On Dependability of FPGA-Based Evolvable Hardware Systems That Utilize Virtual Reconfigurable Circuits

Lukas Sekanina
Faculty of Information Technology
Brno University of Technology
Bozetechova 2, 612 66 Brno, Czech Republic
sekanina@fit.vutbr.cz

ABSTRACT

This paper describes experiments conducted to estimate how the use of (area-demanding) virtual reconfigurable circuits (VRC) influences the dependability of FPGA-based evolvable systems. It is shown that these systems are not so sensitive to faults as their area-demanding implementations could evoke. Evolutionary techniques are utilized to design fault tolerant circuits in a virtual reconfigurable circuit and to perform their automatic functional recovery in case of occurence of faults in a configuration memory of FPGA. All the experiments are performed on models of reconfigurable devices. This paper does not claim that the use of the VRC improves the dependability; it shows how the use of VRCs could influence the dependability.

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

Keywords

evolutionary algorithms, evolvable hardware, FPGA, dependability

1. INTRODUCTION

It is a well-known problem that the reconfiguration subsystem of current field programmable gate arrays (FPGAs) is not suitable for evolvable hardware [17]. In evolvable hardware, we often require the possibility of a fast partial reconfiguration, including the controllable granularity of configurable elements and a transparent structure of the configuration data. Ideally, a specialized reconfigurable device should be constructed for a given application in order to meet its particular requirements. However, developing an application specific integrated circuit (ASIC) is sometimes impossible for many (mainly economic) reasons.

In order to utilize relatively inexpensive FPGAs, the idea of *virtual reconfigurable circuits* was introduced [16]. A virtual reconfigurable circuit (VRC) is, in fact, an implementation of a domain-specific reconfigurable circuit on top of an ordinary FPGA. Because the 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 evolution of logic circuits [18], image filters [10], sorting networks [7] and some other circuits. As the evolutionary algorithm (EA) can be implemented in the same FPGA in case of 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 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 supposed that evolvable hardware at the level of IP cores will become an integrated component in a variety of systems in future [21]. 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, i.e. in extreme environments exhibiting radiation and temperature levels different from Earth surface [22].

In particular, in this paper, we will be interested in faults in the configuration memory of FPGAs. The objective of this research is to perform experiments showing 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 in an FPGA. However, several hundred gates have to 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 an inherent redundancy of the VRC implementation is sometimes useful and can protect the circuits from faults. We will show on simplified models of an FPGA and VRC that the optimistic scenario can partially be taken into account. Evolutionary techniques will be utilized for the design of fault tolerant circuits in the VRC and for automatic repairing of the circuits in the VRC.

In order to perform the experimental analysis of these behaviors, the FPGA, the VRC implemented in the FPGA and a special environment testbed are needed. However, this equipment is relatively expensive. Hence we have decided to perform all experiments using simulators before physical devices will be utilized. We do believe that the obtained results will give us 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)—that are popular in evolvable hardware community (for example, see [5, 12]). The following list introduces the proposed experiments:

1. Analysis of fault tolerance for conventional implementations of test circuits.
2. Evolutionary design of test circuits in the VRC.
3. Evolutionary design of fault tolerant circuits in the VRC.
4. Evolutionary functional recovery in the VRC.

The rest of this paper is organized as follows. Section 2 recapitulates previous relevant research. The proposed models are introduced in Section 3. Section 4 describes performed experiments and obtained results. Conclusions are given in Section 5.

2. PREVIOUS RELEVANT RESEARCH

2.1 Virtual Reconfigurable Circuits and Their Applications in Evolvable Hardware

Virtual reconfigurable circuits were introduced for digital evolvable hardware as a new kind of reconfigurable platform utilizing conventional FPGAs [17, 16]. When a VRC is uploaded into the FPGA then its configuration bitstream has to cause that there will be created the following units 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.

Fig. 1 shows that the VRC is, in fact, a new reconfigurable circuit (consisting of 8 programmable elements in our example) implemented on top of an ordinary FPGA. “Virtual” PE2 depicted in detail in Fig. 1 is controlled using 6 bits that determine selection of its operands (2+2 bits) and its internal function (2 bits). This architecture is very similar

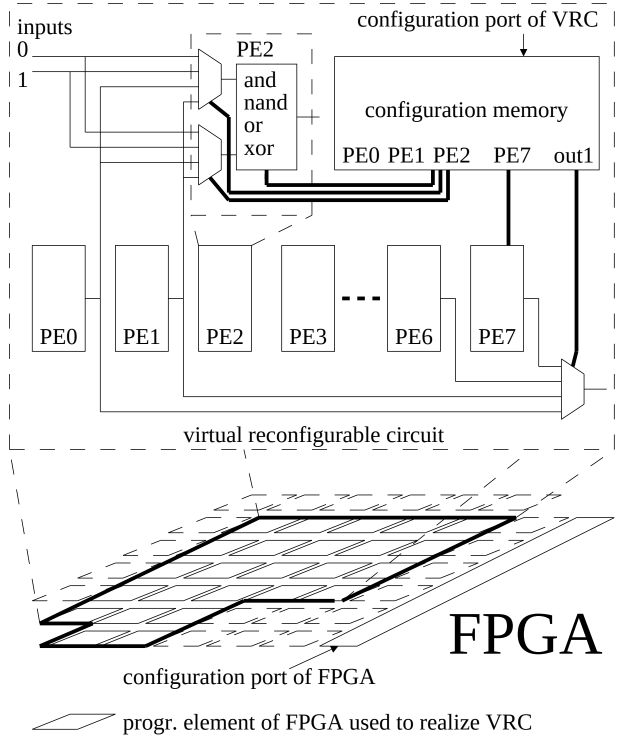

inputs
0
1
PE2
and
nand
or
xor
configuration port of VRC
configuration memory
PE0 PE1 PE2 PE7 out1
PE0
PE1
PE2
PE3
PE6
PE7
virtual reconfigurable circuit
FPGA
configuration port of FPGA
progr. element of FPGA used to realize VRC

Figure 1: Basic idea of the VRC. The VRC is described in HDL, synthesized and uploaded as a configuration bitstream to the FPGA.

to the representation employed in Cartesian Genetic Programming (CGP) that has been developed for circuit evolution [12]. Routing circuits are implemented using multiplexers. All bits of the configuration memory are connected to multiplexers that control routing and 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 evolvable systems were implemented using the idea of the VRC in an FPGA:

- Evolvable image filter for the evolution of 3x3 image operators (EA is implemented either in PC [26] or on the same FPGA [10]).
- Evolvable sorting network for up to 28 inputs (EA is implemented on the same FPGA) [7].
- Evolvable combinational circuits (EA is implemented on the same FPGA as a special circuit [18] or in the PowerPC processor on the same FPGA [6]).
- Intrinsic evolution of polymorphic combinational modules (EA is implemented on the same FPGA) [19].

2.2 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 functional 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 faults in a system increases with the number of components involved, an increasing complexity of the circuits implemented in FPGAs evokes a question of how to make them 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 [24, 2, 1]. Xilinx has offered QPRO Virtex FPGA, which is immune to latch-up, and has an acceptable total-dose tolerance [15]. 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 functionality of the device. These single-event upsets are soft errors that do not usually cause any permanent damage of an FPGA. Paper [24] gives an example, in which memory cells are anticipated to upset at a rate of 3.2 upsets/day. These upsets can mostly get away with a readback and fast partial reconfiguration; some single-event upsets require a total reconfiguration. As dose increases, 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. [4, 2]). All these approaches are based on time or space redundancy [14]. Tri module redundancy with a voting system is a classical example. In case of reconfigurable devices, we are looking for effective methods capable to recover the functionality by means of a smart reconfiguration strategy. Novel approaches have been developed in the recent years in the fields of embryonics and immunotronics. Embryonics utilizes the principles of cellular division and differentiation observed in multicellular organisms [9]. In immunotronics, artificial immune systems are built in order to protect computing devices [3]. Evolvable hardware offers other options.

2.3 Evolvable Hardware and Fault Tolerance

Various evolutionary approaches to fault tolerance are being investigated in the evolvable hardware field. In case of *explicit* fault tolerance, the requirements for fault tolerance are included into the fitness function. For instance, the quality of a candidate circuit is tested for some of possible faults. Due to the existence of redundant elements in reconfigurable devices, evolvable hardware is *inherently* fault tolerant. 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. If a critical number of elements are damaged, the functionality cannot be recovered and the chip "dies". Another method is to evolve such circuits that are able to detect malfunctions autonomously or that are easily testable. These experiments have initially been carried out by Thompson [23]. The following paragraph briefly summarizes major achievements.

Self-recovery of analog circuits has been reported in [25]. Fault tolerance is often investigated using simulators but, for instance, JPL team has performed experiments within a real extreme environment—functional recovery in high temperatures, low temperatures and radiation [22, 20]. Lohn et al performed functional recovery of a quadrature decoder after a stuck-at-zero fault for a model of an FPGA [8]. Garvie and Thompson have directly evolved simple digital circuits containing a build-in self-test system [5]. The messy gate has been introduced as a simplified analog model of a digital gate in order to investigate fault tolerance [13]. Masner et al have carried out studies of the effect of representational bias on the robustness of evolved sorting networks to a range of faults [11]. In this paper, we are interested in the functional recovery using EAs and the evolutionary design of fault tolerant circuits.

3. PROPOSED MODELS

3.1 Hypothetical Reconfigurable Device

The object of our investigations is a simple hypothetical reconfigurable device (HRD) consisting of 8 PEs, having 4 inputs and 4 outputs and the behavior fully defined using 74 configuration bits of SRAM. Figure 2 shows that each of PEs 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 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.

3.2 Two Implementations of HRD

We will compare two implementations of the HRD in Section 4: (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).

It is evident that the implementation (1) has a lot 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 2.1, 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 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 $16p$ configuration bits, where p denotes the number of PEs. It is important to note

Table 1: Implementation cost of various circuits in the *FPGAsim*

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

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. 3 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 1 lists implementation costs of various components of the HRD. All circuits and configurations have been designed manually. Once the HRD is "uploaded" into *FPGAsim*, the configuration of the *FPGAsim* is not being 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). Hence, in total 3232 + 74 = 3306 configuration bits can be corrupted in our model. If we compare this to the implementation (1), we can see that 3306/74 = 45 times more configuration bits can be corrupted. **The question is whether the reliability of the HRD implemented in the FPGA using the concept of the VRC (as modeled by *FPGAsim*) decreases 45 times in comparison to the implementation (1).**

4. EXPERIMENTS AND RESULTS

In this study, the fault simulator simply inverts configuration bits (bit by bit). We will investigate how an (independent) inversion of a single configuration bit influences functionality of a circuit uploaded into the HRD for the implementations (1) and (2). In case of the implementation (2), we will invert (i.e. damage) the configuration bits of *FPGAsim* as well as HRD.

For 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

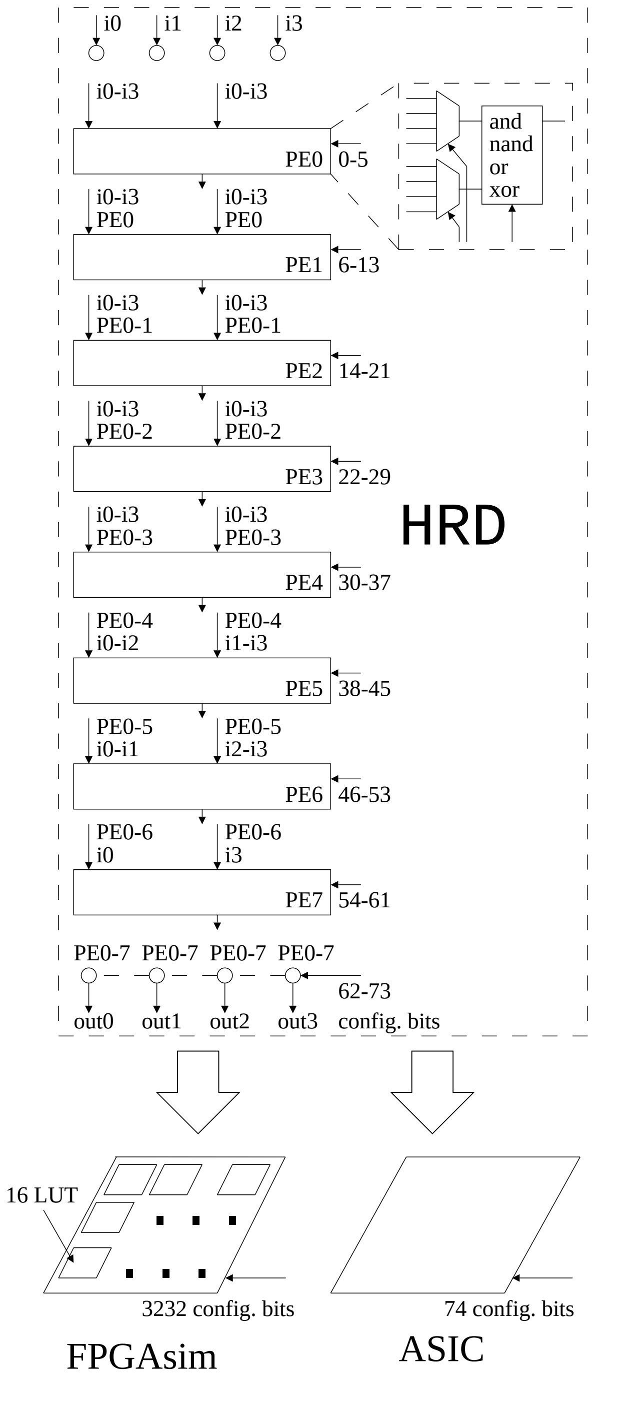
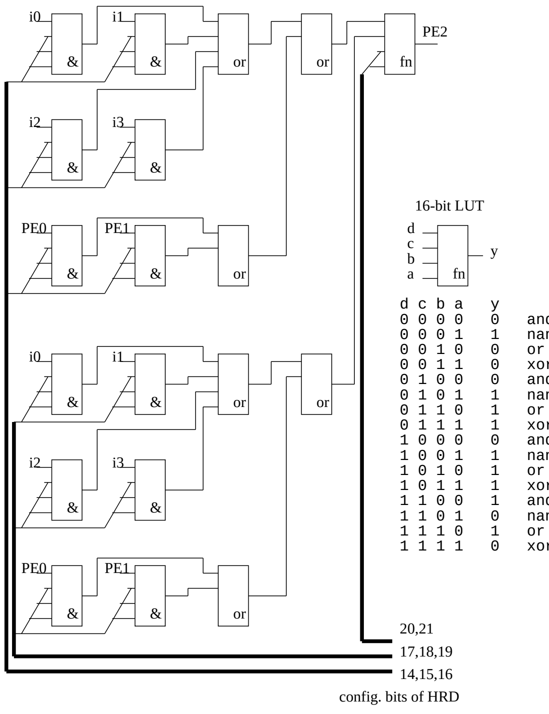


Figure 2: Two implementations of a simple hypothetical reconfigurable device (HRD) used for the comparative study. The HRD consists of 8 PEs, four inputs (i0–i3), four outputs (out0–out3) and a 74-bit configuration memory.



d	c	b	a	y	
0	0	0	0	0	and
0	0	0	1	1	nand
0	0	1	0	0	or
0	0	1	1	0	xor
0	1	0	0	0	and
0	1	0	1	1	nand
0	1	1	0	1	or
0	1	1	1	1	xor
1	0	0	0	0	and
1	0	0	1	1	nand
1	0	1	0	1	or
1	0	1	1	1	xor
1	1	0	0	1	and
1	1	0	1	0	nand
1	1	1	0	1	or
1	1	1	1	0	xor

Figure 3: Implementation of PE2 of the HRD in *FPGAsim*

is extremely sensitive to changes in the configuration bitstream). If $\alpha_1 = 0$, the functionality of the circuit uploaded into HRD will remain unchanged although all 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[1]. 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.

$$\alpha_3 = \alpha_1 + \alpha_2. \quad (1)$$

It is neglected herein that errors in the *FPGAsim* and HRD may compensate each other.

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 the implementation (2), it is $\alpha_3 R$ cells per day. Then, the ratio

$$\beta = \frac{\alpha_3}{\alpha_1} \quad (2)$$

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

[1]Note that HRD is implemented in *FPGAsim*

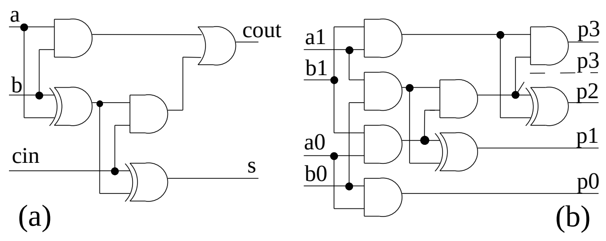


Figure 4: Conventional implementations of test circuits: (a) 1-bit full adder, (b) 2-bit multiplier

Table 2: 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

4.1 Test Circuits

As suitable for the size of the HRD, we decided to use one-bit full adder and two-bit multiplier as test circuits. Their conventional implementations (taken from [12] and depicted in Fig. 4) 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 were 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. 4b: the first utilizes 8PEs and the second only 7 PEs.

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

Table 2 summarizes the results of fault tolerance analysis obtained for conventional implementations of test circuits when implemented according to strategy (1) and (2). For instance, in 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 presumption predicting (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, all resources are utilized. Hence 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 the implementation (2) decreases 6.13 times in comparison with the implementation (1).

Table 3: 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

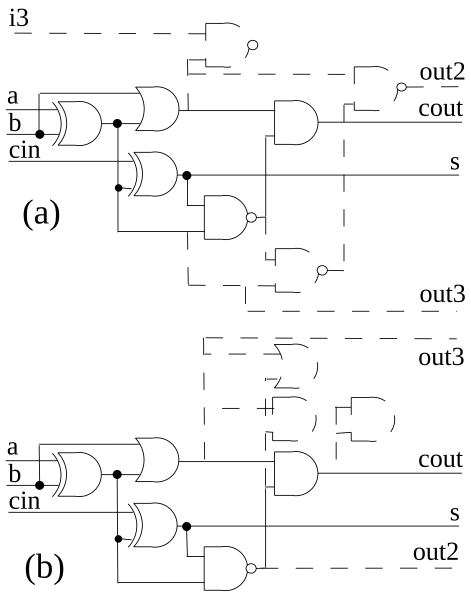


Figure 5: Examples of evolved 1b adders: (a) 1bFA-ev-a, (b) 1bFA-ev-k. Unused elements are given dashed.

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; other populations are formed using deterministic selection. The four chromosomes with the highest fitness values are considered as parents and their clones form every new population. Mutation inverts a randomly selected bit. Elitism is ensured. In the fitness calculation, all possible input combinations 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 was typically stopped (1) when no improvement of the best fitness value occurs in the recent 500 generations, or (2) after 3000 generations.

It is not difficult to evolve these circuits. Figure 5 shows examples of evolved 1bFA. Table 3 gives results of the fault tolerance analysis for some of the evolved circuits.

Table 4: 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

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 fault tolerance of circuits in the HRD, and in addition to the functionality evaluation, the evolution is to minimize: (i) α_1, (ii) α_2 and (iii) α_3. The process of the initial populations generating and fitness functions have been modified in contrast 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 take the following forms:

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

where CB is the number of output bits calculated correctly by a candidate circuit for all possible input combinations. α_i is calculated also for the same candidate circuit.

Table 4 summarizes the results obtained for scenarios (i), (ii) and (iii). All the circuits are perfectly functional; however, they differ in reliability. The best circuits evolved for the implementation (1) are 1bFA-ev-i ($\alpha_1 = 34$) and 2bMU-ev-h ($\alpha_1 = 64$). The best circuits evolved for the 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. 4 (see Table 2 and 3).

Figure 6 shows the best evolved 2bMU. The best-evolved 1bFA-ev-k is, in fact, identical with the circuit depicted in Fig. 5a. The difference, which makes this implementation more robust, lies in the configuration of unused PEs and outputs (see Fig. 5b).

4.5 Problem 4: Evolutionary Functional Recovery

The objective of this task is to perform an evolutionary functional recovery of a circuit placed in the HRD. Faults are injected in steps, bit by bit. In each step, it is assumed that one (randomly selected) configuration bit is set at log. “0” and this corruption is *not* repairable. In the implementation (1), 74 bits can be corrupted in total; in the implementation (2), 74+3232 bits can be corrupted in total. When a failure of a circuit is detected (after $1 \dots n$ steps), the evolutionary

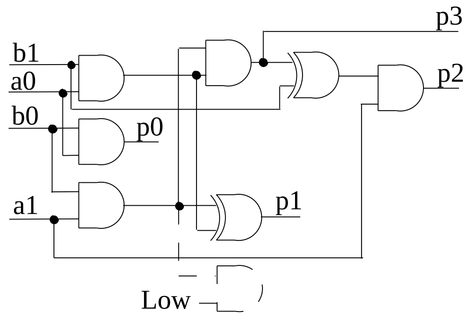


Figure 6: Robust multiplier 2bMU-ev-m for the HRD

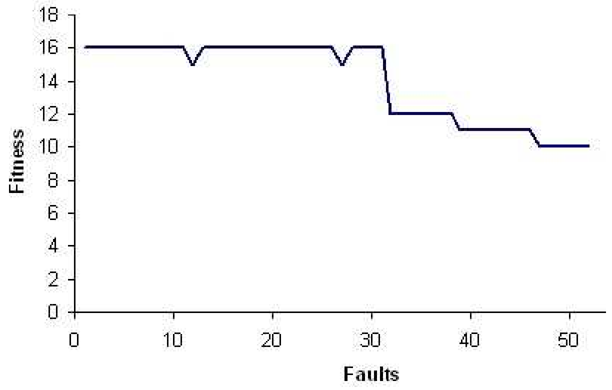


Figure 7: Typical evolutionary functional recovery of 1bFA for the implementation (1)

algorithm (taken from Section 4.3) is used to recover the functionality of the circuit in HRD. EA operates with the structure of the chromosome defined in Section 4.3; however, the corrupted configuration bits remain always at log. "0". With the growing number of corruptions, it is more difficult to repair the functionality of the circuit.

The objective was to recover the functionality of the one-bit full adder. We arranged an experiment, in which the evolutionary algorithm was executed after detecting a failure caused by setting a configuration bit at log. "0". This experiment (i.e. the sequence of EA runs) was performed 20 times for both implementations. We measured the maximum number of faults F_m for which the functionality can fully be recovered. Table 5 summarizes the results.

Figures 7 and 8 show development of the best fitness values of recovered adders for the both implementations (the maximum fitness value is 16). The data represent a typical result of this experiment. We can observe that the evolutionary algorithm is not able to recover the functionality in all cases: either the evolution fails or the amount of resources is not sufficient.

The results can be interpreted in the following way: In case of the implementation (1) 23.6% of configuration bits can be corrupted (i.e. set at log. "0") while the adder remains fully functional. In case of the implementation (2), only 12.15% of configuration bits can be corrupted (on average) to maintain functionality of the adder. These values were obtained using mean value of F_m (F_m-avr).

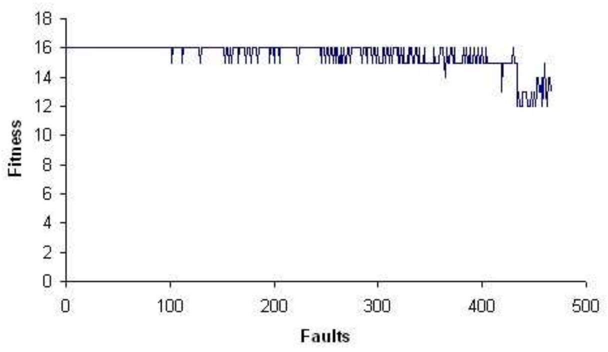


Figure 8: Typical evolutionary functional recovery of 1bFA for the implementation (2)

Table 5: Evolutionary functional recovery of 1bFA in the HRD for the implementations (1) and (2)

Impl.	cnf. bits	F_m-best	F_m-worst	F_m-avr
Impl. (1)	74	30	6	17.45
Impl. (2)	3306	584	181	401.55

5. CONCLUSIONS

In this research, we have performed an initial study of reliability of virtual reconfigurable circuits. We approached the problem with a hypothetical reconfigurable device, its simulator and a simple (incomplete) FPGA simulator. Only errors were considered in a part of the configuration memory which defines configuration of LUTs. Under conditions of our experiments and considering limits of the proposed models, we can conclude that (1) the implementations of evolvable systems which are based on the idea of VRC are not so 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), (2) it is possible to improve fault tolerance of digital circuits by means of their evolutionary design directly in the VRC and (3) redundant components (LUTs) help to protect the circuit from faults in the configuration bitstream. However, these experimental results obtained from simulations must be confirmed by real implementations of VRCs in FPGAs and in real environments.

Acknowledgments

The research was performed with the Grant Agency of the Czech Republic under No. 102/06/0599 *Methods of polymorphic digital circuit design.*

6. REFERENCES

[1] M. Alderighi, A. Candelori, F. Casini, S. DAngelo, M. Mancini, A. Paccagnella, S. Pastore, and G.R. Sechi. Heavy Ion Effects on Configuration Logic of Virtex FPGAs. In *Proc. of 11th IEEE International On-Line Testing Symposium*, pages 49–53, Los Alamitos, CA, USA, 2005. IEEE Computer Society.

[2] P. Bernardi, M. Sonza Reorda, L. Sterpone, and M. Violante. On the Evaluation of SEU Sensitiveness in SRAM-Based FPGAs. In *Proc. of 10th IEEE International On-Line Testing Symposium*, pages

115–120, Los Alamitos, CA, USA, 2004. IEEE Computer Society.

[3] D. Bradley, C. Ortega-Sanchez, and A. Tyrrell. Embryonics + Immunotronics: A Bio-Inspired Approach to Fault Tolerance. In *Proc. of the 2nd NASA/DoD Workshop on Evolvable Hardware*, pages 215–222, Los Alamitos, CA, USA, 2000. IEEE Computer Society.

[4] C. Carmichael, M. Caffrey, and A. Salazar. *Correcting Single-Event Upsets Through Virtex Partial Configuration*. Xilinx Application Note XAPP 216, 2000.

[5] M. Garvie and A. Thompson. Evolution of Self-diagnosing Hardware. In *Proc. of the 5th Int. Conf. on Evolvable Systems: From Biology to Hardware ICES'03*, volume 2606 of *Lecture Notes in Computer Science*, pages 238–248, Trondheim, Norway, 2003. Springer-Verlag.

[6] K. Glette and J. Torresen. A Flexible On-chip Evolution System Implemented on a Xilinx Virtex-II Pro Device. In *Proc. of the 6th Int. Conf. on Evolvable Systems: From Biology to Hardware ICES'05*, volume 3637 of *Lecture Notes in Computer Science*, pages 66–75, Sitges, Spain, 2005. Springer-Verlag.

[7] J. Kořenek and L. Sekanina. Intrinsic Evolution of Sorting Networks: A Novel Complete Hardware Implementation for FPGAs. In *Proc. of the 6th Int. Conf. on Evolvable Systems: From Biology to Hardware ICES'05*, volume 3637 of *Lecture Notes in Computer Science*, pages 46–55, Sitges, Spain, 2005. Springer-Verlag.

[8] J. Lohn, G. Larchev, and R. DeMara. A Genetic Representation for Evolutionary Fault Recovery in Virtex FPGAs. In *Proc. of the 5th Int. Conf. on Evolvable Systems: From Biology to Hardware ICES'03*, volume 2606 of *Lecture Notes in Computer Science*, pages 47–56, Trondheim, Norway, 2003. Springer-Verlag.

[9] D. Mange, M. Sipper, A. Stauffer, and G. Tempesti. Towards Robust Integrated Circuits: The Embryonics Approach. *Proceedings of IEEE*, 88(4):516–541, 2000.

[10] T. Martínek and L. Sekanina. An Evolvable Image Filter: Experimental Evaluation of a Complete Hardware Implementation in FPGA. In *Proc. of the 6th Int. Conf. on Evolvable Systems: From Biology to Hardware ICES'05*, volume 3637 of *Lecture Notes in Computer Science*, pages 76–85, Sitges, Spain, 2005. Springer-Verlag.

[11] J. Masner, J. Cavalieri, J. Frenzel, and J. Foster. Size versus robustness in evolved sorting networks: Is bigger better? In *The Second NASA/DoD workshop on Evolvable Hardware*, pages 81–87, Palo Alto, California, 2000. IEEE Computer Society.

[12] J. Miller, D. Job, and V. Vassilev. Principles in the Evolutionary Design of Digital Circuits – Part I. *Genetic Programming and Evolvable Machines*, 1(1):8–35, 2000.

[13] J. F. Miller and M. Hartmann. Evolving messy gates for fault tolerance: Some preliminary findings. In *The Third NASA/DoD workshop on Evolvable Hardware*, pages 116–123, Long Beach, California, 2001. IEEE Computer Society.

[14] D. K. Pradhan. *Fault-Tolerant Computer System Design*. Prentice Hall, 1996.

[15] Qpro virtex 2.5v qml high-reliability fpgas, xilinx data sheet, 2001.

[16] L. Sekanina. Virtual Reconfigurable Circuits for Real-World Applications of Evolvable Hardware. In *Proc. of the 5th Int. Conf. on Evolvable Systems: From Biology to Hardware ICES'03*, volume 2606 of *Lecture Notes in Computer Science*, pages 186–197, Trondheim, 2003. Springer-Verlag.

[17] L. Sekanina. *Evolvable Components: From Theory to Hardware Implementations*. Natural Computing Series, Springer Verlag, 2004.

[18] L. Sekanina and S. Friedl. An Evolvable Combinational Unit for FPGAs. *Computing and Informatics*, 23(5):461–486, 2004.

[19] L. Sekanina and T. Martinek and Z. Gajda. Extrinsic and Intrinsic Evolution of Multifunctional Combinational Modules. In *IEEE Congress on Evolutionary Computation*, 2006, submitted.

[20] A. Stoica, D. Keymeulen, T. Arslan, V. Duong, R. Zebulum, I. Ferguson, and X. Guo. Circuit Self-Recovery Experiments in Extreme Environments. In *Proc. of the 2004 NASA/DoD Conference on Evolvable Hardware*, pages 142–145, Seattle, USA, 2004. IEEE Computer Society.

[21] A. Stoica, D. Keymeulen, A. Thakoor, T. Daud, G. Klimech, Y. Jin, R. Tawel, and V. Duong. Evolution of Analog Circuits on Field Programmable Transistor Arrays. In *Proc. of the 2000 NASA/DoD Conference on Evolvable Hardware*, pages 99–108, Palo Alta, CA, 2000. IEEE Computer Society.

[22] A. Stoica, D. Keymeulen, and R. Zebulum. Evolvable hardware solutions for extreme temperature electronics. In *The Third NASA/DoD workshop on Evolvable Hardware*, pages 93–97, Long Beach, CA, 2001. IEEE Computer Society.

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

[24] M. Wirthlin, E. Johnson, N. Rollins, M. Caffrey, and P. Graham. The Reliability of FPGA Circuit Designs in the Presence of Radiation Induced Configuration Upsets. In *Proc. of the 11th Symposium on FPGA-Based Custom Computing Machines FCCM'03*, pages 133–142. IEEE CS, 2003.

[25] R. Zebulum, D. Keymeulen, V. Duong, X. Guo, M. I. Ferguson, and A. Stoica. Experimental Results in Evolutionary Fault-Recovery for Field Programmable Analog Devices. In *Proc. of the 2003 NASA/DoD Conference on Evolvable Hardware*, pages 182–186, Chicago, USA, 2003. IEEE Computer Society.

[26] Y. Zhang, S. Smith, and A. Tyrrell. Digital Circuit Design Using Intrinsic Evolvable Hardware. In *Proc. of the 2004 NASA/DoD Conference on Evolvable Hardware*, pages 55–62, Seattle, USA, 2004. IEEE Computer Society.