Evolutionary Functional Recovery in Virtual Reconfigurable Circuits[1]

LUKÁŠ SEKANINA
Brno University of Technology

A virtual reconfigurable circuit (VRC) is a domain-specific reconfigurable device developed using an ordinary FPGA in order to easily implement evolvable hardware applications. While a fast partial run-time reconfiguration and application-specific programmable elements represent the main advantages of VRC, the main disadvantage of the VRC is the area consumed. This study describes experiments conducted to estimate how the use of VRC influences the dependability of FPGA-based evolvable systems. It is shown that these systems are not as sensitive to faults as their area-demanding implementations might suggest. An evolutionary algorithm is utilized to design fault tolerant circuits as well as to perform an automatic functional recovery when faults are detected in the configuration memory of the FPGA. All the experiments are performed on models of reconfigurable devices.

Categories and Subject Descriptors: B.6.3 [**Hardware**]: Logic Design—*Design Aids*; B.8.1 [**Hardware**]: Performance and Reliability—*Reliability, Testing, and Fault-Tolerance*

General Terms: Design

Additional Key Words and Phrases: dependability, evolutionary algorithms, evolvable hardware, FPGA

1. INTRODUCTION

Evolvable hardware is an approach in which a physical hardware is created and adapted using an evolutionary algorithm (EA) [Higuchi et al. 1993; Thompson 1998]. Typically, a configuration bitstream of a reconfigurable device is encoded in the chromosome of the evolutionary algorithm. The evolutionary algorithm is used to find a configuration bitstream which satisfies the requirements on the circuit behavior that are formulated in the fitness function. The fitness function can include behavioral as well as non-behavioral requirements (e.g. functionality vs. circuit size). It has been demonstrated in the literature that the evolutionary approach is sometimes able to produce the results that are better than the best conventional solutions known so far [Miller et al. 2000; Koza et al. 1999; Thompson 1998; Sekanina 2004]. It is also able to produce a (suboptimal) solution in the case that no conventional solution exists, the reconfigurable circuit is partially damaged

[1]THIS IS DRAFT, (C) ACM

Author's address: L. Sekanina, Faculty of Information Technology, Brno University of Technology, Božetěchova 2, 612 66 Brno, Czech Republic, e-mail: sekanina@fit.vutbr.cz

or only a limited time or space is available for the design [Stoica et al. 2001].

Field programmable gate arrays (FPGAs) containing an array of programmable elements, programmable interconnects and programmable ports have become the most popular platform for evolvable hardware. However, it is a well-known problem that the reconfiguration subsystem of current field programmable gate arrays is not well-suited for evolvable hardware. In evolvable hardware, we often require the possibility of a fast dynamic partial reconfiguration, including the controllable granularity of programmable elements and a transparent structure of the configuration bitstream. Ideally, it would be nice to have a specialized reconfigurable device for a given application in order to meet its particular requirements. Unfortunately, developing an application specific integrated circuit (ASIC) is sometimes impossible for many (mainly economic) reasons.

In order to avoid using the reconfiguration subsystem of FPGAs, the concept of *virtual reconfigurable circuits* was introduced [Sekanina 2003]. A virtual reconfigurable circuit (VRC) is, in fact, an implementation of a domain-specific reconfigurable circuit on top of an ordinary FPGA. Because a designer can construct the VRC so that it exactly fits the needs of a given evolvable hardware-based application, a perfect reconfigurable device can be obtained for a given problem. Examples include VRCs for the evolution of logic circuits [Sekanina and Friedl 2004], image filters [Martinek and Sekanina 2005], IIR filters [Gwaltney and Dutton 2005], sorting networks [Korenek and Sekanina 2005] or packet classifiers [Salomon et al. 2006]. As the evolutionary algorithm can be implemented on the same FPGA (as an application-specific circuit or in a processor) in these applications, a fast configuration interface can be established connecting the configuration memory of VRC with chromosomes of EA.

On the other hand, implementations of VRCs are relatively expensive in terms of gates used because the interconnection circuits of VRCs are based on area-expensive multiplexers.

Since VRCs are available at the level of HDL source code, i.e. totally independent of a target platform, they can be utilized (in connection with a hardware implementation of the evolutionary algorithm) to implement soft evolvable IP cores. It is postulated that evolvable hardware at the level of IP cores will become an integrated component in a variety of systems in the future [Stoica et al. 2000]. These evolvable systems will be responsible for completing tasks that are difficult for conventional hardware solutions, for instance, adaptation of functionality, adaptation of sensing, autonomous self-repairing and learning. There is an increasing interest in the use of evolvable hardware in space applications, for example, in extreme environments exhibiting radiation and temperature levels different from Earth surface [Stoica et al. 2001].

In particular, in this study, we will investigate the influence of faults in the configuration memory of VRCs and FPGAs. The objective of this research is to show how the use of (area-demanding) VRCs could influence the dependability of evolvable systems. Because VRCs require more resources than other common approaches used to implement a given function in an FPGA, it is realistic to suppose that their use will yield less reliable solutions. For instance, consider a 1bit full adder. Its implementation costs a few equivalent gates on an FPGA. However,

several hundred gates might be activated if a VRC is utilized. The pessimistic scenario says that the reliability will be decreased one hundred times in the case of the use of the VRC. On the other hand, we can operate with the optimistic scenario, in which the inherent redundancy of the VRC implementation is sometimes useful and can protect the circuits from faults. We will show on simplified models of FPGA and VRC that the optimistic scenario can partially be taken into account. In our previous work [Sekanina 2006], evolutionary techniques were utilized for the design of fault tolerant circuits in the VRC. This paper mainly deals with the evolutionary functional recovery of circuits in VRCs.

In order to perform the experimental analysis of these behaviors, the FPGA, the VRC implemented on the FPGA and a special environment testbed are needed. Unfortunately, this equipment is relatively expensive. Hence we have decided to perform all experiments using simulators before physical devices are utilized. We do believe that the results obtained will give at least a partial image of the problem.

In order to perform the experiments, we have to implement the following programs: a simulator of a simple SRAM-based reconfigurable device, an implementation of VRC using the simulator of the SRAM-based device, a fault generator and an evolutionary algorithm for repairing the circuits and designing fault tolerant circuits. The approach will be validated on two circuits—one-bit full adder (1bFA) and two-bit multiplier (2bMU)—these are both popular problems in the evolvable hardware community (for example, see [Garvie and Thompson 2003; Miller et al. 2000]). After the analysis of fault tolerance for conventional implementations of test circuits, we will perform an evolutionary design of test circuits and an evolutionary design of fault tolerant circuits in the VRC. Finally, we will investigate various approaches to the evolutionary functional recovery in the VRC.

The rest of this paper is organized as follows. Section 2 summarises previous relevant research. The proposed models are introduced in Section 3. Section 4 describes experiments performed with evolutionary design of fault tolerant circuits in the VRC. Section 5 is devoted to the evolutionary functional recovery. Conclusions are given in Section 6.

2. RECONFIGURABLE DEVICES AND FAULT TOLERANCE

2.1 Partial Reconfiguration in FPGAs

As Figure 1 shows, in the case of evolvable hardware, the chromosomes are transformed into the configuration bitstreams and the configuration bitstreams are uploaded into the FPGA. A partial runtime reconfiguration allowing the dynamic reconfiguration of a selected area of the FPGA is very important to accomplish this task effectively. The possibility of a dynamic partial reconfiguration is fully supported by Atmel FPGAs (in particular by Atmel AT94K FPSLIC device); however, these Atmel FPGAs contain only a small matrix of programmable elements.

Complex Xilinx Virtex FPGAs are the most popular reconfigurable devices among the evolvable hardware community. Xilinx FPGAs support a partial reconfiguration. From the configuration perspective, a Virtex FPGA is divided into configuration frames. A single frame represents a part of the configuration information in a single column of programmable elements. In order to reconfigure a single programmable element, 48 frames have to be uploaded. The size of a frame is 832

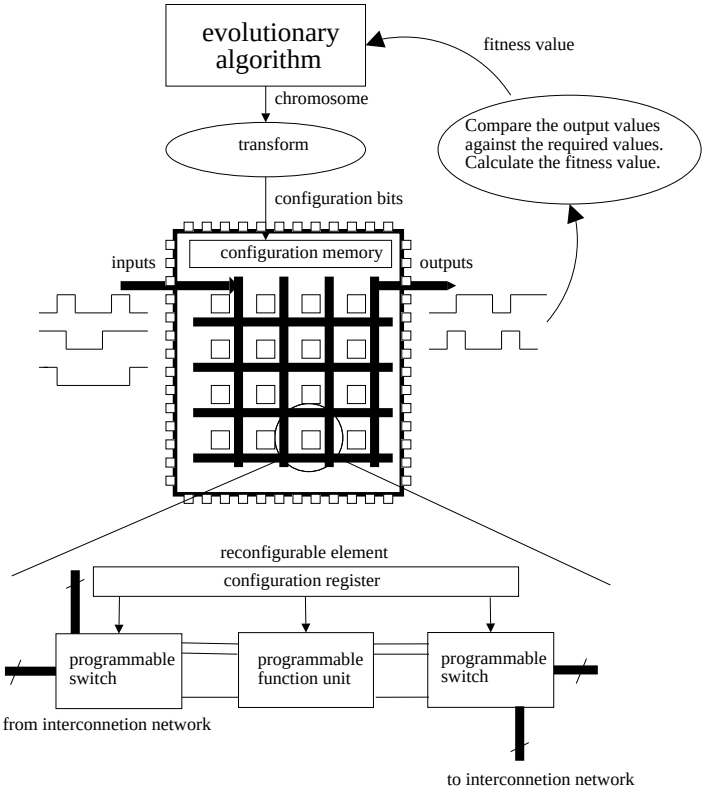

Fig. 1. Evolvable hardware: Candidate configurations are generated by the evolutionary algorithm, uploaded to a reconfigurable device and evaluated using fitness function

bits for the smallest Virtex FPGA. Although the internal configuration access port (ICAP) has been integrated into Xilinx Virtex II – Virtex 5 families, it can still be slow for evolvable hardware because this port works at the level of frames and operates at max. 66 MHz (for 8bit data). Furthermore, details of configuration bitstream structure are not available in the documentation of Virtex FPGAs. This makes a direct evolutionary design at the level of the Virtex configuration bitstream very difficult.

2.2 Virtual Reconfigurable Circuits

Virtual reconfigurable circuits were introduced for digital evolvable hardware as a new kind of reconfigurable platform utilizing conventional FPGAs [Sekanina 2004; 2003]. When a VRC is uploaded into the FPGA then the following units will be created at specified positions: an array of programmable elements (PE), a programmable interconnection network, a configuration memory (implemented as a register array) and a configuration port. The VRC is created in the process of FPGA configuration at the beginning of computation; i.e. a partial reconfiguration does not have to be supported in FPGA at all.

Fig. 2 shows that the VRC is, in fact, a new reconfigurable circuit (consisting

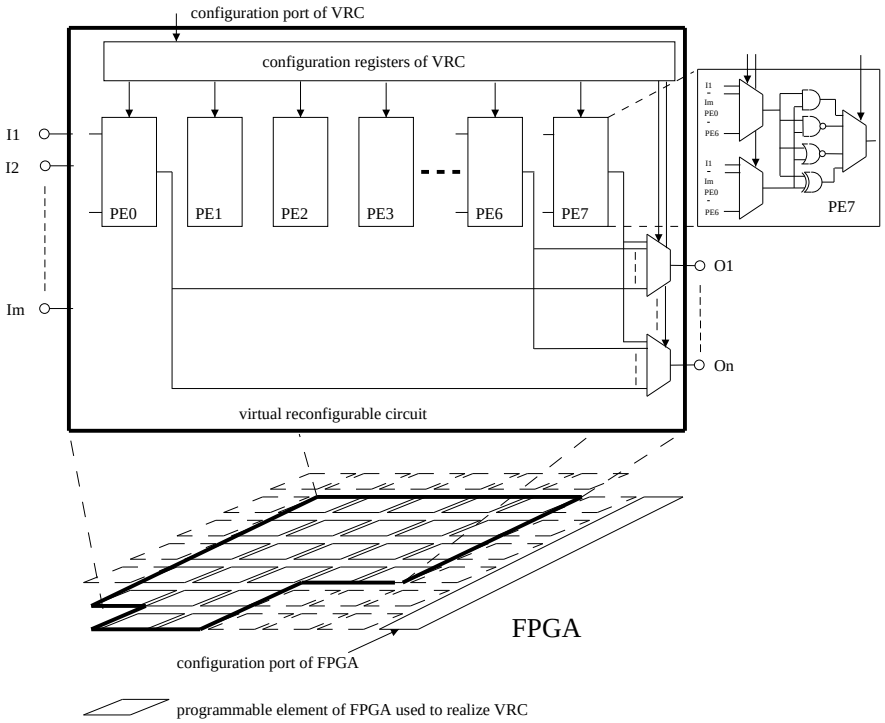

Fig. 2. Basic idea of the VRC. The VRC is described in HDL, synthesized and uploaded as a configuration bitstream to the FPGA.

of 8 programmable elements in our example) implemented on top of an ordinary FPGA. The "Virtual" PE7 depicted in detail in Fig. 2 is controlled using configuration bits that determine the selection of its operands and internal function. This architecture is very similar to the representation employed in Cartesian Genetic Programming (CGP) that has been developed for circuit evolution [Miller et al. 2000]. Routing circuits are implemented using multiplexers. All the bits of the configuration memory are connected to multiplexers that control routing and the selection of functions implemented in PEs.

The main advantage of the proposed method is that the array of PEs, routing circuits and the configuration memory can be designed exactly according to the requirements of a given application. Furthermore, the style of reconfiguration and granularity of the new VRC can exactly fit the needs of a given application. Because VRCs can be described in HDLs, they can be synthesized using common synthesis tools, i.e. with various constraints and for various target platforms.

The following list shows examples of evolvable systems which were implemented using the idea of the VRC on an FPGA:

—Evolvable image filter for the evolution of 3x3 image operators (EA is implemented either in PC [Zhang et al. 2004] or on the same FPGA as a special circuit [Martinek and Sekanina 2005] or in an on-chip PowerPC processor [Vasicek and

Sekanina 2007]).

—Evolvable sorting network for up to 28 inputs (EA is implemented on the same FPGA) [Korenek and Sekanina 2005].

—Evolvable combinational circuits (EA is implemented on the same FPGA as a special circuit [Sekanina and Friedl 2004] or in the PowerPC processor on the same FPGA [Glette and Torresen 2005]).

—Intrinsic evolution of polymorphic combinational modules (EA is implemented on the same FPGA) [Sekanina et al. 2006].

—Evolvable IIR filter (EA is implemented on a DSP) [Gwaltney and Dutton 2005].

—Packet classifiers (EA is implemented as a special circuit on the same FPGA) [Salomon et al. 2006].

2.3 Fault Tolerance

There is an increasing interest in the use of SRAM-based FPGAs in space applications that operate in extreme environments. Challenges for evolvable hardware are to (1) provide fault tolerant designs automatically and to (2) ensure autonomous function recovery for these devices after an occurrence of unavoidable damages caused by extreme radiation, temperature or simple malfunctions (e.g. in a presence of stuck-at-zero faults, severe electric transients, etc.). Because the probability of damage in a system increases with the number of components involved, an increasing complexity of the circuits implemented in FPGAs evokes a question of how can they be made fault tolerant?

Since SRAMs are very sensitive to radiation, faults in the configuration memory of FPGAs are very common in these environments. Space radiation has both long-term and single particle effects on electronic components. Long-term effects include total ionizing dose. Single-event effects include single-event latchup and single-event upset [Wirthlin et al. 2003; Bernardi et al. 2004; Alderighi et al. 2005]. Xilinx has offered the QPRO Virtex FPGA, which is immune to latch-up, and has an acceptable total-dose tolerance [qpro 2001]. However, it is sensitive to single-event upsets, i.e. the changes in states of a digital memory element caused by an ionizing particle that can change the functionality of the device. These single-event upsets are soft errors that do not usually cause any permanent damage to an FPGA. Wirthlin et al. [2003] give an example, in which memory cells are anticipated to upset at a rate of 3.2 upsets/day. These upsets can usually be fixed by a readback and fast partial reconfiguration; however, some single-event upsets require a total reconfiguration. As the dose increases, the FPGA may fail to power-up.

Several techniques have been developed and tested to improve fault tolerance in FPGAs (including restrictions of single-event upsets, e.g. [Carmichael et al. 2000; Bernardi et al. 2004]). All these approaches are based on time or space redundancy [Pradhan 1996]. In case of reconfigurable devices, we are looking for effective methods capable of recovering the functionality by means of a smart reconfiguration strategy. Novel approaches have been developed in recent years in the fields of embryonics and immunotronics. Embryonics utilizes the principles of cellular division and differentiation observed in multicellular organisms [Mange et al. 2000]. In immunotronics, artificial immune systems are built in order to protect computing devices [Bradley et al. 2000]. Evolvable hardware offers other options.

2.4 Evolvable Hardware and Fault Tolerance

Various evolutionary approaches to fault tolerance are being investigated in the evolvable hardware field. In the case of *explicit* fault tolerance, the requirements for fault tolerance are included into the fitness function. Then, in addition to testing of functionality, fault tolerance is also evaluated in the fitness function.

Due to the existence of redundant elements in reconfigurable devices, evolvable hardware is *inherently* fault tolerant [Thompson 1998; Tyrrell et al. 2001]. It means that in case of a failure of a circuit element, the evolutionary algorithm is usually able to recover the original functionality using the remaining elements or using another member of the population which might be insensitive to the failure [Tyrrell et al. 2004]. If a critical number of elements is damaged, the functionality cannot be recovered and the chip "dies". Another method is to evolve circuits that are able to detect malfunctions autonomously or that are easily testable. These experiments have initially been carried out by Thompson [1998]. The following paragraph briefly summarizes major achievements.

Self-recovery of analog circuits has been reported in [Zebulum et al. 2003]. Fault tolerance is often investigated using simulators but, for instance, the JPL team has performed experiments within a real extreme environment—functional recovery in high temperatures, low temperatures and radiation [Stoica et al. 2001; Stoica et al. 2004; Zebulum et al. 2005]. Lohn et al. [2003] performed functional recovery of a quadrature decoder after a stuck-at-zero fault for a model of an FPGA. Garvie and Thompson [2003] have directly evolved simple digital circuits containing a built-in self-test system. The messy gate model has been introduced as a simplified analog description of a digital gate in order to investigate evolutionary fault tolerance mechanisms [Hartmann and Haddow 2004]. Masner et al. [2000] have carried out studies of the effect of representational bias on the robustness of evolved sorting networks to a range of faults. In this study, we are interested in functional recovery using EAs and the evolutionary design of fault tolerant circuits.

3. PROPOSED MODELS

This section introduces models of reconfigurable devices created in order to investigate functional recovery in VRCs.

3.1 Hypothetical Reconfigurable Device

The object of our investigations is a simple hypothetical reconfigurable device (HRD) consisting of 8 PEs, each PE has 4 inputs and 4 outputs and its behavior is defined using 74 configuration bits. Figure 3 shows that each PE can be configured to operate as logic *and*, *nand*, *or* or *xor*. The inputs of a PE can be connected to circuit inputs or to the outputs of preceding PEs. However, only up to 8 combinations are permitted in order to reduce the number of configuration bits. While the complete behavior of PE0 is defined by 6 configuration bits, the behavior of PE1–PE7 is defined by 8 configuration bits. Each of the primary outputs can be connected to one of PE0–PE7, i.e. 3 configuration bits are included into the configuration bitstream for each output. It is evident that only combinational circuits can be created in HRD.

Table I. Implementation cost of various circuits in the *FPGAsim*

Circuit	LUTs
4-input logic function	1
4-MUX (multiplexer)	5
8-MUX, PE0	11
PE1	15
PE2	19
PE3	21
PE4, PE5, PE6, PE7	23
HRD	202

3.2 Two Implementations of HRD

We will compare two implementations of the HRD in Section 4 and 5: (1) the implementation using ASIC (i.e. the HRD is implemented as a new reconfigurable ASIC) and (2) the implementation using an FPGA (i.e. the HRD is implemented on the top of an FPGA using the concept of the VRC).

Implementation (1) has a number of advantages over (2). If the same technology and operation conditions are considered, it will be faster, more area-efficient and fault tolerant since only a few components are utilized and, therefore, only 74 configuration bits (flip-flops) can be corrupted. As mentioned in Section 2.2, the advantage of (2) is that a relatively inexpensive FPGA (however, containing thousands of "sensitive" configuration bits) can be utilized and that the reconfigurable circuit is available as a soft IP core, i.e. the HRD can easily be removed/modified from/on the FPGA.

3.3 *FPGAsim*

In order to simulate the implementation of the HRD in an FPGA, a simulator of the FPGA (called *FPGAsim*) has been developed. *FPGAsim* assumes that the FPGA consists of 16-bit look-up tables (LUTs). It is also assumed that its configuration memory consists of $16 \times p$ configuration bits, where p denotes the number of LUTs. It is important to note that configuration bits defining the interconnection of LUTs and I/O are *not* considered in this initial study. They will be included in the next generation of the *FPGAsim*. If a proper configuration is uploaded into *FPGAsim*, we can obtain an implementation of the HRD. In our case, 202x16=3232 configuration bits must be set up.

As an example of a part of HRD emulated by the *FPGAsim*, Fig. 4 shows the circuits used to build PE2. All the circuits are composed of LUTs. For instance, 11 LUTs are needed to build an 8-input multiplexer. Table I lists implementation costs of various components of the HRD. All circuits and configurations were designed manually. Once the HRD is "uploaded" into *FPGAsim*, the configuration of the *FPGAsim* is not changed (except experimenting with fault tolerance).

3.4 Dependability of the Two Implementations of HRD

The behavior of the HRD is defined using 3232 configuration bits of the *FPGAsim*. The HRD has 4 primary inputs, 4 primary outputs and 74 inputs serving as its configuration inputs. It is assumed that these configuration bits are connected to a configuration memory composed of 74 flip-flops (which has not been modeled yet).

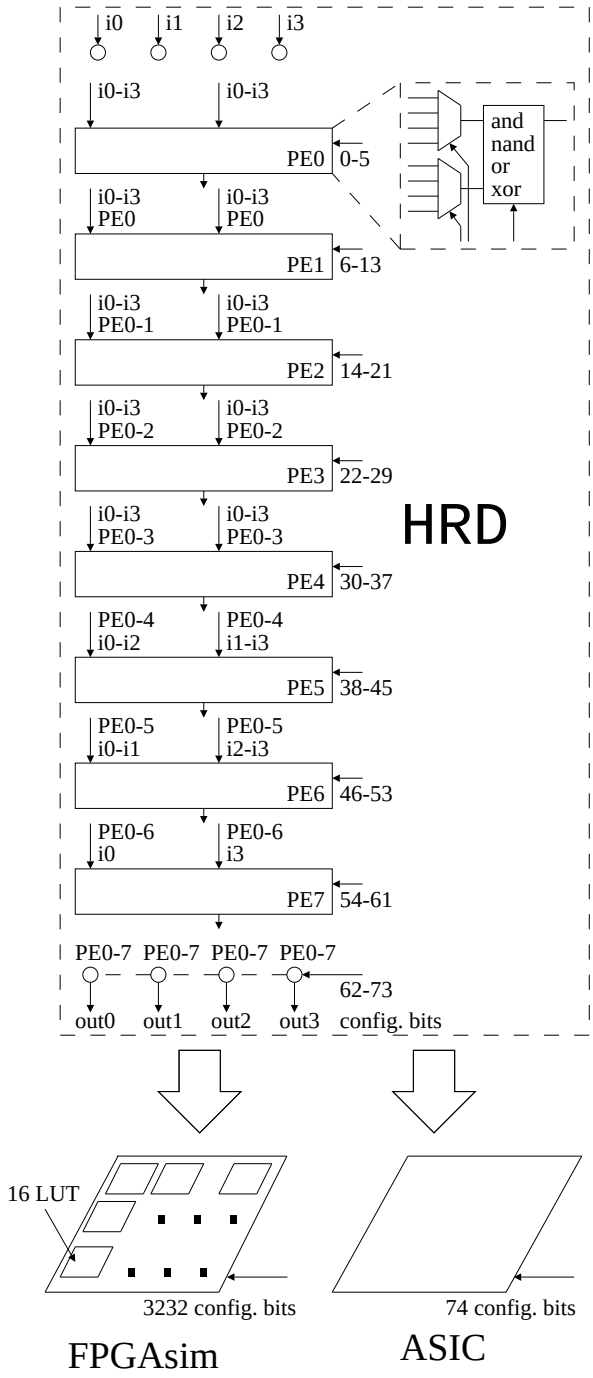

Fig. 3. Two implementations of a simple hypothetical reconfigurable device (HRD) used for the comparative study. The HRD consists of 8 PEs, four inputs (i0–i3), four outputs (out0–out3) and a 74-bit configuration memory.

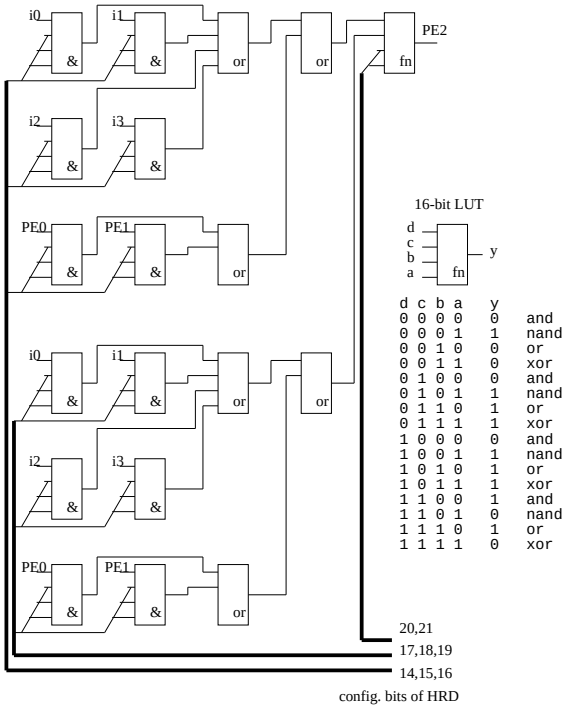

Fig. 4. Implementation of PE2 of the HRD in *FPGAsim*

Hence, in total $3232 + 74 = 3306$ configuration bits can be corrupted in our model. If we compare this to implementation (1), we can see that $3306/74 = 45$ times more configuration bits can be corrupted. The question is whether the reliability[2] of the HRD implemented in the FPGA using the concept of the VRC (as modeled by *FPGAsim*) decreases 45 times in comparison to implementation (1).

4. EVOLUTIONARY DESIGN OF FAULT TOLERANT CIRCUITS

In this study, the fault simulator simply inverts configuration bits (bit by bit). An investigation of multiple faults and other fault scenarios is left opened for the future research. We will investigate how an (independent) inversion of a single configuration bit influences functionality of a circuit uploaded into the HRD for implementations (1) and (2). For implementation (2), we will invert (i.e. damage) the configuration bits of *FPGAsim* as well as HRD.

For the purposes of this paper, let us characterize fault tolerance of a circuit uploaded into a reconfigurable device using the parameter α which indicates the sensitivity of the circuit to the changes in the configuration bitstream. α_1 denotes the number of configuration bits which must not be inverted to ensure a perfect functionality of the circuit uploaded into the HRD. For example, for our HRD, if $\alpha_1 = 74$ then an inversion of any configuration bit will influence the functionality of the circuit uploaded into HRD (i.e. the circuit is extremely sensitive to

[2]Here, by reliability we mean the sensitivity of a device to faults in its configuration bitstream.

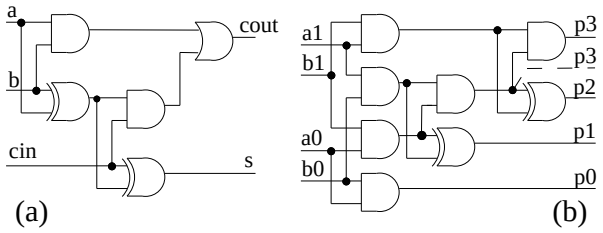

Fig. 5. Conventional implementations of test circuits: (a) 1-bit full adder, (b) 2-bit multiplier

changes in the configuration bitstream). If $\alpha_1 = 0$, the functionality of the circuit uploaded into HRD will remain unchanged although all the configuration bits may be inverted.

Let α_2 denote the number of configuration bits of *FPGAsim* which must not be inverted to ensure a perfect functionality of the circuit uploaded into HRD[3]. If $\alpha_2 = 3232$ then an inversion of any configuration bit of the *FPGAsim* will influence the functionality of the circuit uploaded into the HRD.

Let α_3 denote the number of configuration bits of *FPGAsim* and HRD which must not be inverted to ensure a perfect functionality of the circuit uploaded into HRD, i.e.

$$\alpha_3 = \alpha_1 + \alpha_2 \tag{1}$$

The fact that errors in the *FPGAsim* and HRD may compensate each other is ignored here because it is assumed that its influence is not significant.

Assuming the inversions of configuration bits at a constant rate R (e.g., memory cells per day) and their independent occurrence, we can calculate the number of potential errors in the circuit behavior as $\alpha_1 R$ cells per day for the implementation (1). For implementation (2), it is $\alpha_3 R$ cells per day. Then, the ratio

$$\beta = \frac{\alpha_3}{\alpha_1} \tag{2}$$

determines how the reliability of the circuit uploaded in the HRD (which is implemented in *FPGAsim*) decreases in comparison to implementation (1).

4.1 Test Circuits

To be suitable for the size of the HRD, we decided to use a one-bit full adder (1bFA) and two-bit multiplier (2bMU) as test circuits. Their conventional implementations (taken from [Miller et al. 2000] and depicted in Fig. 5) will be utilized for the comparison with the evolved circuits and for the fault tolerance analysis. The 74-bit configurations of conventional 1bFA as well as 2bMU were created manually, uploaded into the HRD and their functionalities verified. Note that the unused configuration bits (corresponding to the unused PEs and unused outputs) were set at logic "0". There are two implementations of the multiplier depicted in Fig. 5b: the first utilizes 7 PEs and the second 8 PEs.

[3]Note that the HRD is implemented in *FPGAsim*.

Table II. FT analysis of conventional implementations of test circuits

Circuit	PE	α_1	α_2	α_3	β
1bFA-conv	5	43	209	252	5.86
2bMU-conv-1	7	69	354	423	6.13
2bMU-conv-2	8	66	334	400	6.06

Table III. FT analysis of evolved circuits

Circuit	PE	α_1	α_2	α_3	β
1bFA-ev-a	5	42	188	230	5.47
1bFA-ev-b	5	45	217	262	5.82
1bFA-ev-c	7	54	265	319	5.90
1bFA-ev-d	5	41	215	256	6.24
2bMU-ev-a	7	65	335	400	6.15
2bMU-ev-b	7	66	331	397	6.01
2bMU-ev-f	8	72	353	425	5.90
2bMU-ev-g	7	66	330	396	6.00

4.2 Problem 1: Analysis of Fault Tolerance for Conventional Implementations of Test Circuits

Table II summarizes the results of the fault tolerance analysis obtained for conventional implementations of test circuits when implemented according to strategies (1) and (2). For instance, in the case of the conventional 1bFA, 43 (independent) inversions out of 74 configuration bits and 209 out of 3232 configuration bits of the *FPGAsim* (i.e. 252 in total) lead to an unsatisfactory behavior of the adder. The remaining configuration bits ($31 + 3023 = 3054$) can be inverted without any changes observable in the behavior of the adder. Surprisingly, $\beta = 252/43 = 5.86$ is much less than the pessimistic prediction (in Section 3.4) that the reliability should decrease 45 times. The reason for this result is that 3 PEs and 2 outputs of the HRD are not utilized at all and thus the LUTs provide a lot of redundancy.

In case of the multiplier 2bMU-conv-1 (see Table II), 7 PEs are utilized. Only 5 out of 74 configuration bits may be inverted without any changes observable in the behavior of the multiplier. Because $\alpha_2 = 354$, the reliability of implementation (2) decreases 6.13 times in comparison with implementation (1). The values of β are similar for all circuits although the multipliers use more PEs. The reason is that the implementation of VRC exhibits an inherent redundancy.

4.3 Problem 2: Evolutionary Design of Circuits in HRD

The objective of this task is to confirm that 1bFA and 2bMU can be evolved in the HRD (i.e. independently of a chosen implementation (1) or (2)).

In order to evolve these circuits, we have applied the evolutionary algorithm with the following parameters setting. The chromosome is a 74-bit binary string directly corresponding to the configuration bitstream of the HRD. The initial population consisting of 128 chromosomes is generated randomly; subsequent populations are formed using deterministic selection. The four chromosomes with the highest fitness values are considered as parents and their mutants form every new population. Mutation inverts a randomly selected bit. The highest-scored individual is always moved to next population. In the fitness calculation, all possible input combinations

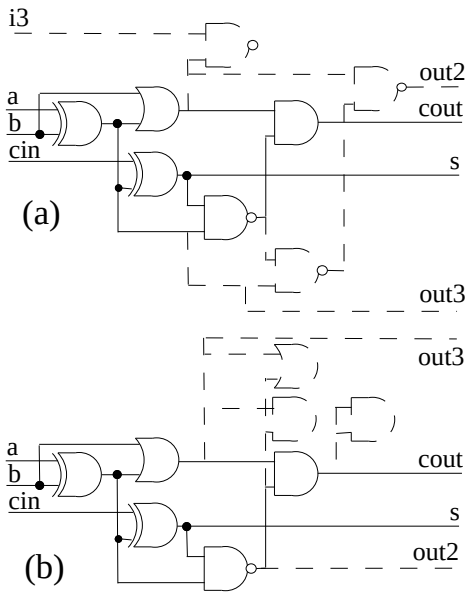

Fig. 6. Examples of evolved 1b adders: (a) 1bFA-ev-a, (b) 1bFA-ev-k. Unused elements are given dashed.

are supplied at the inputs of HRD and the number of correctly calculated output bits represents the fitness value of a candidate circuit. The evolution is typically stopped when (1) a perfect solution is found, (2) no improvement of the best fitness value occurs in the recent 1000 generations, or (3) after 3000 generations.

It proved to be not difficult to evolve these circuits. Nine correct adders can be evolved out of ten runs in an average of 380 generations. Four correct multipliers can be evolved out of ten runs in an average of 829 generations. A single run requires on average 6.8 seconds on a Pentium 4 processor running at 2.6 GHz. Figure 6 shows examples of evolved 1bFA. Table III gives results of the fault tolerance analysis for some of the evolved circuits.

4.4 Problem 3: Evolutionary Design of Fault Tolerant Circuits in HRD

The objective of this task is to evolve 1bFA and 2bMU that exhibit better fault tolerance than the circuits presented in the previous sections. In order to improve the fault tolerance of circuits in the HRD, and in addition to the functionality evaluation, the evolution must minimize (in independent runs): (i) α_1, (ii) α_2 or (iii) α_3. The process of generating the initial populations and the fitness functions have been modified compared to the evolutionary algorithm proposed in the previous section. In scenarios (ii) and (iii), the initial population is created from the best circuits evolved so far. The fitness functions (which combine two criteria—maximizing functionality and fault tolerance) take the following forms:

Table IV. FT analysis of evolved fault tolerant circuits

Circuit	scenario	PE	α_1	α_2	α_3	β
1bFA-ev-e	(i)	5	35	214	249	7.11
1bFA-ev-f	(i)	5	38	202	240	6.31
1bFA-ev-h	(i)	5	39	201	240	6.15
1bFA-ev-i	(i)	5	**34**	196	230	6.76
1bFA-ev-j	(ii)	5	42	185	227	5.40
1bFA-ev-k	(iii)	5	39	184	**223**	5.72
1bFA-ev-i	(iii)	5	42	184	226	5.38
2bMU-ev-h	(i)	7	**64**	333	397	6.20
2bMU-ev-i	(i)	8	71	369	440	6.20
2bMU-ev-j	(i)	7	65	330	395	6.08
2bMU-ev-k	(i)	7	65	332	397	6.11
2bMU-ev-l	(ii)	7	66	325	391	5.92
2bMU-ev-m	(iii)	7	66	325	**391**	5.92

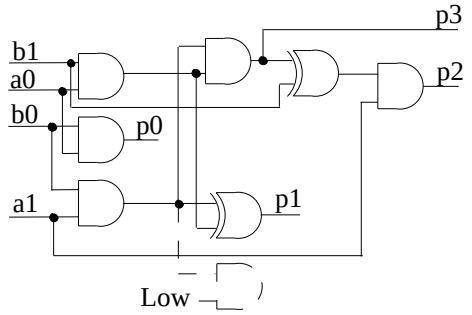

Fig. 7. Robust multiplier 2bMU-ev-m for the HRD

$$\begin{array}{ll} (i) & f = CB * 100 + (74 - \alpha_1) \\ (ii) & f = CB * 10000 + (3232 - \alpha_2) \\ (iii) & f = CB * 10000 + (3232 + 74 - \alpha_3) \end{array} \quad (3)$$

where CB is the number of output bits calculated correctly by a candidate circuit for all possible input combinations. α_i is calculated for the same candidate circuit. The constants 100 and 10000 are used to give a much greater emphasis to the correctness of the circuits and less to the fault tolerance.

Table IV summarizes the results obtained for scenarios (i), (ii) and (iii). All the circuits are functionally perfect; however, they differ in reliability. The best circuits evolved for implementation (1) are 1bFA-ev-i ($\alpha_1 = 34$) and 2bMU-ev-h ($\alpha_1 = 64$). The best circuits evolved for implementation (2) are 1bFA-ev-k ($\alpha_3 = 223$) and 2bMU-ev-l ($\alpha_3 = 391$). These circuits are more reliable than the circuits from Fig. 5 (see Table II and III). Figure 7 shows the best evolved 2bMU. The best-evolved 1bFA-ev-k is in fact identical to the circuit depicted in Fig. 6.

5. EVOLUTIONARY FUNCTIONAL RECOVERY

The objective of this task is to perform evolutionary functional recovery of a circuit placed in the HRD. The evolutionary algorithm is utilized to change the faulty

configuration of a reconfigurable device in order to recover the required function. In these experiments, it is assumed that faults are injected into the configuration memory in steps, bit by bit. At each step, one (randomly selected) configuration bit is set at logic "0" (modeling thus a stuck-at-zero) and this corruption is *not* repairable. In implementation (1), 74 bits can be corrupted; in implementation (2), 74+3232 bits can be corrupted. As the number of corruptions grows, it becomes more difficult to repair the functionality of the circuit. Note that some of the injected logic "0" might not cause any problem because a corresponding configuration bit is already set at logic "0". Similarly to Section 4, the investigation of other fault scenarios is left opened for the future research.

The goal is to recover the functionality of the one-bit full adder and the two-bit multiplier. When the failure of a circuit is detected, the evolutionary algorithm is used to recover the functionality of the circuit in HRD. 100 faults are generated for implementation (1) and 800 faults are generated for implementation (2). Therefore, in a single experiment, the EA is executed at most 100 (resp. 800) times. This experiment is repeated 10 times for both circuits.

Two strategies of initial population seeding are investigated. In the first strategy (denoted as R – random), initial populations for all the EA runs are generated randomly. In the second strategy (denoted as H – heuristic), the initial population for the first run of the EA (which is executed to repair the first fault) is seeded by correct circuits and all subsequent initial populations are generated as a copy of the last population from the previous run of the EA. Because of this knowledge it is assumed that the H strategy could enable finding a correct solution more quickly than the R strategy. In each experiment, we measured two characteristics:

—F_m – the maximum number of faults for which the perfect functionality can fully be recovered and

—F_1 – the first occurrence of an unrepaired fault in the sequence of faults.

These characteristics should clearly indicate when a perfect functionality cannot be recovered at all and when the EA is starting to have problems with repairing configurations.

Examples of typical runs for both strategies are given in Figure 8. Note that the maximum fitness value is 16 for the adder and 64 for the multiplier. As Table V shows, the average number of generations is higher for the R strategy (because it always starts from scratch). In Figure 9, we have plotted the number of generations of EA that were executed to repair a fault in the typical runs shown in Figure 8. When the number of generations is zero then the corresponding fault has no influence on the circuit functionality, i.e. the initial population contains a correct solution in this particular case.

Experimental results in Tables VI and VII show that when the H strategy is utilized then the first unrepaired fault occurs later than in case of the R strategy (i.e. $F_1^R < F_1^H$ in average). However, the R strategy is on average able to repair a circuit after many more faults than the H strategy (i.e. $F_m^R > F_m^H$ on average). For example, Table VI shows that when the H strategy is utilized then the EA is able to repair the adder after 292.8 faults on average; however, 427.8 faults are acceptable on average in case when the R strategy is utilized (for implementation (2)). The problem is that the R strategy is much less reliable than the H strategy for rare

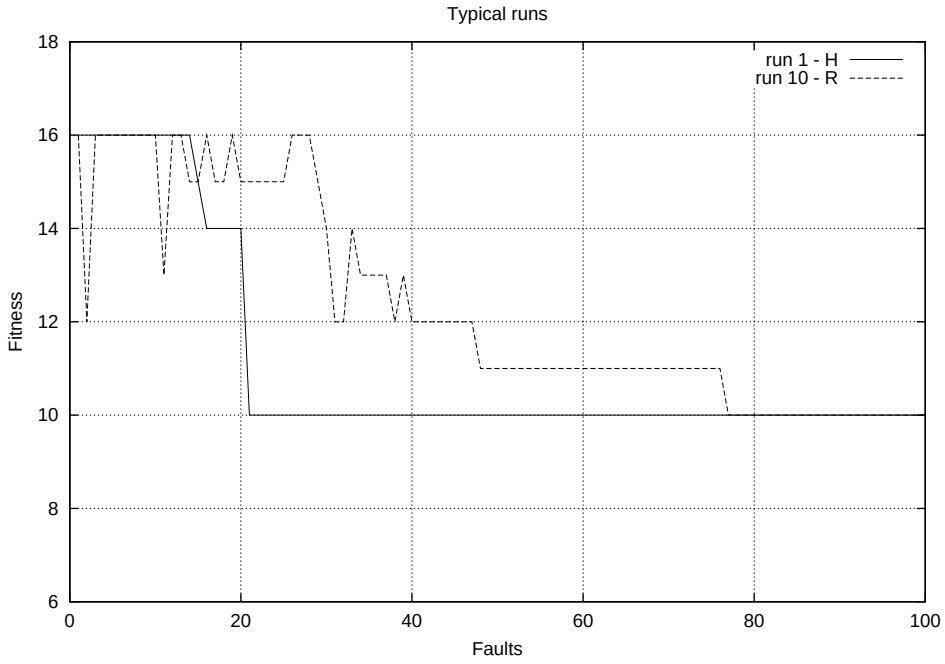

Fig. 8. A typical run of the R strategy and the H strategy for 1b-FA and implementation (1). $F_1^H = F_m^H = 14$, $F_1^R = 2, F_m^R = 28$

Table V. The average number of generations

Circuit	Implementation	Strategy	Generations
1b-FA	1	R	971.2
1b-FA	1	H	838.5
1b-FA	2	R	1096.5
1b-FA	2	H	655.1
2b-mult	1	R	1165.8
2b-mult	1	H	959.2
2b-mult	2	R	1344.5
2b-mult	2	H	832.4

corruptions of the configuration bitstream: as the initial population is generated randomly for the R strategy (and so its diversity is high), a correct solution can be evolved also for many corruptions in the configuration. However, the EA (as a stochastic process) sometimes is not able to deliver a perfect solution in the case that only a few configuration bits are corrupted. On the other hand, the H strategy ensures good results for small corruptions of the configuration (see parameter F_1).

Figures 10 and 11 show the development of the average, minimum and maximum fitness values calculated from 10 independent runs for both circuits and both implementations. As expected, the H strategy produces much smoother curves than the R strategy.

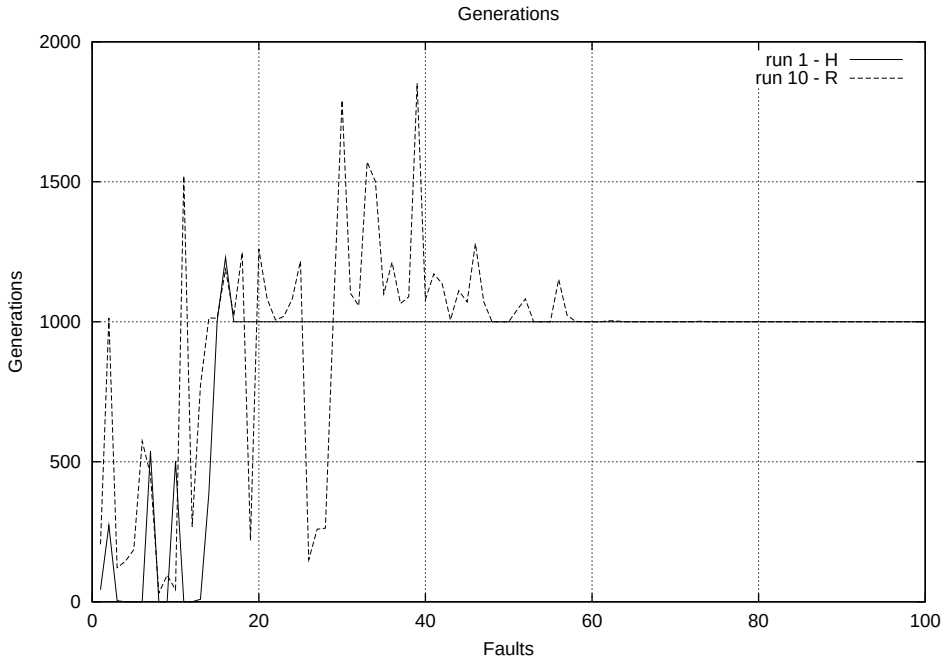

Fig. 9. The number of generations for a typical run of the R strategy and the H strategy (for 1b-FA and implementation (1))

Table VI. Evolutionary functional recovery of the 1bFA – summary of 10 independent runs

Feature	strategy	mean	std. dev.	min	max
Impl. (1) F_1	R	4.4	4.43	1	16
Impl. (1) F_m	R	26.8	13.98	6	48
Impl. (1) F_1	H	13.7	12.59	1	37
Impl. (1) F_m	H	19.2	10.45	3	37
Impl. (2) F_1	R	4.7	4.56	1	16
Impl. (2) F_m	R	427.8	98.33	251	512
Impl. (2) F_1	H	174.6	128.68	3	399
Impl. (2) F_m	H	292.8	174.90	19	546

The results can be interpreted in the following way: in case of implementation (1) 31.1% of the configuration bits can be corrupted (i.e. set at logic "0") while the adder remains fully functional. In case of implementation (2), only 10.9% of the configuration bits can be corrupted (on average) to maintain the functionality of the adder. These values were obtained using mean values of F_m for the R and H strategy. Similarly for the multiplier, it is 11.2% of the configuration bits for implementation (1) and 5.6% for implementation (2).

6. CONCLUSIONS

We have performed an initial study of the reliability of virtual reconfigurable circuits and of the evolutionary functional recovery of the circuits uploaded in virtual

Table VII. Evolutionary functional recovery of the 2-bit multiplier – summary of 10 independent runs

Feature	strategy	mean	std. dev.	min	max
Impl. (1) F_1	R	1.30	0.46	1	2
Impl. (1) F_m	R	9.7	4.31	1	15
Impl. (1) F_1	H	5.90	4.28	1	13
Impl. (1) F_m	H	6.90	4.46	1	13
Impl. (2) F_1	R	1.60	0.92	1	4
Impl. (2) F_m	R	220.2	90.17	46	381
Impl. (2) F_1	H	125.1	40.13	83	205
Impl. (2) F_m	H	147.7	38.63	83	205

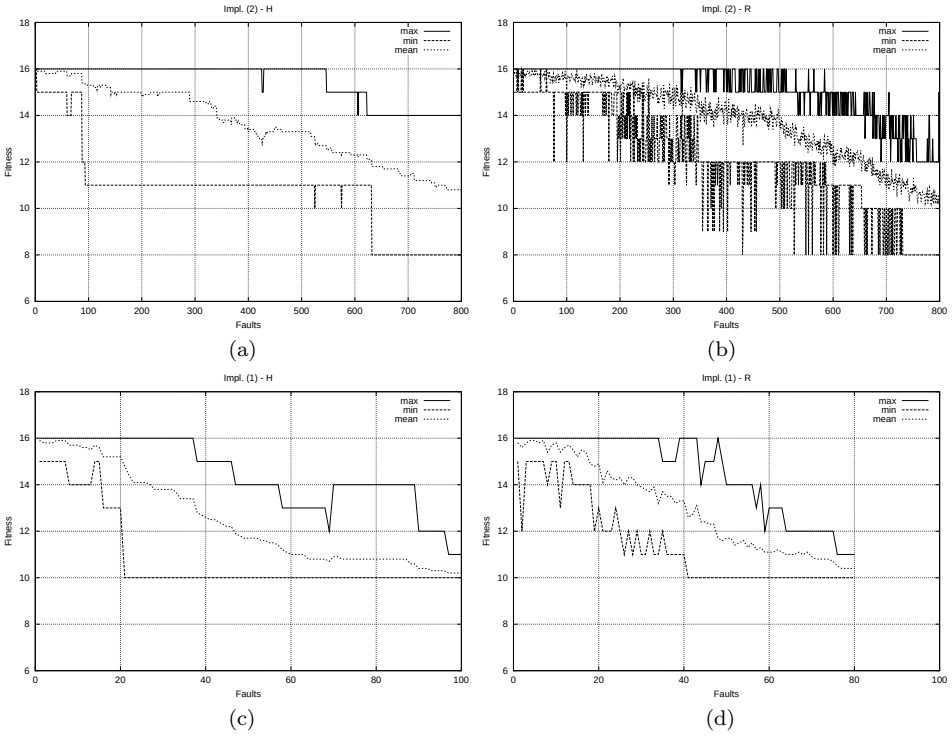

Fig. 10. Functional recovery of a full 1-bit adder. The average, minimum and maximum fitness values calculated from 10 independent runs: a) implementation 2 and the H strategy, b) implementation 2 and the R strategy, c) implementation 1 and the H strategy, d) implementation 1 and the R strategy

reconfigurable circuits. We approached the problem with a hypothetical reconfigurable device, its simulator and a simple (incomplete) FPGA simulator. Only errors in a part of the configuration memory which defines configuration of LUTs were considered.

Under the conditions of our experiments and considering limits of the proposed models, we can conclude that:

—the implementations of evolvable systems which are based on the idea of VRC are

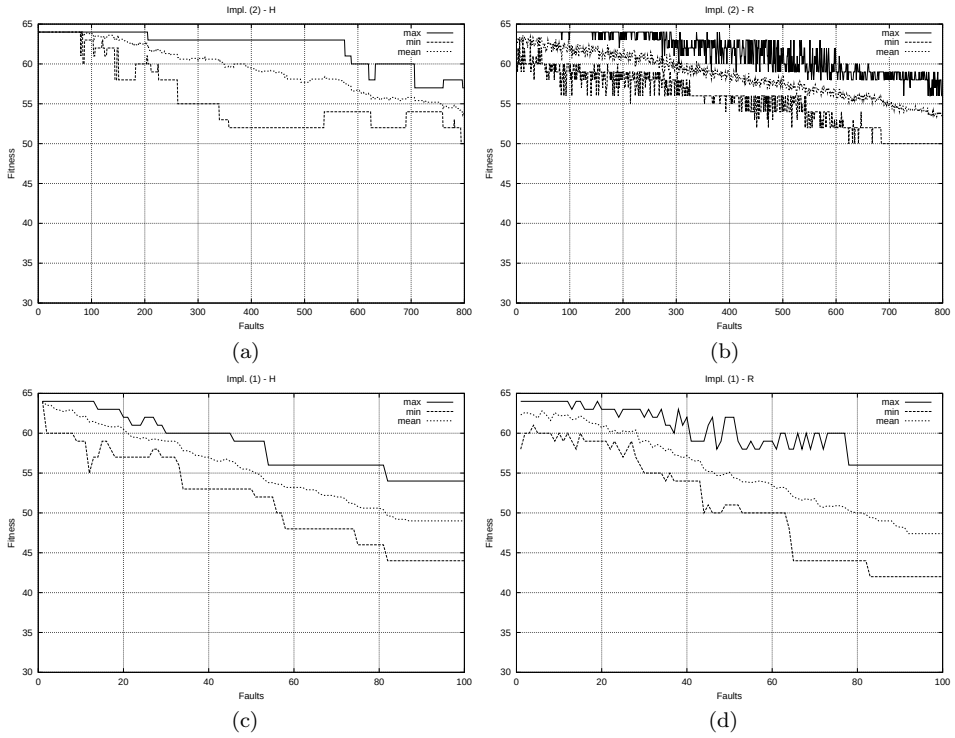

Fig. 11. Functional recovery of a two-bit multiplier. The average, minimum and maximum fitness values calculated from 10 independent runs: a) implementation 2 and the H strategy, b) implementation 2 and the R strategy, c) implementation 1 and the H strategy, d) implementation 1 and the R strategy

not as sensitive to faults in the configuration memory as it could be assumed if one considers their higher implementation cost (however, they are more sensitive to faults than a VRC-less solution);

—it is possible to improve the fault tolerance of digital circuits by means of their evolutionary design directly in the VRC and

—redundant components (LUTs) help to protect the circuit from faults in the configuration bitstream.

Experiments performed with the evolutionary functional recovery have shown different behaviors for different strategies of the initial population seeding. While an EA seeded randomly is able to repair a circuit in the case where there are many faults in the configuration memory, an EA seeded heuristically exhibits more stable behavior.

All these experimental results obtained from simulations must be confirmed by real implementations of VRCs in FPGAs and in real environments.

Acknowledgments

This work was partially supported by the Grant Agency of the Czech Republic under contract No. 102/06/0599 *Methods of polymorphic digital circuit design* and the Research Plan No. MSM 0021630528 – *Security-Oriented Research in Information Technology.*

REFERENCES

Alderighi, M., Candelori, A., Casini, F., D'Angelo, S., Mancini, M., Paccagnella, A., Pastore, S., and Sechi, G. R. 2005. Heavy ion effects on configuration logic of virtex fpgas. In *Proceedings of the 11th IEEE International On-Line Testing Symposium*. IEEE Computer Society, Saint Raphael, France, 49–53.

Bernardi, P., Reorda, M. S., Sterpone, L., and Violante, M. 2004. On the evaluation of seu sensitiveness in sram-based fpgas. In *Proceedings of the 10th IEEE International On-Line Testing Symposium*. IEEE Computer Society, Madeira Island, Portugal, 115–120.

Bradley, D., Ortega-Sanchez, C., and Tyrrell, A. M. 2000. Embryonics + immunotronics: A bio-inspired approach to fault tolerance. In *Proceedings of the 2nd NASA/DoD Workshop on Evolvable Hardware*. IEEE Computer Society, Palo Alto, 215–224.

Carmichael, C., Caffrey, M., and Salazar, A. 2000. Correcting single-event upsets through virtex partial configuration. Xilinx Application Note XAPP 216.

Garvie, M. and Thompson, A. 2003. Evolution of self-diagnosing hardware. In *Proceedings of the 6th Conference on Evolvable Systems: From Biology to Hardware*. LNCS, vol. 2606. Springer, Berlin, 238–248.

Glette, K. and Torresen, J. 2005. A flexible on-chip evolution system implemented on a xilinx virtex-ii pro device. In *Proceedings of the 6th Conference on Evolvable Systems: From Biology to Hardware*. LNCS, vol. 3637. Springer, Berlin, 66–75.

Gwaltney, D. and Dutton, K. 2005. A VHDL core for intrinsic evolution of discrete time filters with signal feedback. In *Proceedings of the 2005 NASA/DoD Conference on Evolvable Hardware*. IEEE Computer Society, Washington D.C., 43–50.

Hartmann, M. and Haddow, P. C. 2004. Evolution of fault-tolerant and noise-robust digital designs. *IEE Proceedings – Computers and Digital Techniques 151,* 4, 287–294.

Higuchi, T., Niwa, T., Tanaka, T., Iba, H., de Garis, H., and Furuya, T. 1993. Evolving hardware with genetic learning: A first step towards building a Darwin machine. In *Proceedings of the 2nd International Conference on From Animals to Animats 2: Simulation of Adaptive Behavior: Simulation of Adaptive Behavior*. MIT Press, Honolulu, Hawai, 417–424.

Korenek, J. and Sekanina, L. 2005. Intrinsic evolution of sorting networks: A novel complete hardware implementation for fpgas. In *Proceedings of the 6th Conference on Evolvable Systems: From Biology to Hardware*. LNCS, vol. 3637. Springer, Berlin, 46–55.

Koza, J. R., Bennett, F. H., Andre, D., and Keane, M. A. 1999. *Genetic Programming III: Darwinian Invention and Problem Solving*. Morgan Kaufmann, San Francisco.

Lohn, J. D., Larchev, G. V., and DeMara, R. F. 2003. A genetic representation for evolutionary fault recovery in virtex fpgas. In *Proceedings of the 5th Conference on Evolvable Systems: From Biology to Hardware*. LNCS, vol. 2606. Springer, Berlin, 47–56.

Mange, D., Sipper, M., Stauffer, A., and Tempesti, G. 2000. Towards robust integrated circuits: The embryonics approach. *Proceedings of IEEE 88,* 4, 516–541.

Martinek, T. and Sekanina, L. 2005. An evolvable image filter: Experimental evaluation of a complete hardware implementation in fpga. In *Proceedings of the 6th Conference on Evolvable Systems: From Biology to Hardware*. LNCS, vol. 3637. Springer, Berlin, 76–85.

Masner, J., Cavalieri, J., Frenzel, J. F., and Foster, J. A. 2000. Size versus robustness in evolved sorting networks: Is bigger better? In *Proceedings of the 2nd NASA/DoD Workshop on Evolvable Hardware*. IEEE Computer Society, Palo Alto, 81–90.

Miller, J., Job, D., and Vassilev, V. 2000. Principles in the evolutionary design of digital circuits – Part I. *Genetic Programming and Evolvable Machines 1,* 1, 8–35.

Pradhan, D. 1996. *Fault-Tolerant Computer System Design*. Prentice Hall, New Jersey.

qpro 2001. Qpro Virtex 2.5v QML high-reliability FPGAs. Xilinx data sheet DS002.

Salomon, R., Widiger, H., and Tockhorn, A. 2006. Rapid evolution of time-efficient packet classifiers. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*. IEEE CIS, Vancouver, Canada, 2793–2799.

Sekanina, L. 2003. Virtual reconfigurable circuits for real-world applications of evolvable hardware. In *Proceedings of the 5th Conference on Evolvable Systems: From Biology to Hardware*. LNCS, vol. 2606. Springer, Berlin, 186–197.

Sekanina, L. 2004. *Evolvable components: From Theory to Hardware Implementations*. Springer, Berlin.

Sekanina, L. 2006. On dependability of FPGA-based evolvable rardware systems that utilize virtual reconfigurable circuits. In *Proceedings of the Computing Frontiers Conference*. ACM, Ischia, 221–228.

Sekanina, L. and Friedl, S. 2004. An evolvable combinational unit for fpgas. *Computing and Informatics 23,* 5, 461–486.

Sekanina, L., Martinek, T., and Gajda, Z. 2006. Extrinsic and intrinsic evolution of multifunctional combinational modules. In *Proceedings of the 2006 IEEE Congress on Evolutionary Computation*. IEEE CIS, Vancouver, Canada, 9676–9683.

Stoica, A., Keymeulen, D., Arslan, T., Duong, V., Zebulum, R. S., Ferguson, I., and Guo, X. 2004. Circuit self-recovery experiments in extreme environments. In *Proceedings of the 2004 NASA/DoD Conference on Evolvable Hardware*. IEEE Computer Society, Seattle, 142–145.

Stoica, A., Keymeulen, D., and Zebulum, R. S. 2001. Evolvable hardware solutions for extreme temperature electronics. In *Proceedings of the 3rd NASA/DoD Workshop on Evolvable Hardware*. IEEE Computer Society, Long Beach, 93–97.

Stoica, A., Keymeulen, D., Zebulum, R. S., Thakoor, A., Daud, T., Klimeck, G., Jin, Y., Tawel, R., and Duong, V. 2000. Evolution of analog circuits on field programmable transistor arrays. In *Proceedings of the 2nd NASA/DoD Workshop on Evolvable Hardware*. IEEE Computer Society, Palo Alto, 99–108.

Thompson, A. 1998. *Hardware Evolution: Automatic Design of Electronic Circuits in Reconfigurable Hardware by Artificial Evolution*. Springer, London.

Tyrrell, A. M., Hollingworth, G., and Smith, S. 2001. Evolutionary strategies and intrinsic fault tolerance. In *Proceedings of the 3rd NASA/DoD Workshop on Evolvable Hardware*. IEEE Computer Society, Long Beach, 98–106.

Tyrrell, A. M., Krohling, R., and Zhou, Y. 2004. A new evolutionary algorithm for the promotion of evolvable hardware. *IEE Proceedings – Computers and Digital Techniques 151,* 4, 267–275.

Vasicek, Z. and Sekanina, L. 2007. An evolvable hardware system in Xilinx Virtex II Pro FPGA. *Int. J. Innovative Computing and Applications 1,* 1, 63–73.

Wirthlin, M. J., Johnson, E., Rollins, N., Caffrey, M., and Graham, P. 2003. The reliability of fpga circuit designs in the presence of radiation induced configuration upsets. In *Proceedings of the 11th IEEE Symposium on Field-Programmable Custom Computing Machines*. IEEE Computer Society, Washington D.C., 133–142.

Zebulum, R. S., Keymeulen, D., Duong, V., Guo, X., Ferguson, M. I., and Stoica, A. 2003. Experimental results in evolutionary fault-recovery for field programmable analog devices. In *Proceedings of the 5th NASA/DoD Workshop on Evolvable Hardware*. IEEE Computer Society, Chicago, 192–198.

Zebulum, R. S., Stoica, A., Keymeulen, D., Sekanina, L., Ramesham, R., and Guo, X. 2005. Evolvable hardware system at extreme low temperatures. In *Proceedings of the 6th Conference on Evolvable Systems: From Biology to Hardware*. LNCS, vol. 3637. Springer, Berlin, 37–45.

Zhang, Y., Smith, S., and Tyrrell, A. 2004. Intrinsic evolvable hardware in digital filter design. In *Applications of Evolutionary Computing*. LNCS, vol. 3005. Springer, Berlin, 389–398.

Received August 2006; accepted December 2006