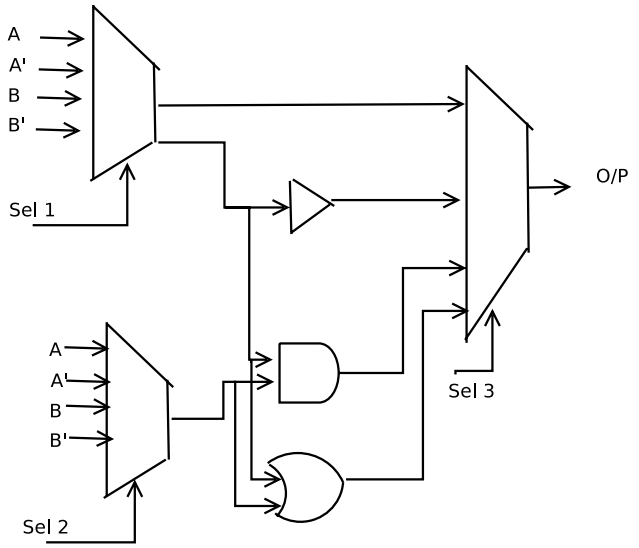

Fig. 12 Single PE representation of quadrature decoder

FE array obtains the input from external ($\overline{A}, \overline{B}$, A, B), and the final output is obtained from the last column. The total chromosome length for each input is 2+4*(2+2+2)+2=28 bit following the Eq. (7). Figure 12 shows the valid input combination for restricting the random population initialization at Generation 0. The fitness value of the generated chromosome C_i is obtained by comparing the expected result stored in the FlashROM following the Algorithm 1. The genetic operators in Sect. 3.3.3 are applied as such to rectify the control circuit from the fault.

$$S_1 = \overline{A}; S_2 = \overline{AB} + AB \tag{8}$$

3.4.2 Control Circuit of Robotic Straight Line Navigation

The fundamental task for robotics is line-following navigation. The control circuit for line navigation is a mealy machine since it accepts both the input and the current state to produce the next state. S1 and S2 sensors are used to detect the presence or absence of a line and three states. I1 and I2 denote the action or the state of the navigation that has to be followed. For instance, 00,01,10 in Table 5 indicates the state encoding for the robot to navigate straight, left, and right, respectively. The state transition table of the navigation is shown in Table 5. The minimized Boolean expression reduced from the transition table is S1=$\overline{I1}$ and S2=$\overline{I2}$. The proposed FE array consists of 8×2 programming elements, and each PE contains two multiplexers for selecting the input signal and the function (Sel 1, Sel 2). Since the complexity of the circuit is less, the total chromosome length is 4+2*(3+1)+2=14 bits.

Table 5 State Transition table of straight-line robotic navigation

Current State		Input		Next State	
S1	S2	I1	I2	S1	S2
0	0	0	0	1	1
0	0	1	1	0	0
0	0	0	1	1	0
0	1	0	0	1	1
0	1	1	0	0	1
0	1	1	1	0	0
1	0	0	0	1	1
1	0	1	1	0	0
1	0	1	1	0	0

3.4.3 Control Circuit of RISC-V Processor

The RISC-V processor is utilized as an instruction set architecture in multiple SoC-based FPGAs like the ARM-Cortex in Microsemi. The control circuit of the RISC-V architecture is modeled using a finite machine containing ten states (0-9), as shown in Table 6. States 1 and 2 depict the instruction fetch and instruction decode states, which accept the input (instruction opcode). Each state, on reaching the next state, sets the respective control signals to 1. The Boolean expression mentioned in Eqs. (9)-(12) obtained for the next state is a sample for a complex combinational circuit. The above circuit realized as a functional evolution array will contain 10*11 PE, where each PE will contain three multiplexers to select input and function. The possible function for each PE is 5; Sel 3 utilizes 3 bits, whereas Sel 1,2 utilizes 5 bits to select inputs 1 and 2. This example is specifically chosen to demonstrate the scalability of the proposed EHW system.

$$NS3 = \overline{S3S2S1S0P5P4P3P0}(P2 \oplus P1) \tag{9}$$

$$NS2 = (\overline{S2}(\overline{P4P2}(\overline{S1}S0\overline{P5P3P1P0} + S1\bar{S}0P5P3P1P0) + S1S0)) \tag{10}$$

$$NS1 = \bar{S3}(\overline{S2}(\overline{P4P2}(\overline{S1}S0(P5P1P0 + P5\bar{P}3\overline{P1P0})S1\overline{S0}P5\overline{P3}P1P0)) + S2S1\overline{S0} \tag{11}$$

$$NS0 = \overline{S3}(\overline{S2}(\overline{P4P2P1}(S1\overline{S0}P5P0 + \overline{S1}S0\overline{P5P3P0}) + \overline{S1S0}) + S2S1\overline{S0} \tag{12}$$

The summary of the optimized FE array for all the discussed control circuits is depicted in Table 7. Among the examples, the control circuit of the RISC-V processor occupies a larger area of the functional evolution array than the chromosome length, which is huge compared to other circuits. This control circuit is rarely used in EHW system-related work and was chosen specifically to test the scalability of the proposed EHW system. The analysis of the control

Table 6 State transition Table of RISC-V Processor

States	Present State	Input						Next state
		OP0	OP1	OP2	OP3	OP4	OP5	
0	0000	x	x	x	x	x	x	0001
1	0001	1	0	0	0	1	1	0010
1	0001	1	0	1	0	1	1	0010
1	0001	0	0	0	0	0	0	0110
1	0001	0	0	0	1	0	0	1000
1	0001	0	0	0	0	1	0	1001
2	0010	1	0	1	0	1	1	0011
2	0010	x	x	x	x	x	x	0101
3	0011	x	x	x	x	x	x	0100
4	0100	x	x	x	x	x	x	0000
5	0101	x	x	x	x	x	x	0000
6	0110	x	x	x	x	x	x	0111
7	0111	x	x	x	x	x	x	0000
8	1000	x	x	x	x	x	x	0000
9	1001	x	x	x	x	x	x	0000

circuit clearly indicates that the FE array and chromosome bit are directly proportional to the complexity of the boolean expression and the hardware utilization in terms of multiplexers for each PE. For complex circuits, bitstream and standard function level evolution will require high fault recovery time. When compared to the works discussed in [15] and [42], the chromosome length for QD and BLDC is reduced by 25% and 36.66%, respectively, using optimized function level evolution and heuristic guided genetic algorithm, which greatly accelerates the convergence of the self-healing process.

4 Implementation Methodology

The proposed EHW is operable in two states: non-faulty and faulty. In a non-faulty state, the fault in the control circuit is non-existent, and the error detection circuit sets GA status to 0. The output of the control circuit is signaled, such as the control signal and next state. In a faulty state, the error in the control circuit is detected by comparison of the training data stored in the FlashROM. The GA status signal is enabled to initiate the evolutionary algorithm. The non-faulty state is achieved on successful convergence, and the control circuit proceeds with regular operation.

The proposed EHW system is implemented on Microsemi-based ProAsic 3e family FPGA - A3PE3000. The FPGA was chosen because it is widely used in many avionics and military applications. The FPGA fabric does not feature any API for bitstream access or hold any microprocessor in the FPGA fabric. In addition, the bitstream in the configuration memory is encrypted, which challenges the bitstream evolution.The proposed heuristics-guided GA module, error detection circuit, FlashROM, and target circuit are implemented on the same FPGA fabric operating at 350 Mhz with a maximum combination delay of 2.23 ns. The minimum input arrival time and output required time before and after the clock are 3.45 ns and 2.78 ns, respectively.

The proposed design is subjected to analysis based on the metrics in such a way that the complete EHW system is studied. Since EA is an iterative algorithm, the algorithm's termination marks the solution's convergence. The main focus of this work is to increase the scalability by optimizing the VRC. Hence, the resource utilization is analyzed compared to similar methodologies mentioned in related work. A complete fault injection system is simulated and analyzed with the help of maximum and minimum generation in fault detection efficiency. Any fault tolerance mechanism has to be rapid; hence, our analysis records the fault recovery time from faulty to non-faulty state.

Table 7 Summary of Optimized Functional evolution array

Control Circuit Example	# Input Bits	#Output Bits	FE Array Utilization (#m*#n)	# Chromosome bits
Quadrature Decoder	2	2	4*4	28
Navigation Robot	4	4	8*2	14
RISC-V	10	4	11*10	144
BLDC	3	6	6*3	33

Resource Utilization: Area power and energy consumed for the proposed system are compared with results obtained in [15] for QD and BLDC, the comparison is obtained for the following works [28, 42] where standard function level evolution and dynamic partial reconfiguration are utilized respectively. The area comparison for the proposed and related is accounted for in terms of the number of LUT, registers, and FlashROM utilized. The energy and power consumption of the proposed system are estimated by deploying the hardware module in LiberoSOC 11.8 estimation tools.

Fault Detection Efficiency is calculated by subjecting the target circuit to simulated faults at the PE level. The stuck-at 0/1 fault and single/multiple bit upset errors are injected in the selection lines of the multiplexers. The fault detection rate is estimated by dividing the total number of PE retrieved from faults after evolution by the number of PE injected with faults. The convergence rate increases in the repair process also have to be considered in determining the efficiency of the evolvable system.

Fault Recovery Rate is the mean time taken by the system to repair the faulty state to a non-faulty state, calculated by the Eqs. 13 and 14, where $T_{access}+T_{fit}+T_{EA}$ is the time consumed for accessing the training example chromosome, time for calculating the fitness value and time for computing genetic operations like selection, mutation, and crossover for each chromosome, and total recovery time is calculated by the summation of time taken for each generation.

$$T\_gen_{\mathrm{i}} = Pop\_size * (T_{\mathrm{access}} + T_{\mathrm{fit}} + T_{\mathrm{EA}}) \tag{13}$$

$$Total\_time = \sum_{i=0}^{Gen} T\_gen_{\mathrm{i}} \tag{14}$$

5 Results and Discussion

This paper proposes a novel EHW system that incorporates improvements to the reconfigurable layer and the evolutionary algorithm. The deterministic nature of control circuits is extensively demonstrated with the help of controllers used in various digital electronics applications, as discussed in Sects. 3.4.1-3.4.3. The convergence of the proposed Heuristic Guided Genetic Algorithm (HGA) is compared with the Standard Genetic Algorithm presented in [15] and [42] for quadrature decoder and brushless DC motor, respectively, in Fig. 13(A) and (B). According to the convergence graph, if heuristics guides genetic operations, the termination and healing time in the number of generations is drastically reduced by 10 and 50 times, respectively. The guided population initialization and reduced chromosome bits compared to [15] and [42] has accelerated the convergence of the proposed HGA, which is essential for self-repairing mission-critical components like control circuits. In addition to the reasons for the accelerated convergence rate, elitism can also influence convergence, as mentioned earlier. In our proposed genetic algorithm, elitism is adopted where the convergence of the current signal is given higher priority in the evolution process. Although elitism applies selective pressure to convergence, it cannot guarantee that the current input signal will evolve before other signals. The Fig. 13(C) and (D) show the convergence of the navigation robot and the RISC-V processor to investigate the scalability of the proposed HGA, which is regarded as a challenge in the related work. The graph depicts that the fitness value grows drastically during the initial generation, whereas once the fitness value reaches close to 0.8, the convergence becomes slow and reaches a termination when the fitness value reaches 100%.

Tables 8 and 9 summarise the fault injection profile implemented to study the fault detection efficiency. Two types of fault injection locations accounting for routing and functionality are chosen. The random bits are selected by the fault injection profile implemented as a module in the Hardware additional to the EHW system. The fault-free convergence rate in Tables 8 and 9 denotes the average number of generations at which the evolution converges when zero error is injected. Tables 8 and 9 also denote the number of PE affected due to the faults injected with the minimum, maximum, and average generation required for mitigating the faults. The results clearly show that the stuck at 1/0 fault entails a higher convergence rate when compared to a single event upset because the number of PE under repair increases with the number of bit interchanges. In QD, for example, the stuck at 0, and 1 fault results in errors at 2 and 5 programming elements, accounting for 50% of the total functional evolution layer, whereas a single event upset fault results in faults at two programming elements, accounting for 16%. The faults in the input selection line drastically increase the faulty PE because the fault in the input level can propagate the error and increase the repair at the PE selected at each column, thereby initiating the routing error. The function selection line fault may not directly influence the faulty PE rate, but the final control signal and next state are altered, which can affect the functionality of the control circuit. Thus, routing and function level faults can directly influence the control circuit's operation. The proposed EHW system can detect and correct errors for all types of errors tabulated in Tables 8 and 9.

The resource cost analysis of three implementations is presented in Table 10. The proposed optimized function level evolution is compared with standard VRC conducted entirely at the hardware level. The analysis shows that the hardware utilization of registers, LUTs, and FlashROM is relatively higher when standard VRC is followed because of the increase in the multiplexer at each PE level. The increase in multiplexer can increase the selection line and the

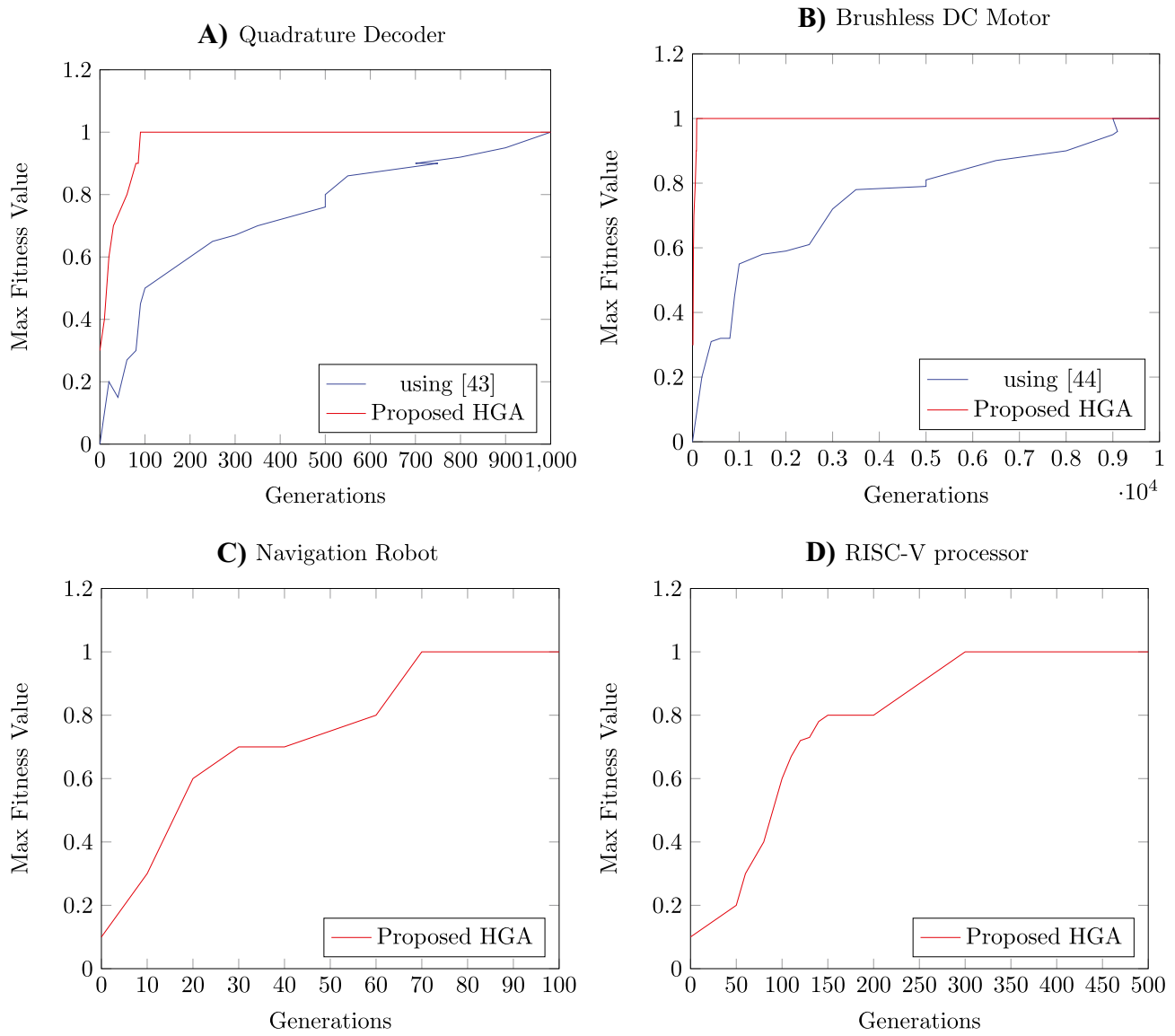

Fig. 13 Convergence comparison of proposed HGA and SGA

complexity of each PE, which is an essential factor in deciding the chromosome length stored in Flash-ROM. In our proposed EHW system, the GA module is written in HDL and implemented as a digital circuit. This implementation is compared with the hardware/software implementation of the GA module, where the modules in the genetic algorithm are written in c script and hosted on the M1A3PE3000. This FPGA is a variant of the A3PE3000 with the availability of an ARM cortex processor. The control circuit for fault recovery is implemented using standard VRC. Since the processor is utilized in the evolution, the number of registers for holding the chromosome for genetic operations is less when compared to hardware evolution. The processor communicates with the AXI bus operating at 350 Mhz to send the evolved chromosome to standard VRC.

The box plot graph of execution time shown in Fig. 14 depicts the total execution time for self-repairing the control circuit. The values depicted in the graph are calculated by the Eqs. (13) and (14) for 30 runs. The red box and blue box denote the fault recovery time of the Complete Hardware Evolution (CHE) and hybrid evolution of respective control circuits. The figure demonstrates that the fault recovery time of intrinsic(CHE) is 30% × faster than the hybrid evolution on average. The speed acceleration of complete hardware evolution is possible because the GA module requires fewer clock cycles to communicate the evolved chromosome to the target circuit than hybrid evolution. The usage of the AXI bus to transfer the evolved circuit has an operation limitation of 250 MHz. The inherent parallelism of the FPGA facilitates the high-speed operation of genetic operations

Table 8 Comparison of Fault detection efficiency with convergence rate for SEU and MBU

Fault Location	Control circuit	Fault-Free Convergence Rate	SEU				MBU			
			# PE injected with Fault	Min Generation	Max Generation	Average Generation	# PE injected with Fault	Min Generation	Max Generation	Average Generation
Input Select Line	QD	120	2	111	183	147	5	93	233	163
	BLDC	292	4	207	419	313	7	108	234	361
	NR	74	2	28	146	87	4	32	235	93
	RISC-V	876	8	345	1339	842	7	234	236	896
Function Select Line	QD	120	3	87	175	131	5	47	247	147
	BLDC	292	3	112	542	327	8	127	605	366
	NR	74	2	31	153	92	3	54	198	126
	RISC-V	876	11	236	1426	831	9	368	1514	941

Table 9 Comparison of Fault detection efficiency with convergence rate for stuck-at faults

Fault Location	Control circuit	Fault-Free Convergence Rate	Stuck-at-0				Stuck-at-1			
			# PE injected with Fault	Min Generation	Max Generation	Average Generation	# PE injected with Fault	Min Generation	Max Generation	Average Generation
Input Select Line	QD	120	3	121	311	216	4	112	352	232
	BLDC	292	8	101	603	352	11	145	545	345
	NR	74	6	37	169	103	7	31	159	95
	RISC-V	876	11	578	2356	1467	9	345	2119	1232
Function Select Line	QD	120	2	76	286	181	5	49	255	152
	BLDC	292	9	156	668	412	12	112	580	346
	NR	74	6	51	191	121	8	45	179	112

Table 10 Resource utilization of control circuit profile

Application	Resource	Available	Complete Hardware Evolution using Optimized FE		Complete Hardware Evolution using Standard VRC		Hybrid Evolution in Microprocessor (M1A3PE3000) using Standard VRC	
			Usage	Utilization%	Usage	Utilization%	Usage	Utilization %
Quadrature Decoder	Register	7890	78	0.988	85	1.07	56	0.70
	LUT	75264	6	0.729	10	0.78	10	1.3
	BRAM	112	0	0	0	0	0	0
	FlashROM	1024	490	47.8	674	65	576	56.2
	IO	620	9	1.4	9	1.4	19	3.0
	Clock processor	667	-	-	-	-	667	NA
Brushless Dc Motor	Register	7890	87	1.11	102	1.29	69	0.87
	LUT	75264	16	0.212	23	0.305	18	0.2
	BRAM	112	0	0	0	0	0	0
	FlashROM	1024	524	51	654	638	789	77
	IO	620	17	2.7	17	2.7	21	3.39
	Clock processor	667	-				667	-
Navigation Robot	Register	7890	45	0.57	59	0.74	32	0.40
	LUT	75264	4	0.5314	6	0.5123	8	0.7890
	BRAM	112	0	0	0	0	0	0
	FlashROM	1024	123	12.0	345	33.6	415	40.5
	IO	620	6	0.96	6	0.96	10	1.61
	Clock processor	667	-		-		667	-
RISC-V Processor	Register	7890	678	8.59	750	9.50	560	7.09
	LUT	75264	121	0.16	104	0.158	98	0.132
	BRAM	112	0	0	0	0	0	0
	FlashROM	1024	780	76.1	970	94	813	67
	IO	620	45	7.2	45	7.2	56	9.0
	Clock processor	667	-	-	-	-	667	-

when deployed as a digital circuit. As a result of the above analysis, complete hardware evolution accelerates the evolution process compared to extrinsic and hybrid evolution with a limited increase in register utilization. However, hybrid evolution is more suitable than hardware implementation for applications where flexibility is mandatory for the genetic algorithm.

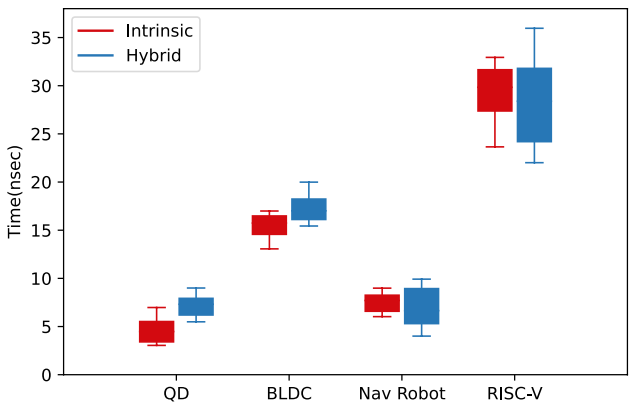

Fig. 14 Execution time for self-repairing the control circuits

The fault tolerance or the convergence of the proposed EHW system was not achievable when faults in the PE of the first column of the optimized FE array were more significant than 75% of the PE in this column. The main reason for this non-convergence scenario is that when more than 75% of PEs are affected due to fault injection in the first column, the propagation of all three input signals is not ensured. Due to this, the fault-free circuit will not evolve. This situation has not occurred in our 30 trials of experimentation, whereas it was identified during theoretical analysis. The other limitation of the EHW system is that when a genetic algorithm is implemented as a digital circuit on the same FPGA, there are chances for faults to occur in the genetic algorithm, which can mislead the mitigation process. In our future work, additional responsibilities will be to protect both the target circuit and the implemented genetic algorithm using redundancy methods.

6 Conclusion

Self-healing electronics are the need of the hour as the requirement for FPGA usage in critical systems increases. The faults in these critical systems' components must be detected and mitigated expeditiously. The traditional methods like TMR and hamming distance provide the system with reliability but with an increase in area and delay. Hence, in our work, we have imitated bio-organisms capability of positioning faults and removing them using evolvable Hardware. The proposed EHW system is a complete hardware-level evolution in which the genetic algorithm is deployed on the same FPGA and the target control circuit. This intrinsic implementation of the algorithm has contributed to accelerating the execution time of fault repair on an average by 30% when compared to hybrid or SoC-based evolution. In addition, the standard genetic algorithm has been modified by applying heuristics from the behavior model (state transition table) to reduce the convergence of the healing process in terms of the number of generations. As a result, the number of generations is reduced by 47% on average compared to the current work. The resource utilization in terms of the number of LUTs has decreased by 7.5 × compared to standard VRC when the proposed optimized functional evolution is utilized. The above results demonstrate that the proposed EHW system can absolutely mitigate the faults occurring in the control circuit. In future work, the proposed EHW system's scalability must experiment with multiple complex control circuits in addition to the RISC-V processor discussed. The memory occupancy to store the training example for fitness calculation in FlashROM is high compared to hybrid evolution. Our future work will investigate alternative efficient measures for storing the training example.

Data Availability Data sharing does not apply to this article as no data sets were generated or analyzed during the current study.

Declarations

Conflict of Interest/Competing Interest The authors have no conflicts of interest to declare relevant to this article's content.

References

1. Álvarez I, Proenza J, Barranco M, Knezic M (2017) Towards a time redundancy mechanism for critical frames in time-sensitive networking. In: Proc. 22nd IEEE International Conference on Emerging Technologies and Factory Automation (ETFA), pp 1–4. https://doi.org/10.1109/ETFA.2017.8247721
2. Asselmeyer T, Ebeling W, Rosé H (1997) Evolutionary strategies of optimization. Phys Rev E 56:1171–1180. https://doi.org/10.1103/PhysRevE.56.1171
3. Bergmann NW, Sutton PR (1998) A high-performance computing module for a low earth orbit satellite using reconfigurable logic. In: Proc. International Workshop on Field Programmable Logic and Applications, Springer, pp 416–420
4. Bouhali M, Shamani F, Dahmane ZE, Belaidi A, Nurmi J (2017) FPGA applications in unmanned aerial vehicles-a review. In: Proc. International Symposium on Applied Reconfigurable Computing, Springer, pp 217–228
5. Carmichael C, Fuller E, Blain P, Caffrey M (1999) SEU mitigation techniques for virtex FPGAs in space applications. In: Proceeding of the Military and Aerospace Programmable Logic Devices International Conference (MAPLD), p C2
6. Das S, Tokunaga C, Pant S, Ma WH, Kalaiselvan S, Lai K, Bull DM, Blaauw DT (2008) Razorii: In situ error detection and correction for PVT and SER tolerance. IEEE J Solid-State Circuits 44(1):32–48
7. El-Maleh AH, Al-Qahtani AS (2014) A finite state machine based fault tolerance technique for sequential circuits. Microelectron Reliab 54(3):654–661
8. Garnica O, Glette K, Torresen J (2018) Comparing three online evolvable hardware implementations of a classification system. Genet Program Evolvable Mach 19(1):211–234
9. Holland JH (1992) Genetic algorithms. Sci Am 267(1):66–73
10. Keymeulen D, Zebulum RS, Jin Y, Stoica A (2000) Fault-tolerant evolvable hardware using field-programmable transistor arrays. IEEE Trans Reliab 49(3):305–316
11. Koza JR, Bennett FH, Andre D, Keane MA, Dunlap F (1997) Automated synthesis of analog electrical circuits by means of genetic programming. IEEE Trans Evol Comput 1(2):109–128
12. Kumar U, Umashankar B (2007) Improved Hamming code for error detection and correction. In: Proc/ 2nd International Symposium on Wireless Pervasive Computing, IEEE
13. Langeheine J, Becker J, Folling S, Meier K, Schemmel J (2001) A CMOS FPGA chip for intrinsic hardware evolution of analog electronic circuits. In: Proceedings Third NASA/DoD Workshop on Evolvable Hardware. EH-2001, IEEE, pp 172–175
14. Lohn J, Larchev G, DeMara R (2003a) A genetic representation for evolutionary fault recovery in virtex FPGAs. In: Tyrrell AM, Haddow PC, Torresen J (eds) Evolvable Systems: From Biology to Hardware, Springer Berlin Heidelberg, Berlin, Heidelberg, pp 47–56
15. Lohn J, Larchev G, DeMara R (2003b) A genetic representation for evolutionary fault recovery in virtex FPGAs. In: Proc. International Conference on Evolvable Systems, Springer, pp 47–56
16. Ma X, Sun H, Xu E, Cui S, Yin B, Faied M (2020) FSM for robot target search and retrieval under semi-constructed environment. In: Proc. IEEE International Conference on Mechatronics and Automation (ICMA), pp 296–301
17. Miller JF (1999) An empirical study of the efficiency of learning Boolean functions using a cartesian genetic programming approach. In: Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation - Volume 2, Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, GECCO'99, p 1135–1142
18. Ortega-Sánchez C, Tyrrell A (1998) Muxtree revisited: Embryonics as a reconfiguration strategy in fault-tolerant processor arrays. In: Sipper M, Mange D, Pérez-Uribe A (eds) Evolvable Systems: From Biology to Hardware. Springer, Berlin Heidelberg, Berlin, Heidelberg, pp 206–217
19. Pratt B, Caffrey M, Carroll JF, Graham P, Morgan K, Wirthlin M (2008) Fine-grain SEU mitigation for FPGAs using partial TMR. IEEE Trans Nucl Sci 55(4):2274–2280
20. Pratt B, Caffrey M, Graham P, Morgan K, Wirthlin M (2006) Improving FPGA design robustness with partial TMR. In: IEEE International Reliability Physics Symposium Proceedings, pp 226–232. https://doi.org/10.1109/RELPHY.2006.251221
21. Rochet R, Leveugle R, Saucier G (1993) Analysis and comparison of fault tolerant FSM architecture based on SEC codes. In:

Proceedings of IEEE International Workshop on Defect and Fault Tolerance in VLSI Systems, pp 9–16
22. Ruano O, Maestro JA, Reviriego P (2009) A methodology for automatic insertion of selective TMR in digital circuits affected by SEUs. IEEE Trans Nucl Sci 56(4):2091–2102. https://doi.org/10.1109/TNS.2009.2014563
23. Sekanina L (2003) Virtual reconfigurable circuits for real-world applications of evolvable hardware. In: Proc. International Conference on Evolvable Systems, Springer, pp 186–197
24. Sekanina L, Drábek V (2000) The concept of pseudo evolvable hardware. IFAC Proceedings Volumes 33(1):117–122
25. Silva GNP, de Oliveira Duarte R (2018) Towards evolvable hardware and genetic algorithm operators to fail safe systems achievement. In: Proc. IEEE 19th Latin-American Test Symposium (LATS), pp 1–4
26. Silva GNP, Duarte RO (2018) Towards evolvable hardware and genetic algorithm operators to fail safe systems achievement. In: Proc. IEEE 19th Latin-American Test Symposium (LATS), pp 1–4
27. Skorobogatov S, Woods C (2012) Breakthrough silicon scanning discovers backdoor in military chip. In: Prouff E, Schaumont P (eds) Cryptographic Hardware and Embedded Systems - CHES 2012. Springer, Berlin Heidelberg, Berlin, Heidelberg, pp 23–40
28. Srivastava AK, Gupta A, Chaturvedi S, Rastogi V (2014) Design and simulation of virtual reconfigurable circuit for a fault tolerant system. In: Proc. International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014) pp 1–4
29. Stomeo E, Kalganova T, Lambert C (2006) Generalized disjunction decomposition for the evolution of programmable logic array structures. In: Proc. First NASA/ESA Conference on Adaptive Hardware and Systems (AHS'06), IEEE, pp 179–185
30. Suhas S, Malhotra G, Rajini VH (2021) HsClone genetic algorithm implementation on a combinational circuit. IETE J Res pp 1–9
31. Thompson A (1995) Evolving fault tolerant systems. In: Proc. First International Conference on Genetic Algorithms in Engineering Systems: Innovations and Applications, IET, pp 524–529
32. Thompson A, Harvey I, Husbands P (1996) Unconstrained evolution and hard consequences. In: Towards evolvable hardware, Springer, pp 136–165
33. Thompson A, Layzell P, Zebulum RS (1999) Explorations in design space: Unconventional electronics design through artificial evolution. IEEE Trans Evol Comput 3(3):167–196
34. Tufte G, Haddow PC (2000) Evolving an adaptive digital filter. In: Proc. Second NASA/DoD Workshop on Evolvable Hardware, IEEE, pp 143–150
35. Tyrrell AM, Hollingworth G, Smith SL (2001) Evolutionary strategies and intrinsic fault tolerance. In: Proceedings Third NASA/DoD Workshop on Evolvable Hardware. EH-2001, IEEE, pp 98–106
36. Vasantha Rani SPJ, Ranjith NA (2020) Performance analysis of intrinsic embedded evolvable hardware using memetic and genetic algorithms. Int J Bio Inspir Com 15:43–51
37. Wang JJ (2003) Radiation effects in FPGAs. https://doi.org/10.5170/CERN-2003-006.34
38. Wang J, Kang J, Hou G (2019) Real-time fault repair scheme based on improved genetic algorithm. IEEE Access 7:35805–35815. https://doi.org/10.1109/ACCESS.2019.2905042
39. Wang J, Liu J (2017) Fault-tolerant strategy for real-time system based on evolvable hardware. J Circuits Syst Comput 26(07):1750111
40. Zhang J, yan Cai J, Meng Y, Meng T (2020) A novel self-adaptive circuit design technique based on evolvable hardware. Int J Autom Comput pp 1–8
41. Zhang W, Li Y, He G (2007) Intrinsic evolution of frequency splitter with a new analog EHW platform. In: Proc. International Symposium on Intelligence Computation and Applications, Springer, pp 611–620
42. Zhu P, Yao R, Du J (2017) Design of self-repairing control circuit for brushless DC motor based on evolvable hardware. In: Proc. NASA/ESA Conference on Adaptive Hardware and Systems (AHS), IEEE, pp 214–220

S Deepanjali has obtained her B.Tech. from RMK Engineering College, Anna University, Chennai, M.Tech. from SSN College of Engineering. She is currently pursuing her PhD at the department of Computer Science and Engineering, Indian Institute of Information Technology Design and Manufacturing Kancheepuram. Her research interest is Evolvable hardware and fault Tolerant computing.

Noor Mahammad Sk has obtained his PhD from Indian Institute of Technology Madras. He is currently working as Associate professor in the department of Computer Science and Engineering, Indian Institute of Information Technology Design and Manufacturing, Kancheepuram, Chennai. His research interest are evolvable hardware and reconfigurable computing.