Journal of Electronic Testing (2023) 39:111–122
https://doi.org/10.1007/s10836-023-06050-1

Intrinsic Based Self-healing Adder Design Using Chromosome Reconstruction Algorithm

Raghavendra Kumar Sakali[1] · **Noor Mahammad Shak**[1]

Received: 16 December 2022 / Accepted: 29 January 2023 / Published online: 29 March 2023

Abstract
Evolvable hardware-based fault-tolerant hardware design is an efficient approach to self-adaptability. It is an essential feature to mitigate errors on the fly. But there are two issues while designing an adder using the evolutionary hardware (EHW) approach: scalability issues in circuit representation and a low error recovery speed due to many evolutions. To avoid scalability issues, we designed an optimized virtual reconfiguration circuit (VRC) for adder. In this paper, we introduce the chromosome reconstruction algorithm for evolving the circuit to recover faults in an adder circuit at a faster speed. The proposed self-healing adder design is implemented on a single FPGA using an intrinsic approach. The complete hardware is designed on a Proasic A3PE3000 FPGA. Compared to existing work, the proposed work's resource utilization is optimal.

Keywords Self-healing hardware · Evolvable hardware · Adder · FPGA · SEU

1 Introduction

FPGAs are prominent devices in the semiconductor area due to their programmability. Due to the dynamic programmability feature, these devices are adaptable to any specific system on the fly. FPGAs can also be capable of partial reconfiguration; only specified hardware parts of the circuit can be modified at runtime [25]. So, many industries, such as commercial, defense, and medical, are interested in implementing their work on FPGAs. These devices are achieving good computing performance with ease of implementation. But it has limitations as the FPGA is an electronic device. This device is sensitive to errors. Errors are classified as either transient (soft) or hardware (permanent) errors. FPGAs are affected by transient errors. Transient errors have occurred due to environmental conditions such as cosmic rays and electromagnetic interference [27]. For example, these errors may mutate the information bits in memory elements. These are known as single-event upsets. A fault-tolerant technique is the best approach to suppress upsets. The fault-tolerant system can be efficiently implemented on the FPGA because of its reprogrammable and reusable capabilities. As a result, fault-tolerant FPGAs are the best option for radiation environments.

Fault tolerance is a significant factor in various fields for the sophisticated working condition of a system because it involves detecting and correcting failure points automatically. Fault tolerance can be grouped into active and passive fault tolerance [8, 32]. The active fault tolerance mechanism uses the information to mitigate system faults and reconfigures based on approaches. Evolvable hardware and reconfiguration techniques are parts of active fault tolerance. The traditional fault-tolerant methods like redundancy and self-reconfiguration techniques are subsets of passive fault tolerance. The evolvable hardware will quell the limitations of conventional approaches with self-repair and self-adaptability in the fault tolerance mechanism. The redundancy technique will require extra hardware to mitigate the faults in the circuit [26]. The self-reconfiguration technique has a more extended delay in recovering the standard functionality of the system. These two are drawbacks to error recovery in the design, but its working will be simple and effective.

Evolutionary hardware (EHW) [3, 14] is self-adaptive hardware inspired by natural evolution. The term EHW has appeared in electronic research works from the early '90s because of Hugo De Garis. The motive of self-adaptive

Responsible Editor: A. Yan

✉ Noor Mahammad Shak
noor@iiitdm.ac.in

Raghavendra Kumar Sakali
CS21D0002@iiitdm.ac.in

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

hardware is to adapt to changes in the computational behavior based on dynamic self-reconfiguration, which will quell faults and degradation of the hardware by recovering unexpected functionalities and disruptions from the environment. Order to achieve motive requires programmable architecture [30] and evolutionary algorithms [6]. Despite the limitations of traditional methods, such as size/power, component types, and fault tolerance, EHW has accomplished a great deal in real-world engineering applications. EHW uses numerous algorithms to evaluate the effectiveness of circuits, such as genetic algorithms and memetic algorithms. Extrinsic or intrinsic evolutions [1] is used to determine the chromosome of the circuit. An extrinsic approach is a software-based approach where the evaluation of chromosomes is carried out using the simulator. The intrinsic method evaluates the chromosomes on the targeted FPGA itself because of their programmability and compatibility with EHW. A virtual reconfigurable circuit allows the evolution of evolvable hardware in a conventional FPGA. In a virtual reconfiguration circuit (VRC), an FPGA creates a virtual reconfigurable hardware layer. It is known as "programmable architecture" for EHW.

SRAM-based FPGAs have the best features for EHW implementation [11], many researchers have used Xilinx-based FPGAs such as the Virtex series, Spartan, and Zync-7000 [4, 23]. The majority of EHW projects have used either the DPR mechanism or the VRC mechanism with an extrinsic approach. The EHW work was implemented on FPGAs, which had three components: a microprocessor, an evaluation unit, and an AXI bus. The microprocessor hosts the genetic algorithm, which was designed using a native high-level language. An evaluation unit was used to identify the fault in a targeted circuit. The evaluation was compared with a copy of the targeted circuit. An AXI bus was an interface between the processor and the evaluation unit. In addition to this, the ICAP controller was used as an API. With the assistance of an ICAP controller, users can modify the configuration bits to change the circuit structure and functionality during the evolution process.

The addition is an essential operation in digital systems. The adder is a combinational circuit designed with digital gates such as AND, XOR, and OR. The combinational circuit is delicate and prone to a single-event upset [16]. This might have resulted in inaccurate outcomes. To resolve this issue, it required a fault-tolerant hardware design. Traditional fault-tolerant adder design necessitates additional hardware and increases power consumption [12]. So we explored a self-healing fault-tolerant approach to avoid the limitations of conventional methods. In this paper, an adder is designed using an optimized VRC technique to address scalability concerns. This adder is fault-tolerant thanks to human-inspired techniques. An efficient error detection approach is also introduced to reduce hardware scalability. We are concerned with error recovery speed in our design to increase efficacy in the proposed self-heal based fault-tolerant adder.

The rest of the paper is categorized as follows. Section 2 provides detailed information on the importance and limitations of existing works. Section 3 determines the design of the proposed VRC-based Adder. Section 4 explains the self-healing adder design. Section 5 illustrates the experimental setup and results of the proposed work. Section 6 describes the proposed work's advantages compared to the existing ones. The conclusion of the work is presented in Section 7.

2 Related Work

Evolvable hardware is an essential topic in self-healing and self-adaptable circuit design. EWH uses bio-inspiring approaches. So far, many researchers have worked in this field in various ways. The researchers had designed fault-tolerant evolvable hardware using various reconfigurable hardware, different evolutionary algorithms, diverse reconfiguration techniques, and different circuit evaluation processes based on their requirements.

Many evolvable hardware circuits were designed on Xilinx XC6200 FPGAs. These FPGAs were the first commercial FPGAs with bitstream readability access and were provided publicly for research purposes. But these FPGAs were stopped due to security issues. Later, most of the evolvable hardware circuits were implemented on Xilinx SRAM-based FPGAs such as the Virtex series and Spartan series. Due to the use of JBITS, the reconfiguration of EHW became complicated after the shutdown of XC6200 FPGAs.

Later, FPGAs were insufficient for EHW as they did not have the feature of partial reconfiguration. Sekanina [17] introduced the virtual reconfiguration circuit for this problem. VRC is the top-layer architecture of an FPGA implemented in HDL. Sekanina and Friedl [18] proposed an FPGA-based evolvable combinational circuit. The authors designed adder and multiplier circuits with virtual reconfigurable circuits to simplify evolution. Some things could have been improved, such as no carry being generated and two output bits remaining with logic zero in implementing the two 3-bit adders. This resulted in inaccurate output. Another drawback is insufficient resources; it is difficult to evolve the circuit in FPGA as the input size of the circuit increases. Sekanina [19] explained the reliability of virtual reconfigurable circuits and implemented a 1-bit full adder and 2-bit multiplier to recover the faults with an evolutionary algorithm using the VRC architecture. Vasicek et al. [24] stated the limitations of evolutionary combinational circuits, such as the scalability of representation and evolution. They implemented an FIR filter with multiple constant multipliers composed of adders, shifters, and subtractors.

The authors [13] implemented a fault-tolerant adder using evolvable hardware. The bitstream of the adder circuit evolved with a genetic algorithm to recover the faults. The complete process was a direct manipulation approach. The evolution of the adder took around thirty minutes to recover the faults. The Xilinx Virtex II Pro FPGA was used to implement the experimental setup. Cancare et al. [2] proposed the evolution of digital circuits using an evolutionary algorithm and a dynamic partial configuration technique. It was implemented on the Xilinx Virtex-5 FPGA. The evolution time of the adder was around 45 minutes to recover the faults, and for the 9-bit adder, it took about 10 hours. The authors have tried to reduce the scalability issue. Here, as the circuit's input size increases, the circuit's scalability also increases.

Salvdor et al. [15] designed a fault-tolerant circuit using DPR method. They implemented the circuit on Xilinx Vertex FPGA. These FPGAs can read and write configuration bitstream through the HWICAP port. But, this facility has a drawback like data loss or corruption of the chip. Some vendors don't manufacture the FPGAs which support the DPR; to these FPGAs, VRC is the best approach for implementing evolutionary-based circuits. Silva et al. [21] have designed a combinational circuit using evolutionary research and artificial neural networks.

Wang et al. [28] have introduced an evolutionary fault-tolerant approach using a genetic algorithm and improved its efficiency using self-adaptive sampling model, which is adapted from the cartesian genetic algorithm and employed this approach on the 2-bit multiplier. Jian and Mengfei [7] implemented a 2-bit full-adder with 176 configuration bits. The adder was designed based on neural-network architecture using VRC. This VRC adder scales up the memory space to accumulate the configuration bits for a longer-length adder. The adder was evolved using the extrinsic approach to avoid faults in it. The VRC adder was implemented on Xilinx Virtex. The evolutionary algorithm, fault-detection module, and evolution module were executed on Xilinx Spartan3 to recover the faults in the adder. Another limitation is hardware overhead because two chips were used to implement the fault-free circuit.

Mora et al. [11] have compared two different topologies: systolic array (SA) and Cartesian genetic programming (CGP), and analyzed which algorithm would scale more hardware resources. The authors concluded that CGP scaled resources 60% more than the SA, and both have similar computational performance. Shang et al. [20] digital circuits using evolvable hardware. Circuits were implemented using a hybrid intrinsic approach. The evolution of the circuit was done using the AGA algorithm to improve the convergence rate. The experimental work was implemented on the Intel Cyclone V SOC. The evolutionary algorithm was executed on an ARM core, and the VRC was designed on an FPGA. But the authors had yet to mention evolution time or configuration bit length. These are the two necessary parameters to calculate the efficiency of an evolvable and its convergence rate.

From these existing works identified some critical issues such as scalability of representation, evolution time, and dependability. These are significant parameters to manage during the construction of fault-tolerant hardware. The detailed description of each parameter and fault-tolerant adder construction will be explained in the following sections.

Contribution of the proposed work states as follows:

- The scalability issue in the existing VRC adder was addressed with the optimized VRC adder.
- The optimized VRC adder requires less hardware area. It efficiently utilizes the available resources in the FPGA.
- The optimized VRC adder's configuration bitstream will be shorter than previous works, improving the circuit's evolution speed.
- The novel chromosome reconstruction algorithm will accelerate the error recovery rate.
- The optimized VRC adder, error detection unit, and chromosome reconstruction unit were deployed as digital circuits on a single FPGA to avoid dependability and reduce the delay.
- The proposed work was tested with single-bit and multi-bit errors using an error injection simulator to estimate its efficiency and error recovery speed.

3 Proposed VRC Adder

As for FPGA devices, the minimal functional units controlled by configuration are logical gates, which make the device extremely flexible and able to implement all the logic with specified numbers of inputs and outputs. In the FPGA, configuration size is too large during the evolutionary operation; it will be problematic to handle and can face scalability issues [5]. This limitation can be achieved using the VRC approach. It is commonly constructed as an artificial array of PEs built on top of an FPGA, forming a virtual reconfigurable device layer. The functionality of each PE is designed based on practical applications. VRC has features of a knowledgeable configuration format and a coarse-grained array of elements. Thus, VRC is best suitable for complex circuit evolution.

3.1 VRC Adder Design

The VRC-based Adder is built with programmable elements (PE). The programmable elements were arranged in a $N \times M$ matrix. The implementation of PE required three multiplexers (MUXs). Among these, two MUXs were used to select

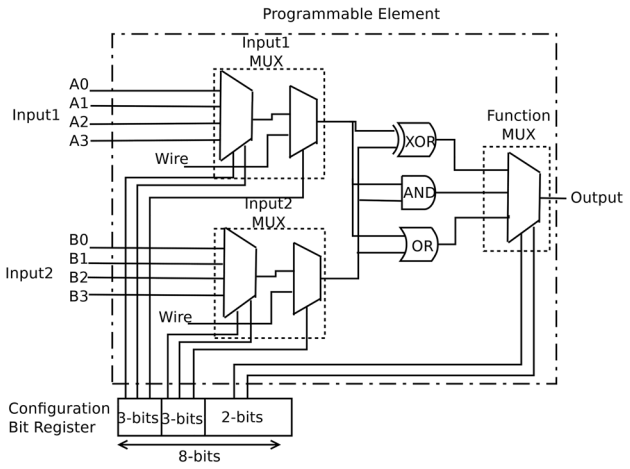

Fig. 1 Programmable Element of VRC Adder

the inputs, and one MUX was used to select the logical function. The adder circuit is designed with XOR, AND, and OR logical functions. These functions were given as inputs to the logical function MUX. The size of this MUX is 4:1, and two select lines are required to choose the function. The logical function MUX will be kept at the constant size for any input length, as shown in Fig. 1. The input-based MUX will change its size based on the input length of the adder. This MUX is required as an N+1:1 MUX for implementation. Here, N is the number of input bits, and the extra "1" is the wire which is used to provide input from the output of a previous PE. Based on the adder's input size, the input-based MUX selection lines will be increased. All PEs would require a common register to store configuration bits. Based on these configuration bits, the complete VRC adder will work. The architecture of the programmable element is shown in Fig. 1. The four-bit adder is designed using the proposed programmable element as shown in Fig. 2.

3.2 VRC Adder Routing and Configuration Bitstream

The routing of the internal circuits in the adder was controlled using MUXs through the configuration bits. These bits are stored in a configuration register and provided as input to selection lines of MUX to perform the operation. In the N-bit adder design, two PEs were required to design a half-adder for zeroth bit input. Five PEs were needed to develop the full adder for remaining (1^{st} *to* $(N-1)^{th}$) adders. We have designed a 4-bit adder with 17 PEs shown in Fig. 2. Here, each PE requires the eight configuration bits to perform the operation shown in Fig. 1. Selecting input-1 and input-2 requires three bits for each and a total of six bits for input selection. The two bits are required for function selection. For a 4-bit adder, PEs were arranged in 2×9 matrix format. A total of 136 configuration bits were needed to implement the 4-bit adder. In the $M \times N$ arrangement of PEs, 'M' would remain the same for any adder length (where M is row), and 'N' would increase its scalability based on adder length (where N is column). This matrix format will be simple to access and process the result without any ambiguity.

Table 1 Input based MUX size for various adder lengths

Adder Bit Length	*MUX Size*
4-bit adder	5 : 1
8-bit adder	9 : 1
16-bit adder	17 : 1
32-bit adder	33 : 1

The same as the 4-bit VRC adder, the 8-bit, 16-bit, and 32-bit VRC adders were designed using PEs and MUX sizes for inputs, selection lines, and configuration bits. As an adder's input size increases, the size of the input-based MUX will also increase, as shown in Table 1. Similar to the previous statement based on the input size of an adder, the count of PEs, selection lines, and configuration bit-length will increase in 8-bit, 16-bit, and 32-bit VRC adder as shown in Tables 2, 3 and 4 respectively. The configuration bitstream of VRC adder is calculated based on the count of PEs and selection lines as shown in Eq. (1), where x = number of PEs for half-adder, y = number of PEs for full-adder, m = the number of columns of full adder and s = number selection lines in each PE. The 4-bit and 8-bit VRC adders were considered for experimental and testing of intrinsic EHW fault-tolerant adders.

$$Configuration \;\; bitlength = (x + (y \times m)) \times s \tag{1}$$

In existing works, the matrix design of the VRC adder was expensive and more challenging to implement. Sekanina and Friedl [18] designed a programmable element with 11-bit.

Fig. 2 4-bit VRC Adder

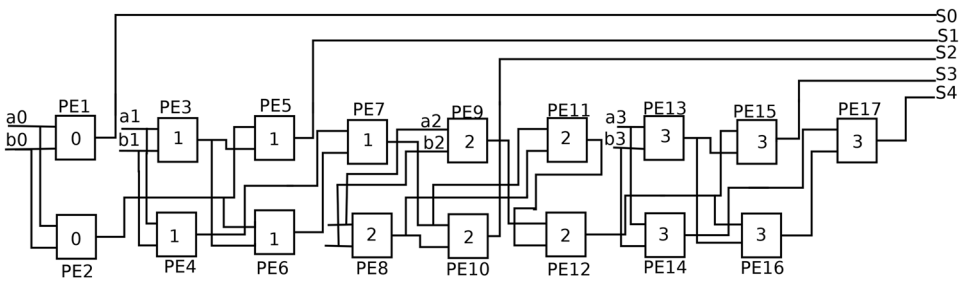

Table 2 # PEs for various bit length adders

Adder Bit Length	0^{th} *Bit Adder*	1^{st} *to* $(N-1)^{th}$ *Bit Adders*	*Total*
4-bit adder	2	15	17
8-bit adder	2	35	37
16-bit adder	2	75	77
32-bit adder	2	155	157

The PEs were arranged in a 10×8 matrix for a 3-bit adder. A total of 880 bits were utilized in constructing the 3-bit adder. Jian and Mengfei [7] designed a 2-bit adder in the format of neural network architecture. This looks fine for representation, but the implementation might be risky. A total of 176 bits were required to construct a 2-bit adder with neural network architecture. Many evolvable hardware adders were designed with direct bitstream manipulation using dynamic partial reconfiguration. This alternative implementation of VRC. This implementation was done with SRAM-based Xilinx FPGAs using ICAP or Jbits controller. Microsemi FPGAs were designed with AES encryption for encrypting the bitstream used for military and space applications. These FPGAs will have challenges in manipulating the bitstream directly with ICAP/JTAG controller during evolution. To avoid these issues, VRC is the best option to use. Although a solution is available, scalability issues were faced in existing works. But the proposed VRC adder has resolved the scalability issue at maximum. This adder can be generalized and used to maximum bit length based on the resources in the FPGA.

4 Proposed Intrinsic based Self-healing Adder

Evolvable hardware is hardware that can modify its behavior and architecture autonomously and dynamically according to its environment. In the early days of EHW design, EA was used for optimizing the circuit. Later, researchers tried to design fault-tolerant hardware with an evolutionary algorithm. The evolutionary algorithm has the capability of self-healing and self-adaptability. It will recover the faults in the circuit using a reconfigurable device. Hence, EHW combines a reconfigurable platform and an evolutionary algorithm. In EHW, the configuration bitstream is named as a chromosome. The existing work used reference circuits to calculate fitness in the evolution process during error recovery. This may require lots of memory space to store configuration memory [31]. Moreover, a long-length circuit could not be implemented due to memory space. This could be one of the limitations, as previously proposed adders with short bit lengths had been designed in this manner.

Genetic algorithm is one of the most used evolutionary algorithms in EHW. In the genetic algorithm, the primary issue is the number of iterations. This has variant operators, such as population generation, fitness calculation, selection operation, and crossover. Suppose we have an n-bit chromosome; we should generate a 2^n population. For example, the 4-bit adder requires a 136-bit chromosome for implementation and requires 2^{136} populations in the initial iteration to perform further operations. Then it will be memory overhead to store 2^{136} populations. The maximum number of iterations will be required until the fitness function is validated. These are significant complications in the genetic algorithm. But the novel chromosome reconstruction algorithm is feasible to implement and execute. The chromosome will be reconstructed based on information such as the number of programmable elements in half and full adders, the number of selection lines, the size of the adder, and the results of VRC construction and implementation.

The design of the proposed adder circuit is based on human-inspired techniques. This algorithm has various stages for recovering a fault in a circuit, as shown in Fig. 3. In the initial stage, the designed VRC adder circuit will be operated based on the chromosome along with inputs given by the user. The second stage is evaluation. In this stage, the resultant of the adder circuit is verified and validated with the help of the result generated by the reference output generator. The reference output generator is pre-executed before the first stage has been started. The absence of 1's bit in the evaluation result will indicate that no error has occurred; otherwise, it will indicate that there has been an error. If an error is absent, it will give back the result and terminates the execution of the adder. Else, the process will be continued to recover the error result. It follows the third stage, i.e., the chromosome reconstruction algorithm. This algorithm will restore the original chromosome of the adder in linear computational complexity. The proposed self-healing adder architecture is depicted in Fig. 3. The self-healing adder was implemented using an intrinsic approach. More details of the proposed work will be explained in further sections.

Table 3 Selection lines for each PE in various adders

Adder Bit Length	*Selection lines* (*s*)
4-bit adder	8
8-bit adder	10
16-bit adder	12
32-bit adder	14

Table 4 Configuration bit length for each adder

Adder Bit Length	*Configuration bitlength*
4-bit adder	136 bits
8-bit adder	370 bits
16-bit adder	924 bits
32-bit adder	2198 bits

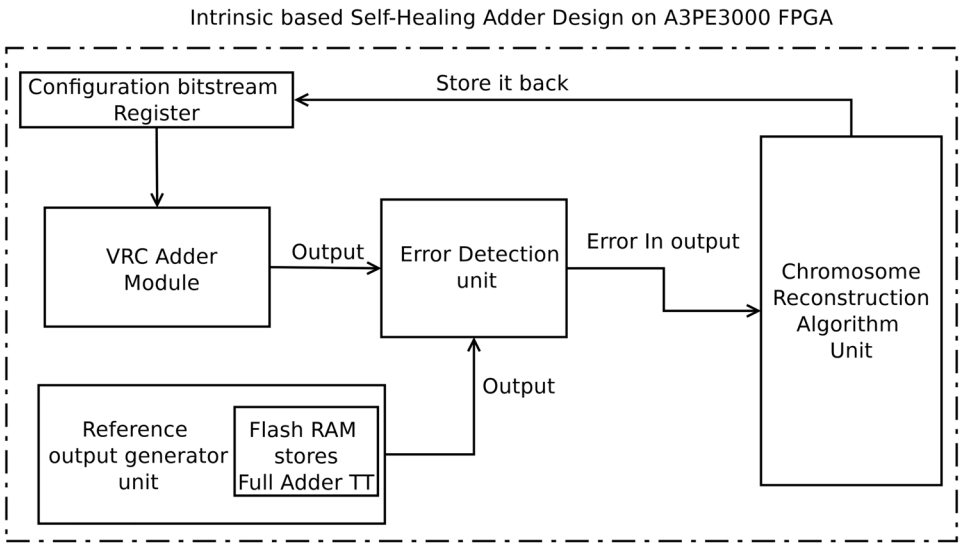

Fig. 3 Proposed Intrinsic based Self-Healing Adder Design

4.1 Reference Output Generator (ROG)

The theme of the fault-tolerant EHW adder is to execute the operation with a self-repair mechanism for generating accurate output. This process requires a fault-detection mechanism to locate the fault in a circuit before processing the self-repair mechanism. The fault detection unit requires a reference output to compare with the targeted circuit output. So, we designed a reference output generator (ROG) module for generating a reference output. In this module, the full-adder truth table has been used to generate the output of the adder circuit. The process of output generation is designed with an algorithm using logical AND. The truth table is stored in FlashROM of the FPGA fabric to avoid distractions. The main advantage of this approach is constant memory occupancy. Even as the size of the adder grows, the size of the ROG module remains constant. It is used for one-time output generation. The output will be generated at an initial stage and stored permanently. It will be operated before the VRC adder execution.

Algorithm 1 An algorithm for ROG

Require: *Given input to the ROG*
Ensure: *returns Sum*
1: x=input1
2: y=input2
3: $z[0] = 0$
4: Truth table of full-adder stored in FlashROM
5: **for** i=0 to n **do**
6: **for** j=0 to 8 **do**
7: **if** $x[i] == a[j] \&\& y[i] == b[j] \&\& z[i] == c[i]$ **then**
8: $su[i] = sum[j]$;
9: $z[i-1] = carry[j]$;
10: **end if**
11: $su[i+1] = z[i]$;
12: $Sum = su$
13: return Sum;

4.2 Fault Detection Mechanism

The output of the VRC adder is evaluated to analyze its accuracy and circuit efficiency. Hence, an error detection unit is designed with an XOR operation for evaluation. This unit will accept the outputs of the VRC adder and ROG as inputs. Later, it compares both the outputs with an XOR operation. The VRC adder generates an accurate result when the result of the fault detection operation is equal to all zero bits. If the function generates an output containing "1" bits, there might be a fault in the circuit. Then the circuit requires a self-repair facility. Later, it initiates the RC Algorithm unit to recover faults in a circuit. This algorithm will be helpful in reducing the execution time and increasing the system's performance.

4.3 Fault Recovery Mechanism

We are developing a novel algorithm for reconstructing the chromosome of the VRC adder. The proposed work avoids the first limitation of existing works by not storing a reference chromosome in memory. The chromosome is generated by the information of the VRC adder, such as the total configuration bitstream, total number of PEs, and number of PEs for the adder using this algorithm. This information will allow us to set the original chromosome of a circuit. The chromosome for each PE will be generated using simple mathematical calculations and programming techniques; it is induced in the proposed algorithm. This will reduce the recovery time. The proposed work avoids the first limitation of existing works by not storing a reference chromosome in memory. In this work, the chromosomes of functional elements and the N^{th} wire of inputs will remain the same in each bit adder, as mentioned in Tables 5 and 6.

Table 5 Chromosome of functional elements (FEs) and N^{th} wire of input1 and input2

S.No	Wires	Chromosome of N^{th} wire in various length adders			
		4-bit	8-bit	16-bit	32-bit
1	PE3	100	1000	10000	100000
2	PE4	100	1000	10000	100000
3	PE5	100	1000	10000	100000
S.No	FEs	Chromosome			
1	AND	00			
2	XOR	01			
3	OR	10			

In the error recovery process, the reconstructed chromosome is transferred to the configuration memory of the adder module to recover a fault in a circuit. Again, based on the new chromosome, the VRC adder will be processed and reperforms the operation to mitigate a fault in a circuit and generate a result. The result will be tested in an error detection unit. This evolution will be processed until the circuit generates an error-free result. We looked into many existing research studies that used the genetic algorithm for new chromosomes and noticed that they required the longest evolution time. This is another limitation of evolvable hardware. For the proposed approach, a genetic algorithm isn't needed. This limitation could be overcome with the proposed algorithm. The proposed algorithm is named the chromosome reconstruction (RC) algorithm.

4.3.1 Chromosome Reconstruction Algorithm

The chromosome reconstruction algorithm will work based on the blueprint of the VRC adder. Initially, information about the configuration bitstream length and the total number of PEs of the VRC adder was required. This information will be retrieved based on the number of PEs used in the full-adder and half-adder and the size of the adder. At the time of VRC design, the number of PEs for the full-adder and half-adder was finalized. The adder size can be identified from the input size. The total number of PEs and configuration bitstream length will be calculated using Eqs. (2) and (3), respectively. The algorithm's flow was started as a result of this information. In the algorithm, the configuration bitstream is stated as the chromosome. Allocating the required memory after finalizing the chromosome length of the complete adder in the algorithm. The allocated memory will be partitioned based on the number of PEs. Now, the chromosome bits for each PE will be generated and stored in the allocated memory. In this algorithm, the chromosomes will be rebuilt in the bottom-up approach.

Table 6 Mapping FEs with PEs

S.No	PEs	FEs	Chromosome
1	PE1	XOR	01
2	PE2	AND	00
3	PE3	XOR	01
4	PE4	AND	00
5	PE5	OR	10

In the VRC adder, the zeroth bit adder has two PEs, and the remaining bit adders from the first bit to the N-1 bit will have five PEs. We have information that each PE has three parts: input 1, input 2, and function. To operate the adder, these are accessed through selection lines. In the early stages of VRC construction, it was determined that each PE required eight chromosome bits for selecting the inputs and functions. Among these eight bits, six are used for accessing inputs 1 and 2 (three bits for each input), and two are used for accessing the function. As shown in Table 5, the two-bit chromosome was also finalized during VRC adder construction for accessing the function. The three-bit chromosome will access each input. The chromosome bits for input1 and input2 of PE1 and PE2 will be generated based on their adder positions, as described in Section 4.3.2. The chromosome bits for input1 and input2 for PE3, PE4, and PE5 will remain static. So, these bits were determined at the time of the VRC adder design shown in Table 5. Later, based on the blueprint of the VRC adder structure, the chromosome bits of each PE of every adder will be arranged and stored in allocated memory in the algorithm. This algorithm is described in HDL.

4.3.2 Procedure for Chromosome Reconstruction Algorithm

- Initially calculated number of PEs that required for adder using following equation

$$PEs = PEs\ in\ HA\ +\ ((PEs\ in\ FA) \times (size\ of\ adder - 1)) \quad (2)$$

- Calculated the chromosome length of complete adder.

$$Z = PEs \times sel\_lines \quad (3)$$

- Assign the chromosome bit length for each PE based on selection lines information.
- Assign the chromosome bit length for each bit adder based on the number of PEs in each full adder and half-adder.
- In each PE chromosome, last two LSBs related to functional element(n) and remaining bits are related to inputs(m).

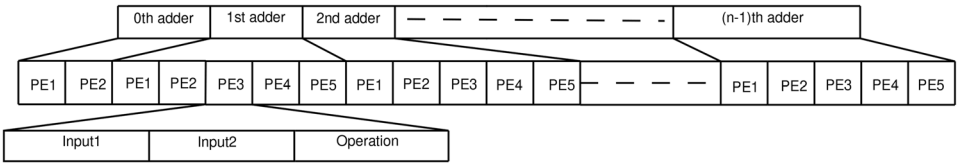

Fig. 4 Representation of VRC Adder Chromosome

- Remaining chromosome length related to inputs divided by two. Hence, provides the chromosome length for each input.
- In each bit adder, the PE1 and PE2 input wire chromosomes will be generated based on the bit position of adder.
- For example, the zero wire will be selected in zeroth bit adder. Then chromosome for input1 is 000 and chromosome for input2 is 000 (for 4-bit adder).
- Based on chromosome length of input1 and input2, could be able to retrive the appropriate chromosome for *N th* wire inputs of PE3, PE4 and PE5.
- Also set the functional elements(FEs) chromosome of each PE based on pattern 01, 00 and 01, 00, 01, 00, 10 for half-adder and full-adder respectively.
- Now combine the chromosome bits of PEs and set for each bit adder with help of the bit position of each adder from 0 to $N-1$, shown in Fig. 4.
- The complete process has been structured in the HDL according to logic.

The proposed algorithm is generic to reconstruct the chromosome of a VRC adder for any input length. Its simple and easy to implement without hardware overhead and time delay. The above procedure will be explained with 4-bit adder in the following section.

4.3.3 Example for Chromosome Reconstruction Algorithm

After identification of the error in the result of the 4-bit adder, we can restore the chromosome of the 4-bit adder using a chromosome reconstruction algorithm as below.

1. calculated number of PEs that required for adder using following equation $PEs = 2 + ((5 \times (4-1)) = 17$
2. Calculated the chromosome length of complete adder. $z = 17 \times 8 = 136$ bits
3. Assign 8-bits for PE of adder based on selection lines information.
4. Assign number of chromosome length for each adder based on count of PEs in full-adder and half-adder and with bits for each PE.
   - PEs in full-adder,m = 5 and PEs in half-adder,n = 2
   - Chromosome length of full-adder, k = $m \times z = 5 \times 8$ = 40 bits
   - Chromosome length of half-adder, l= $n \times z = 2 \times 8$= 16 bits
5. In each PE chromosome, the last two LSBs (i= 2 bits) are assigned to adder functions.
6. Remaining chromosome bit length (j) are assigned to two inputs. For each input, the chromosome bit length are assigned by dividing with 2.
   - $j = z - i = 8 - 2 = 6$-bits
   - $p = j/2 = 6/2 = 3$-bits
   - Then assigned chromosome length $input\ 1 = 3 - bits$ and $input\ 2 = 3 - bits$
   - Generate the chromosome bits for input1 and input2 of PE1 and PE2 (shown in Table 7) based on 'p' and bit-position of adder.
7. Retrive the chromosome information for input1 and input2 of PE3, PE4, and PE5, and also logical functions chromosome for each PE from Flash ROM memory based on size of adder (shown in Table 8).
8. Now reconstruct the chromosome according to representation shown in Fig. 4. This chromosome representation has been derived from VRC adder structure.
9. The reconstructed chromosome (shown in Table 9) will be transferred to configuration bitsream memory.

Table 7 Chromosome for Input1 and Input2 for PE1 and PE2

S. No	Bit position of adder	Inputs PE1 and PE2	
		Input 1	Input 2
1	Zeroth	000	000
2	First	001	001
3	Second	010	010
4	Third	011	011

Table 8 Chromosome of full-adder for 4-bit Adder

S. No	PE Number	Chromosomes		
		Input 1	Input 2	Function
1	PE1	XXX	XXX	00
2	PE2	XXX	XXX	01
3	PE3	100	100	00
4	PE4	100	100	01
5	PE5	100	100	10

Table 9 Chromosome of 4-bit Adder

zeroth adder						First adder														
PE1			PE2			PE1			PE2			PE3			PE4			PE5		
000	000	01	000	000	00	001	001	01	001	001	00	100	100	01	100	100	00	100	100	10
						Second adder														
						PE1			PE2			PE3			PE4			PE5		
						010	010	01	010	010	00	100	100	01	100	100	00	100	100	10
						Third adder														
						PE1			PE2			PE3			PE4			PE5		
						011	011	01	011	011	00	100	100	01	100	100	00	100	100	10

5 Experimental Setup and Results

5.1 Experimental Environment

The experimental work was carried out with the 4-bit and 8-bit adders to test the proposed work efficiency. The fault-tolerant evolvable hardware adder prototype and algorithm hardware testing were hosted on the Proasic3e 3000 FPGA. This FPGA is encrypted with the AES algorithm. Reading and writing the bitstream of a circuit is complicated compared to Xilinx-based FPGAs. It is a challenging task to implement a fault-tolerant combinational unit on the Proasic3e 3000 FPGA. So, we designed the optimized VRC adder to overcome the significant complication. The adder circuit evolved through the reconstruction chromosome algorithm. This algorithm is compensated with a genetic algorithm to reduce the recovery time during the recovery process of the original chromosome. Most existing works were created using an extrinsic or hybrid approach. But, the proposed work has been implemented using intrinsic approach. The entire proposed self-healing hardware was deployed and executed on a single FPGA running at a frequency of 350 MHz. All four modules were designed in HDL using the Libero SOC design suite 11.8. This design suite is related to the Microsemi vendor, which uses it to develop the HDL models for their FPGAs. This suite is an integrated FPGA design tool that incorporates a modelsim simulation tool, a synopsys synthesis tool, and a programming debug tool.

In the Proasic3e 3000, the FPGA has flash ROM memory. It is one of the advantages of storing the essential information. The FPGA chip contains the full adder truth table for reference output generation, a chromosome of functional information elements, PEs information for full-adder and half-adder, several columns for full-adder, and the number of selection lines of a MUX. This is a secure memory that offers programmers the ability to read, modify, and write the content using the JTAG interface. This is one of the best features compared to other SRAM-based FPGAs. The primary motivation for the proposed work is to solve the scalability issues by optimizing the VRC adder design and to improve the fault-recovery time with a novel human-inspired algorithm, i.e., chromosome reconstruction. Here, the circuit evolution process continues until the error detection notifies the no error with a complete all-zero bit. The proposed work has been tested by fault injection to analyze the efficiency of the work. Hence, the fault-injection simulator was implemented for injecting faults in a VRC adder in two locations, i.e., at input routing and at functional MUX.

5.2 Experimental Results

The proposed algorithm required around eight milliseconds to reconstruct the chromosome of Adder. It takes very little time compared to the genetic algorithm. In one of the existing works, it takes around 2.5 seconds to recover 90% matched chromosome compared to its original [29]. In a best-case scenario, the algorithm unit may be used once to evolve a circuit to get a fault-free result. In the worst case, the algorithm unit may be executed N times until the error detection unit results in a zero error. The best and worst-case scenarios depend on the working conditions and usage of the device. But during physical error injection, the algorithm unit was used once, and at the first evolution of the circuit, the error detection validated the result with zero errors. The execution time of the complete self-healing adder circuit will be the sum of the execution times of a VRC adder, an error detection unit, a reference output generator unit, and a chromosome reconstruction unit. This is the execution time for an initial error occurrence. If the error occurs again for the second time or N times, exclude the execution time of a reference output generator unit. When the circuit is initially tested without error, calculate the execution time based on the execution times of the VRC adder, an error detection unit, and a reference output generator unit. The proposed work for 4-bit and 8-bit adders takes around 12 milliseconds and 28 milliseconds to generate fault-free results, respectively. In Jian and Mengfei [7] work, the 2-bit adder took approximately 17 milliseconds of evolution time using the hybrid approach. We analyzed Jian's adder with 4-bit input, and it takes around 38 milliseconds to evolve a circuit. In Cancare et al.'s work [2], the evolution time for a 4-bit

Table 10 Comparing the evolution time of a proposed work with existing works

Works	Adder size	Reconfiguration mechanism	Evolvable approach	Evolution time (in seconds)
Existing works	4-bit [2]	DPR	Extrinsic	2040
	4-bit [7]	VRC	Extrinsic	0.038
Proposed work	4-bit	VRC	Intrinsic	0.012
	16-bit	VRC	Intrinsic	0.028

adder took approximately 34 minutes to get a fault-free result. However, it was implemented using the DPR mechanism using an extrinsic approach. The comparison of the evolution time of existing works and proposed work is shown in Tables 10 and 11. In contrast to previous work, the proposed work evolves the circuit more efficiently during fault occurrence.

The area occupancy of the proposed work is accounted for in terms of IO cells, core cells, and flashROM utilization. The power consumption is estimated by the LiberoSOC design suite tools. Therefore, the resource usage and recovery time of the proposed work are compared with similar works that were discussed in the related works. The proposed work has a core cell utilisation of 53.64%, which is lower than existing work for both size adders. In addition, IO cells and FlashROM were used 42.77% and 54.9% less, respectively, than in previous work. Hence, the resource utilisation of the proposed work is 50% lower than the existing work.

6 Discussion

The proposed fault-tolerant evolvable adder design is motivated by human-inspired algorithms [10]. Human-inspired algorithms are a subset of nature-inspired algorithms. These algorithms are designed based on human-related techniques. These techniques are related to non-physical activities, such as behavior and thinking, known as human activities.

The state-of-the-art works encounter challenges like scalability issues, evolution time, dependability, and redundancy. Scalability is a major issue in EHW with VRC mechanisms. The design space of the application was increased due to the usage of more functions in the VRC adder circuit [18]. Due to this, the configuration bit size will increase. Therefore, the configuration bit size is directly related to the circuit size. This affects the evolution time of a circuit. The adder's design would require more resources in the FPGA if its input size increased. If the same circuit is designed with VRC, then there may be insufficient resources in the FPGA during circuit deployment. The proposed VRC adder resolves this scalability issue by using the required functions of the operation. This will decrease the size of the MUX. As the size of the MUX is optimized for function selection, the configuration bitstream length will also be reduced. This will reduce the evolution time of a circuit and improve the error recovery rate. In some works [7, 29], due to the larger size of the configuration bitstream resulted in a long evolution time and slow error recovery speed. The use of a genetic algorithm for recovering the configuration bitstream in previous works increased the search space due to the larger bitstream size. This led to slow error recovery speed. Using a genetic algorithm, it is not always possible to expect the desired outcome within the specified time frame.

The proposed chromosome reconstruction algorithm doesn't require functions like fitness, selection, mutation, and crossover for children's generations. Because these functions were unavailable, the search space was avoided in this algorithm. Hence, the error recovery rate of the circuit will be improved. The other point is that the execution time of an algorithm only depends on the size of the configuration bitstream. The optimized VRC adder has a smaller configuration bitstream compared to previous

Table 11 Hardware utilization of Proposed Work

Adder size	Resources	Available	Proposed Self-Healing Adder with Optimized VRC		Existing EHW Adder with standard VRC	
			Utilization	Utilization %	Utilization	Utilization %
4-bit	Core Cells	75264	15	0.0199	32	0.0425
	IO Cells	620	55	8.870	95	15.32
	FlashROM	1024	200	19.53	430	41.99
8-bit	Core Cells	75264	33	0.0438	72	0.0956
	IO Cells	620	112	18.06	198	31.94
	FlashROM	1024	400	39.06	840	82.03

work. The EHW was implemented using an extrinsic and hybrid approach in previous works [7, 9, 22]. It means that an evolutionary algorithm will be implemented on the external processor or processor on the SoC and evaluate the circuit's functionality in hardware, which can create an additional delay. The proposed work is completely designed on a single FPGA. Due to this intrinsic approach, implementation avoids dependability. It will reduce the delay during the operation as compared to previous work. Other works suggested that to avoid mission halts [7, 9], the EHW be implemented using the redundancy method at the FPGA board level, which meant dual boards were used to avoid the hardware overhead. The redundancy method at the board was not encouraged in the proposed work. The limitation of the EHW is that when algorithms are implemented on the same FPGA as the circuit, there might also be a probability of errors occurring in the algorithm. This may mislead the VRC adder circuit's operation. In future work, we will overcome this issue with an optimized redundancy approach in the algorithm.

7 Conclusion

The radiation environment or other external sources are the most common causes of SEU in a combinational circuit such as an adder. This could flip a single bit of the original content and result in a fault. To avoid this obstruction, there are many conventional fault-tolerant techniques and EHW approaches for combinational circuits. However, each of these has drawbacks such as area, delay, and error recovery time. The proposed work has overcome these drawbacks. Compared to previous works described in related works, the optimized VRC adder required only 136 configuration bits for 4-bit. This adder evolved using a novel algorithm named the chromosome reconstruction algorithm. This will generate a new chromosome (configuration bitstream) when an adder result notices an error. This algorithm is efficient for recovering the fault-free circuit and producing results within the short time mentioned in the results section. The complete fault-tolerant self-healing adder architecture required 50% less hardware than existing fault-tolerant hardware.

Acknowledgements This work was supported by DRDO/DFTM/05/3424/EMECS/001/M/01/RIC-35. Research and Innovation Centre, DRDO, Chennai.

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

Declarations

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

References

1. Almeida M, Pedrino EC (2018) Hybrid evolvable hardware for automatic generation of image filters. Integr Comput Aided Eng 25(3):289–303
2. Cancare F, Bartolini DB, Carminati M, Sciuto D, Santambrogio MD (2012) On the evolution of hardware circuits via reconfigurable architectures. ACM Trans Reconfigurable Technol Syst (TRETS) 5(4):1–22
3. Eiben AE, Smith JE et al (2003) Introduction to evolutionary computing 53
4. 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
5. Haddow PC, Tyrrell AM (2011) Challenges of evolvable hardware: past, present and the path to a promising future. Genet Program Evolvable Mach 12(3):183–215
6. Huang X, Wu N, Zhang X, Liu Y (2015) An evolutionary algorithm based on novel hybrid repair strategy for combinational logic circuits. IEICE Electron Exp pp. 12–20150765
7. Jian G, Mengfei Y (2018) Evolutionary fault tolerance method based on virtual reconfigurable circuit with neural network architecture. IEEE Trans Evol Comput 22(6):949–960. https://doi.org/10.1109/TEVC.2017.2779874
8. Jiang J, Yu X (2012) Fault-tolerant control systems: A comparative study between active and passive approaches. Annu Rev Control 36(1):60–72
9. Lohn J, Larchev G, DeMara R (2003) A genetic representation for evolutionary fault recovery in Virtex FPGAs. In: International conference on evolvable systems. Springer, pp 47–56
10. Mohamed AW, Hadi AA, Mohamed AK (2020) Gaining-sharing knowledge based algorithm for solving optimization problems: a novel nature-inspired algorithm. Int J Mach Learn Cybern 11(7):1501–1529
11. Mora J, Salvador R, de la Torre E (2019) On the scalability of evolvable hardware architectures: comparison of systolic array and cartesian genetic programming. Genet Program Evolvable Mach 20(2):155–186
12. Morgan KS, McMurtrey DL, Pratt BH, Wirthlin MJ (2007) A comparison of tmr with alternative fault-tolerant design techniques for FPGAs. IEEE Trans Nucl Sci 54(6):2065–2072
13. Oreifej RS, Al-Haddad RN, Tan H, DeMara RF (2007) Layered approach to intrinsic evolvable hardware using direct bitstream manipulation of Virtex II pro devices. In: 2007 International Conference on Field Programmable Logic and Applications, pp 299–304. https://doi.org/10.1109/FPL.2007.4380663
14. Salvador R (2016) Evolvable hardware in FPGAs: Embedded tutorial. In: 2016 International Conference on Design and Technology of Integrated Systems in Nanoscale Era (DTIS), IEEE pp 1–6
15. Salvador R, Otero A, Mora J, dela Torre E, Sekanina L, Riesgo T (2011) Fault tolerance analysis and self-healing strategy of autonomous, evolvable hardware systems. In: 2011 International Conference on Reconfigurable Computing and FPGAs, IEEE, pp 164–169
16. Sayil S (2019) A survey of circuit-level soft error mitigation methodologies. Analog Integr Circ Sig Process 99(1):63–70
17. Sekanina L (2003) Virtual reconfigurable circuits for real-world applications of evolvable hardware. In: International Conference on Evolvable Systems, Springer, pp. 186–197
18. Sekanina L, Friedl Š (2004) An evolvable combinational unit for FPGAs. Comput Inform 23(5–6):461–486
19. Sekanina L (2007) Evolutionary functional recovery in virtual reconfigurable circuits. ACM J Emerg Technol Comput Syst 3(2):8. https://doi.org/10.1145/1265949.1265954
20. Shang Q, Chen L, Wang D, Tong R, Peng P (2019) Evolvable hardware design of digital circuits based on adaptive genetic

algorithm. In: International Conference on Applications and Techniques in Cyber Security and Intelligence, Springer, pp 791–800
21. Silva BA, Dias MA, Silva JL, Osorio FS (2010) Genetic algorithms and artificial neural networks to combinational circuit generation on reconfigurable hardware. International Conference on Reconfigurable Computing and FPGAs. https://doi.org/10.1109/reconfig.2010.25
22. Silva GNP, Duarte RO (2018) Towards evolvable hardware and genetic algorithm operators to fail safe systems achievement. In: 2018 IEEE 19th Latin-American Test Symposium (LATS), pp 1–4. https://doi.org/10.1109/LATW.2018.8349669
23. Trefzer MA, Tyrrell AM (2015) Devices and architectures for evolutionary hardware. In: Evolvable Hardware, Springer, pp. 27–87
24. Vašíček Z, Žádník M, Sekanina L, Tobola J (2008) On evolutionary synthesis of linear transforms in FPGAa. In: International Conference on Evolvable Systems. Springer, pp 141–152
25. Vipin K, Fahmy SA (2018) FPGA dynamic and partial reconfiguration: A survey of architectures, methods, and applications. ACM Computing Surveys (CSUR) 51(4):1–39
26. VonNeumann J (2016) Probabilistic logics and the synthesis of reliable organisms from unreliable components. Automata studies 34
27. Wang F, Agrawal VD (2008) Single event upset: An embedded tutorial. In: 21st International Conference on VLSI Design (VLSID 2008), IEEE pp. 429–434
28. Wang J, Liu J, Feng B, Hou G (2015) The dynamic evaluation strategy for evolvable hardware 2015. 9th International Conference on Frontier of Computer Science and Technology. https://doi.org/10.1109/fcst.2015.35
29. Wang J, Liu J (2017) Fault-tolerant strategy for real-time system based on evolvable hardware. J Circuits Syst Comput 26(07):1750111
30. Yao R, Zhu P, Du J, Wang M, Zhou Z (2018) A general low-cost fast hybrid reconfiguration architecture for FPGA-based self-adaptive system. IEICE Trans Inform Syst 101(3):616–626
31. Yao X, Higuchi T (1999) Promises and challenges of evolvable hardware. IEEE Trans Syst Man Cybern Part C (Appl Rev) 29(1):87–97
32. Zhang Y, Jiang J (2008) Bibliographical review on reconfigurable fault-tolerant control systems. Annu Rev Control 32(2):229–252

Raghavendra Kumar Sakali has obtained his B.Tech. in Information Technology and M.Tech. in Computer Science and Engineering from Jawaharlal Nehru Technological University, Anantapur. He is currently pursuing PhD at the department of Computer Science and Engineering, Indian Institute of Information Technology Design and Manufacturing Kancheepuram, Chennai, India. His research interest is evolvable hardware and fault tolerant computing.

Noor Mahammad Shak has obtained his PhD from Indian Institute of Technology Madras, Chennai, India. 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, India. His research interest are evolvable hardware and reconfigurable computing.