On the feasibility of using evolvable hardware for hardware Trojan detection and prevention

Labafniya, Mansoureh; Picek, Stjepan; Etemadi Borujeni, Shahram; Mentens, Nele

DOI
10.1016/j.asoc.2020.106247

Publication date
2020

Document Version
Final published version

Published in
Applied Soft Computing Journal

Applied Soft Computing Journal 91 (2020) 106247

Contents lists available at ScienceDirect

Applied Soft Computing Journal

journal homepage: www.elsevier.com/locate/asoc

On the feasibility of using evolvable hardware for hardware Trojan detection and prevention

Mansoureh Labafniya [a], Stjepan Picek [b], Shahram Etemadi Borujeni [a,*], Nele Mentens [c]

[a] University of Isfahan, Iran
[b] TU Delft, The Netherlands
[c] ES&S and imec-COSIC/ESAT, KU Leuven, Belgium

ARTICLE INFO

Article history:
Received 2 June 2019
Received in revised form 2 January 2020
Accepted 11 March 2020
Available online 18 March 2020

Keywords:
Evolvable Hardware (EH)
Virtual Reconfigurable Circuit (VRC)
Hardware security
Hardware Trojan Horse (HTH)
Field-Programmable Gate Array (FPGA)

ABSTRACT

Evolvable hardware (EH) architectures are capable of changing their configuration and behavior dynamically based on inputs from the environment. In this paper, we investigate the feasibility of using EH to prevent Hardware Trojan Horses (HTHs) from being inserted, activated, or propagated in a digital electronic chip. HTHs are malicious hardware components that intend to leak secret information or cause malfunctioning at run-time in the chip in which they are integrated. We hypothesize that EH can detect internal circuit errors at run-time and reconfigure to a state in which the errors are no longer present. We implement a Virtual Reconfigurable Circuit (VRC) on a Field-Programmable Gate Array (FPGA) that autonomously and periodically reconfigures itself based on an Evolutionary Algorithm (EA). New VRC configurations are generated with an on-chip EA engine.

We show that the presented approach is applicable in a scenario in which (1) the HTH-critical areas in the circuit are known in advance, and (2) the VRC is a purely combinatorial circuit, as opposed to the on-chip memory holding the golden reference, which requires one or more cycles to be read/written. We compare two different approaches for protecting the system against HTHs: Genetic Programming (GP) and Cartesian Genetic Programming (CGP). The paper reports on experiments on four benchmark circuits and gives an overview of both the limitations and the added value of the presented approaches.

1. Introduction

With the rise of the Internet of Things (IoT), electronic chips or integrated circuits (ICs) play an increasingly important role in our society. Often, ICs process sensitive personal or company-critical information. Nevertheless, several steps in the development of ICs are outsourced to different (sometimes untrusted) parties. This opens the door for the manipulation of ICs to extract secret information, e.g., through wireless communication or disabling (parts of) the chip. A malicious building block that is inserted in an IC to cause this undesired behavior is called a Hardware Trojan Horse (HTH).

The potential threat of HTHs was first reported by the US Department of Defense in 2005 [1]. They expressed their concerns on ICs in military applications, mainly related to untrusted foundries and untrusted actors in the supply chain. Although there are no HTHs in ICs reported in real-world applications yet, there are many examples of academic research results, both on injecting and detecting/preventing HTHs. Moreover, there was recently the alarming disclosure of a tiny chip that was added to the motherboard of servers of the company Elemental Technologies [2]. The chip that was not part of the original design of the motherboard creates a secret connection to each network in which the server is included. The fact that Elemental's servers are massively deployed in US Defense data centers underlines the severity of the matter. Investigations showed that the chips were inserted by Chinese subcontractors during the manufacturing process.

In this paper, we investigate the possibility of using run-time reconfigurable circuits to prevent the insertion, the activation, and the propagation of HTHs. More specifically, we explore solutions based on evolvable hardware (EH) architectures, which are reconfigurable circuits that adapt their behavior dynamically through interactions with the environment. EH concentrates on the generation of efficient electronic circuits through the use of Evolutionary Algorithms (EAs). Originally, EH techniques were proposed for the efficient design of new circuits, i.e., to do a fast exploration of potential circuits with a given functionality at design-time [3]. In this case, the terms "evolutionary circuit design" and "evolved hardware" are also commonly used. Here, we consider the scenario in which the generation of new circuits

* Corresponding author.
E-mail addresses: mlabaf@eng.ui.ac.ir (M. Labafniya), s.picek@tudelft.nl (S. Picek), etemadi@eng.ui.ac.ir (S. Etemadi Borujeni), nele.mentens@kuleuven.be (N. Mentens).

https://doi.org/10.1016/j.asoc.2020.106247
1568-4946/

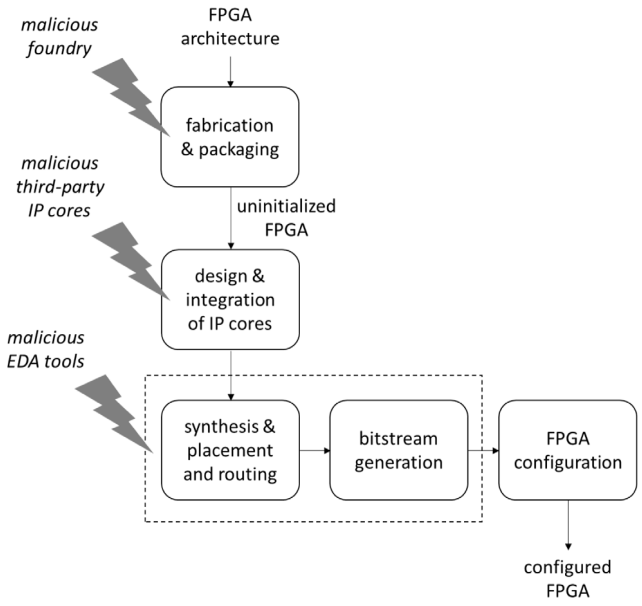


Fig. 1. FPGA design flow and potential insertion of HTHs.

is done at run-time, adapting to changes in the environment, as proposed in [4] for run-time filter updates in image processing applications.

A popular way of implementing EH architectures is through a Virtual Reconfigurable Circuit (VRC) [5]. This is a circuit that consists of programmable elements with a programmable interconnect. As opposed to commercially available configurable hardware platforms or Field-Programmable Gate Arrays (FPGAs), VRCs can be reconfigured in only one or a few clock cycles based on the direct output of an EA.

The contributions of this paper can be summarized as follows:

- We are the first to investigate the use of EH for HTH detection and prevention.
- We implement a VRC as a virtual overlay architecture on an FPGA for preventing HTH insertion, activation, and propagation.
- We compare two approaches for the generation of new configurations. The first is based on a tree structure using Genetic Programming (GP), while the second one is based on a graph structure using Cartesian Genetic Programming (CGP).
- We evaluate the overhead in FPGA resources, power consumption, and computational delay for four commonly used hardware circuits, implemented on the VRC architecture.
- We discuss the limitations of the proposed solution and the application scenarios in which the presented architectures are of interest.

The paper is structured as follows. Section 2 gives the necessary background information on HTHs in digital circuits, evolvable hardware, and VRCs. Section 3 discusses related work on HTH detection and prevention. In Section 4, we propose an FPGA architecture consisting of a VRC and an on-chip EA engine. In Section 5, we evaluate the feasibility of our approach based on four benchmark circuits. Section 6 concludes the paper and gives an outlook on future work.

2. Preliminaries

2.1. Hardware Trojan horses in digital circuits

HTHs are malicious circuits that intend to leak secret information, e.g., through wireless communication, or to cause malfunctioning at run-time in the chip in which they are integrated. HTHs can be inserted by untrusted foundries and actors at different stages in the design and development of FPGAs and ASICs (Application-Specific Integrated Circuits). For example, in a chip that needs to communicate over the Internet, the hardware component that takes care of Internet communication at the physical layer is typically bought from a third-party vendor. If this core contains built-in additional functionality that communicates sensitive data from within the chip to the outside world, the security of the whole chip is compromised. We call this additional functionality an HTH. An HTH consists of two parts: a trigger and a payload. The trigger usually corresponds to a rare data input (sequence), while the payload is the activity that causes the data leakage or the malfunctioning when the HTH is triggered. In the example of the third-party communication core, the chip will send out sensitive data to the outside world when a specific combination or sequence of input signals is applied.

HTHs are usually inserted in special places in the design that have low testability or high slack time [6–8]. Testability is measured through two parameters: controllability, which is determined by the effort needed to change an internal signal by controlling primary inputs, and observability, which is determined by the effort needed to observe the effect of an internal signal variation at the output of the chip. At points with low testability, HTHs are more difficult to trigger and more difficult to detect through a read-out of the output values. The slack time is the difference between the required arrival time of a signal and the actual arrival time. A high slack time leaves room for inserting logic without violating the required critical timing paths. Inserting HTHs at points in the design with low testability or high slack time minimizes the chance of HTH insertion to be detected. This means that we can predict which areas in the design are most likely to be chosen by attackers for HTH insertion. This justifies our claim that we can insert our protection mechanism in well-chosen relatively small parts of the chip.

Since the experiments presented in this paper are performed on FPGA platforms, Fig. 1 zooms into the different steps in the design and development of FPGA applications that are susceptible to HTH insertion. We distinguish the steps the FPGA goes through before it is deployed in an end application. The first place where HTHs can be inserted is in the foundry that fabricates and packages the FPGA chip. In this case, we assume that the foundry is malicious. It modifies the physical design, i.e., it adds an HTH, before fabrication and packaging. The insertion of an HTH in this step leads to an FPGA that contains an additional circuit that, e.g., has access to the internal signals of the benign FPGA logic and communicates this internal information through a wireless link or alters the internal signals to induce unwanted behavior of the FPGA. The design phase is the second step that suffers from a potential insertion of HTHs, in particular when third-party Intellectual Property (IP) cores are used. IP cores are pre-designed building blocks, e.g., for communication or security purposes, that are based on the IP of the third-party design house that offers/sells them. An HTH in an IP core uses part of the configurable logic of the FPGA in a malicious way to retrieve secret information or to cause malfunctioning of the FPGA. In this case, we assume that the vendor of the IP core is malicious. While the designer that buys the IP core trusts the third-party vendor to deliver an IP core with the requested functionality, the vendor actually sells an IP core that additionally contains an HTH.

After the design is ready, Electronic Design Automation (EDA) tools are used to perform synthesis, placement and routing, and to generate the bitstream that contains the configuration data. In this case, we assume that the EDA tool provider is malicious. The automated processes for synthesis, placement and routing, and bitstream generation, result in a bitstream that does not only invoke the requested configuration in the FPGA but also an additional configuration that holds an HTH. In the final phase, the actual configuration is performed by storing the bitstream in the FPGA's configuration memory.

Note, HTHs are mentioned as an important threat in an overview of FPGA security by Trimberger and Moore of the company Xilinx, which is one of the world-leading FPGA vendors [9]. Saar Drimer also gives a comprehensive overview of security threats on FPGA platforms in [10].

2.2. Evolutionary algorithms and evolvable hardware

Evolutionary algorithms (EAs) are population-based metaheuristic optimization techniques inspired by biological evolution [11]. Candidate solutions to the optimization problem are individuals in a population, and the fitness function determines the quality of the solutions. We show pseudocode for evolutionary algorithms in Algorithm 1.

Algorithm 1: Pseudocode for EA.

Input : *Parameters of the algorithm*
Output : *Optimal solution set*
$t \leftarrow 0$
$P(0) \leftarrow$ *CreateInitialPopulation*
while *TerminationCriterion* **do**
  $t \leftarrow t + 1$
  $P'(t) \leftarrow$ *SelectMechanism* $(P(t-1))$
  $P(t) \leftarrow$ *VariationOperators*$(P'(t))$
end while
Return OptimalSolutionSet(P)

2.3. Genetic programming

Genetic Programming (GP) is a type of EA in which the data structures that undergo an evolutionary process are executable computer programs [12]. GP has a history longer than 50 years, but its full acceptance comes from the work of Koza at the beginning of the 1990s, where he formalized the idea of employing chromosomes on the basis of tree data structures [13]. GP aims to automatically generate new programs, and each individual of a population represents a computer program [12] where the most common are symbolic expressions representing parse trees. A parse tree is an ordered, rooted tree that represents the syntactic structure of a string according to some context-free grammar. As it makes fewer assumptions about the structure of possible solutions, GP could be regarded as a more general form of genetic algorithms (GAs) [14]. Building elements in a tree-based GP are functions and terminals. Both functions and terminals are known as primitives.

2.4. Cartesian genetic programming

Julian Miller introduced a new form of GP in 1999 that represents programs as directed graphs instead of trees [15]. In CGP, a program is represented as an indexed graph. The graph is encoded in the form of a linear string of integers. Terminal set (inputs) and node outputs are numbered sequentially. Node functions are also numbered separately [16]. CGP has three parameters that are chosen by the user number of rows n_r, number of columns n_c, and levels-back l. The number of rows and the number of columns make the two-dimensional grid of computational nodes. Their product gives the maximal number of computational nodes. The levels-back parameter controls the connectivity of the graph, i.e., it determines which columns a node can get its input from. The genotype is a list of integers that represents the program primitives and how they are connected. The genotype is mapped to the directed graph that is executed as a program. Genotypes are of fixed length, while phenotypes have a variable length in accordance with the number of unexpressed genes. Those nodes that constitute the phenotype are called the active nodes. CGP has been shown to be more computationally efficient on a number of problems [17].

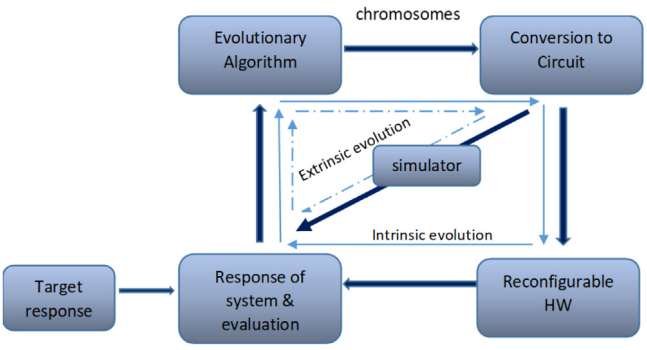


Fig. 2. Extrinsic and intrinsic methods for evolutionary algorithms.

2.5. Evolution in hardware

EAs can be implemented both in software and hardware. The software implementation of EAs is easy but has unacceptable delays that are not desirable for most electronic systems. In contrast, implementing EAs in hardware, especially with parallel computing platforms like FPGAs, is more efficient. Because of the Dynamic Partial Reconfiguration (DPR) capabilities of many FPGAs, they are proper hardware platforms for implementing EAs [18]. The DPR feature of FPGAs can be used in different application domains like the Internet of Things (IoT), image processing, and arithmetic circuits. For example, [19] describes how IoT networks can take advantage of the DPR features of FPGAs in a secure way.

Evolution in hardware can be classified into three categories: extrinsic implementations, intrinsic implementations, and VRCs. In extrinsic implementations, EAs are implemented on an external computing device, and the fitness of all chromosomes is evaluated by software models and simulators. Only the best option will be reconfigured on the FPGA as the desired circuit. In intrinsic implementations, EAs are implemented on an external computing device, but the evaluation of each chromosome is done on the FPGA through DPR. The relatively large delay of DPR makes both the intrinsic and extrinsic methods time-consuming on an FPGA. Fig. 2 shows the structure of intrinsic and extrinsic EAs [20]. The extrinsic and intrinsic methods are depicted in Fig. 2, while the third category, VRCs, is explained in the next paragraph.

2.6. Virtual reconfigurable circuits

The third model for hardware evolution is implemented completely on FPGA and is called Virtual Reconfigurable Circuits (VRC). This model was introduced for evolvable digital hardware as a way to rapidly reconfigure platforms using conventional FPGAs. A VRC consists of an array of programmable elements (PEs) with programmable interconnect. A VRC can be seen as

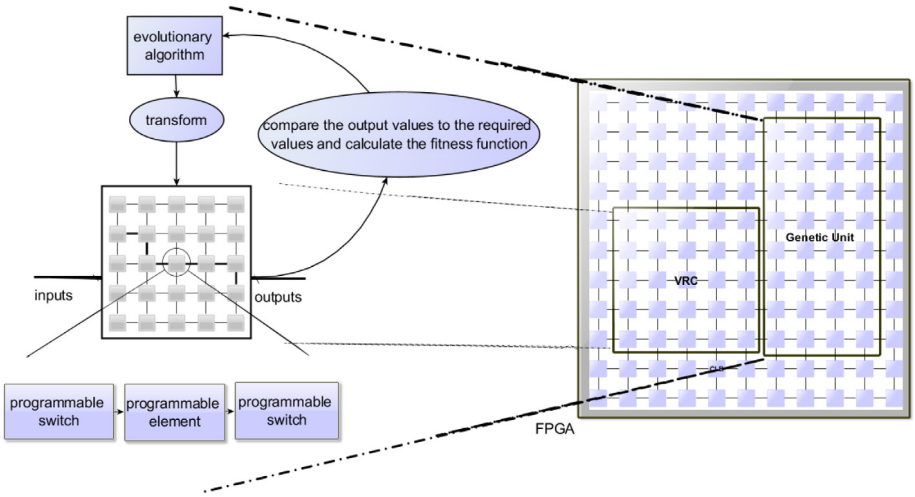


Fig. 3. Example of a VRC, configured by a genetic unit.

a structure that is configured by a second reconfiguration layer (on top of the built-in reconfiguration layer that is part of any FPGA), which is configured by an EA [5]. Fig. 3 shows a VRC structure which consists of 25 Programming Elements (PEs), connected through programmable switches. The generation of new configurations is based on CGP [16]. A genetic unit generates new configurations that are evaluated for correctness on the VRC. The flexibility and heuristic approach of the VRC leads to a real-time application of the EA [18]. In comparison to extrinsic and intrinsic implementations, VRCs provide a much faster reconfiguration of the FPGA.

In our work, the golden-reference functionality is stored in an embedded memory block on the FPGA. We assume that it is much more difficult to apply HTHs in these memory blocks than in the FPGA's configurable logic. This assumption is motivated by the fact that most memories have built-in error detection/correction mechanisms, which is also the case for the Block RAM (BRAM) modules that are used in our implementation [21]. Moreover, the delay and resource constraints on the BRAM modules are much more relaxed than on the configurable logic, because the BRAM memory is accessed only during the golden-reference comparison, and because the behavior of only a small circuit is stored in BRAM. Consequently, additional methods that cause an overhead in delay and resources, like the widely used Triple-Modular Redundancy (TMR) technique [22], can easily be applied to the memory modules without violating delay or resource constraints.

3. Related work

3.1. HTH prevention and detection

To obtain secure systems, methodologies that incorporate Design for Security (DfS) should be used. Both prevention and detection methods are necessary to prevent adversaries from infecting a system with HTHs.

In the category of prevention techniques, a new TMR structure is introduced in [23], called Adapted Triple Modular Redundancy (ATMR). ATMR uses three different circuits for implementing the same module, relying on the fact that it is highly unlikely that the circuits are all triggered simultaneously by HTH activation. Both the conventional TMR approach and the proposed ATMR have high area overhead and power consumption due to the redundant structure. Filling unused space on an FPGA is another way to protect a circuit from HTH insertion [24]. This method has a minimized performance and power penalty. Using physical and logical keys in [25] improves the security of an FPGA system by obfuscating the FPGA bitstream; the technique is based on a dedicated configurable architecture. Preventing HTH insertion by CAD tools is done in [26], where moving FPGA-Oriented Moving Target Defense (FOMTD) methods are proposed based on three defense lines. The FOMTD method generates uncertainties for attackers to make it harder to insert HTHs.

Techniques for the detection of HTH insertion can be categorized into two classes: physical analysis and functional analysis:

- Physical analysis performs an inspection of the physical properties of the chip. This can be done destructively by decapsulation and delayering, after which the identification of the HTHs takes place through visual layout inspection [27,28]. A non-destructive way of performing the physical inspection is through the analysis of side-channels, such as the power consumption or the electromagnetic emanation of the chip [29]. In [30], the FPGA is divided into different zones based on trustworthiness, calculated by the frequency response of Ring Oscillators (ROs) in different positions in the FPGA.
- The goal of functional analysis is to find rare trigger signals or sequences that activate the HTH payload. This can be done externally by testing the input–output behavior of the chip. Functional analysis of the internal building blocks of the chip is enabled through built-in self-test (BIST) modules, under the assumption that the BIST modules do not contain HTHs [31].

3.2. Evolutionary algorithms for improving electronic circuits

Examples of successful applications of evolutionary algorithms span diverse domains, e.g., image processing algorithms [32], the design of fault tolerant systems [18], face recognition [33], power consumption optimization [34], and arithmetic circuit design [35]. In [33], a method is proposed to improve the security of hardware circuits based on EAs by decreasing the number of rare signals with low testability and increasing the efficiency of logic encryption. Logic encryption is a method that allows only authorized users to activate a hardware circuit.

Another type of evolutionary algorithms, genetic algorithms, have been successfully applied in various digital circuit implementations [36–38]. In [39], parallel GAs are used for the design of digital circuits for FPGA-based architectures. The GA employed involves the use of a linear representation which can be readily employed for intrinsic evolution systems such as through the

direct manipulation of the FPGA configuration bitstream. In [36], the logic circuit is organized on a two-dimensional array of cells using GAs to have an optimal circuit design in terms of circuit complexity, power, and time delay.

Other works concentrate on designing digital combinational circuits using CGP [37,40–43]. In [44], the authors show that if a CGP-based evolutionary system can produce solutions for digital design such that the selected parameters for CGP are close to human-made solutions, the produced results will be better. In [45], the authors try to minimize the digital circuit size using CGP. They conclude that a bigger CGP dimension results in a higher success rate to reach the intended truth table while scarifying space and time for searching through the chromosome space. Using CGP to self-reconfigure digital circuits is proposed in [46] and shown on a full adder FPGA design.

4. Proposed HTH prevention/detection method

In this paper, a new method to protect FPGAs from the insertion of HTHs is presented. Our approach uses virtual partial FPGA reconfiguration and implements EAs entirely on the FPGA. We assume that HTHs are inserted in specific subcircuits of the digital architecture, and, therefore, we concentrate on protecting those subcircuits. The criteria for selecting the subcircuits are based on testability and time slack, as explained in Section 2.1, or on the type of building block with respect to the sensitivity of the processed data, e.g., cryptographic algorithms might be interesting components for an attacker to insert an HTH. Our method consists of reconfiguring the selected subcircuit periodically. Consequently, even if an adversary inserts any HTHs, reconfiguration will modify the circuit such that, in spite of the presence of HTHs, the functionality of the design will be corrected.

Fig. 4 shows our proposed architecture for securing FPGAs against HTH insertion by using a CGP structure. There are two important sections: the main design and the genetic unit. The FPGA contains the main digital design, of which an important subcircuit is implemented in the VRC. This subcircuit is carefully selected, as explained in the previous paragraph. The number of inputs and outputs of the main design is different for each benchmark circuit. This number does not change when we secure the main design. Only if we use an external LFSR to produce chromosomes, an extra signal will be added to the main design according to the length of each chromosome. Securing the design means replacing one or more subcircuits by a VRC, which has the same number of functional inputs and outputs as the original subcircuit. Each VRC also has an extra control signal that comes from the Genetic Unit to reconfigure the PEs. The genetic unit is controlled by an enable signal and a clock input, which comes from the on-board or on-chip system clock with a fixed period. Based on the latency and the idle time of the main design, the genetic unit will be enabled periodically or at random instances in time. Whenever the process is activated, an internal Linear Feedback Shift Register (LFSR) produces a chromosome based on a random seed at the start of each clock cycle. The produced chromosome is evaluated, i.e., the VRC is reconfigured, the output values of the VRC are compared to the required values, and the fitness of the chromosome is calculated. The fitness function is based on a truth table that reflects the required functionality of the subcircuit implemented by the VRC. The truth table is stored in an embedded memory on the FPGA that is assumed to be HTH-free. The reason why the truth table is not used as a functional unit in the main design is that it is much slower than the logic implemented in the VRC. An optimal chromosome corresponds to a configuration of the VRC that adheres to the required truth table. Eqs. (1) and (2) show the fitness function that we use for evolving the VRC. $F(x)$ is the output of the truth table when the input is equal to x, and $S(x)$ is the output of the VRC when the input is equal to x. x is the number of different possible inputs in the intended truth table. For example, if the truth table has three inputs, x ranges from 0 to 7.

$$G(x) = \begin{cases} 1 & S(x) == F(x) \\ 0 & S(x)! = F(x). \end{cases} \tag{1}$$

$$Fitness = \sum_{x=0}^{2^{(\text{number of inputs in truth table})}-1} G(x). \tag{2}$$

The VRC reconfiguration consists of reconfiguring the functionality of the processing elements (PEs) and the programmable switches. Fig. 4 shows, for example, four PEs in a VRC. The number of PEs is chosen based on the application. If the circuit is complex and/or large, the EA module will not converge with a small number of PEs, so a large number must be selected. But for small and simple circuits, a small amount of PEs suffice. We start designing the circuit using the smallest size $2 * 2$, and then if it does not converge, we gradually move towards larger VRCs until the system converges to the desired output. It is important to notice that the evaluation of chromosomes is done according to the current situation of the unsecured section. If any HTH is inserted in the circuit, the evaluation is done in spite of the presence of HTHs, and the system will repair itself in the existence of HTHs. We use both CGP and GP to implement our solution and compare both methods.

In the method presented in Fig. 4, we depict CGP operation. One Linear Feedback Shift Register (LFSR) produces the initial population in each run. After producing four children by mutation from the parent, the evaluation on the VRC is done. In this phase, four chromosomes are evaluated according to a truth table that corresponds to the expected behavior of the circuit. For each input combination, the outputs of the candidate individual are evaluated and compared with the originally required outputs, as described in the truth table of the digital function to be implemented. Bitwise comparison is done in this case, incrementing the fitness value with each output line match. This is accumulated over all possible input combinations. In our case, the selected function has three inputs, which produce eight different outputs, so the maximum fitness is equal to eight.

In the last module in Fig. 4, we select the chromosome with the highest fitness value. All the other individuals are discarded. If the fitness function with the value 8 is acquired by the best chromosome, the algorithm finishes, and the VRC keeps its current configuration. Otherwise, the algorithm must continue with the mutation mechanism to produce four new children from the parent. Table 1 contains the fixed parameters for implementing and comparing the GP and CGP structures. We use a "3-tournament" selection mechanism for GP (where the worst from the 3 randomly selected individuals is eliminated) and $(1+4)$ evolution for CGP (where the offspring are favored over parent when they have a fitness better than or equal to the fitness of the parent). Other parameters for CGP and GP are presented later as we investigate various configurations. Finally, we use node replacement mutation and subtree crossover.

Note, for reducing the hardware overhead, the number of chromosomes in the population must be small. A larger population size consumes more FPGA resources for saving and processing the chromosomes. The maximum number of generations for each population is 50. This number is determined by exploration through simulation: optimal chromosomes are calculated before the 50th generation. If, after 50 generations from the initial population, the optimum individual is not found, the new population is produced randomly, and again 50 generations will be produced. The termination condition is finding an optimal individual or

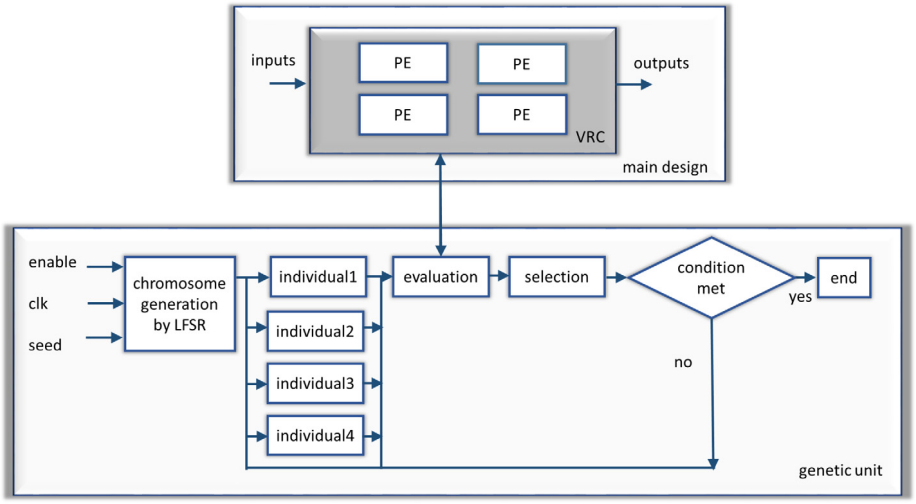


Fig. 4. Proposed structure for protection of FPGA against HTHs.

Table 1
Fixed parameters for the CGP and GP approach.

Parameters	CGP	GP
Selection mechanism	$1+\lambda = 1+4$	3-tournament
Population size	$1+4$	5
Max generations	50	50
Termination condition	500	500

reaching the 500th run for producing a new population. For synthesis, we neglect the overhead of the LFSR for producing random chromosomes. Note that the LFSR can also be replaced by a Pseudo Random Number Generator (PRNG) or a True Random Number Generator (TRNG). Each PE in the VRC has two inputs and one output. The function set is AND, OR, NAND, NOR, NOT, BUFFER (i.e., pass-through), XOR, and XNOR.

An important feature of the proposed protection method is that the system will not fail or denial service after the detection of an HTH. It can repair itself and stay alive. This is in contrast to the work in [47] that presents an FPGA solution that shuts down the system when an HTH is detected.

The structure of each chromosome depends on the number of PEs in the VRC. In the following section, the proposed scheme is described in more details, and simulation results are given.

5. Evaluation results

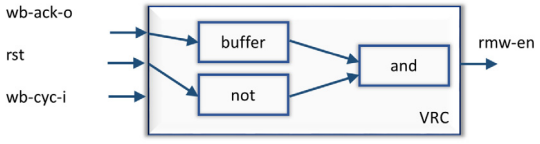
This section describes the details of the proposed scheme in this paper through the implementation of four benchmark circuits. The first one is *Mem-ctrl*, a memory controller from the IWLS benchmark [48] because a desirable place for an attacker to insert an HTH is at the control input of a memory. On this benchmark circuit, we perform an investigation on the use of CGP and GP for HTH protection. We also evaluate the resource occupation and the power consumption of three other benchmark circuits: *AES-core*, an encryption core, *AC97-ctrl*, an audio codec controller, and *Ethernet*, an Ethernet communication core.

5.1. The Mem-ctrl benchmark circuit and the expected behavior of the HTH protection mechanism

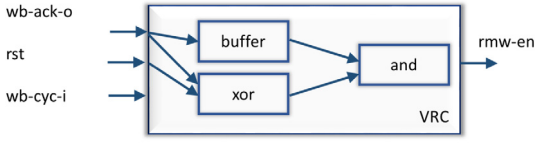
The behavioral code for the *Mem-ctrl* benchmark circuit is shown in Fig. 5a. The corresponding truth table that is evaluated by the genetic unit is represented in Table 2. After activating the enable signal in Fig. 4, the genetic unit reconfigures the VRC until

```
if(rst)                    rmw_en <= #1 1'b0;
else
if(wb_ack_o)               rmw_en <= #1 1'b1;
else
if(!wb_cyc_i)              rmw_en <= #1 1'b0;
```

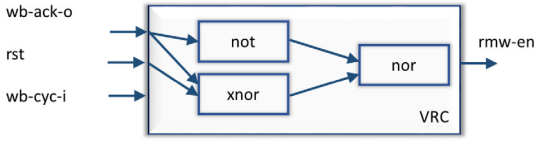
(a)



(b)



(c)



(d)

Fig. 5. Behavioral code to activate the "rmw-en" signal (a), and different chromosomes implemented on the VRC that generate the correct output behavior (b, c, and d).

a correct output for "rmw-en" is observed for all input values. Examples of three chromosomes that function exactly like the behavioral code in Fig. 5a and the truth table in Table 2 are presented in Figs. 5b, 5c, and 5d. As the output of the truth table in Table 2 does not depend on "wb-cyc-i", a simplified form that omits the "wb-cyc-i" input is used in Figs. 5b, 5c and 5d. For different truth tables, all inputs might be used.

Examples of circuits that are infected by an HTH and corrected through VRC reconfiguration are shown in Figs. 6 and 7. Fig. 6b

Table 2
Truth table that produces "rmw-en" based on "rst", "wb-ack-o", and "wb-cyc-i" in the *Mem-ctrl* benchmark.

Inputs			Output
rst	wb-ack-o	wb-cyc-i	rmw-en
0	0	0	0
0	0	1	Do not care
0	1	0	1
0	1	1	1
1	0	0	0
1	0	1	0
1	1	0	0
1	1	1	0

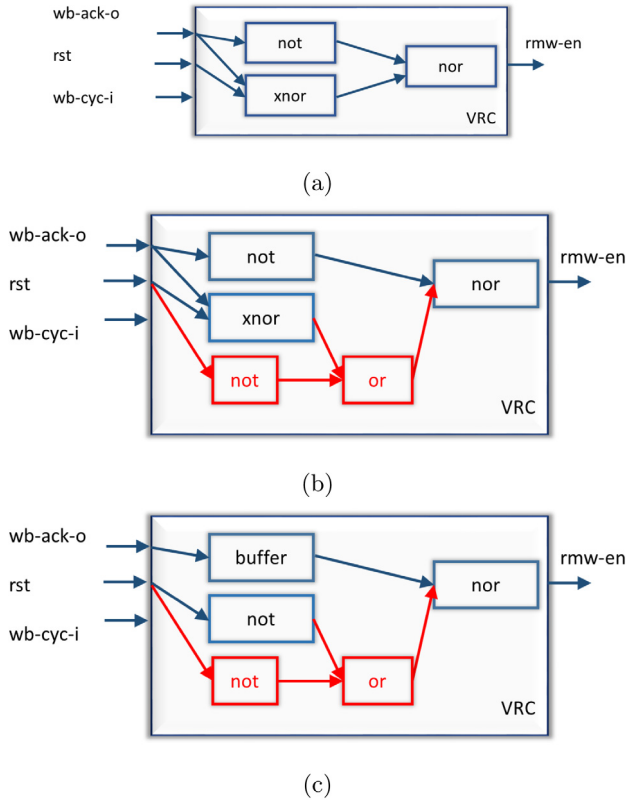


Fig. 6. (a) original circuit; (b) HTH inserted in the circuit; (c) EA-based reconfiguration to correct the output.

shows an infected version of a circuit from Fig. 6a that forces the output to be zero for inputs ("rst", "wb-ack-o", "wb-cyc-i") = (0, 1, 1) and ("rs", "wb-ack-o", "wb-cyc-i") = (0, 1, 0). After the execution of the EA and the reconfiguration of the VRC, the functionality of the circuit is corrected, as shown in Fig. 6c. The HTH in Fig. 6 affects the interconnection of the VRC, while the HTH in Fig. 7 only modifies the functionality of a PE. Fig. 7b shows an infected version of a circuit from Fig. 7a. The HTH forces the output to be 1 for the inputs ("rst", "wb-ack-o", "wb-cyc-i") = (1, 1, 1) and ("rst", "wb-ack-o", "wb-cyc-i") = (1, 1, 0). The EA evolves the circuit to repair itself and make an attack unsuccessful. Fig. 7c shows the repaired circuit.

5.2. CGP versus GP on the Mem-ctrl circuit

We evaluate both tree- and graph-based structures in different dimensions with the parameters given in Table 1. The implementation results are generated using the Vivado 2018.2 EDA tool (64-bit version) for the xc7vx485tffg1157-1 device, which

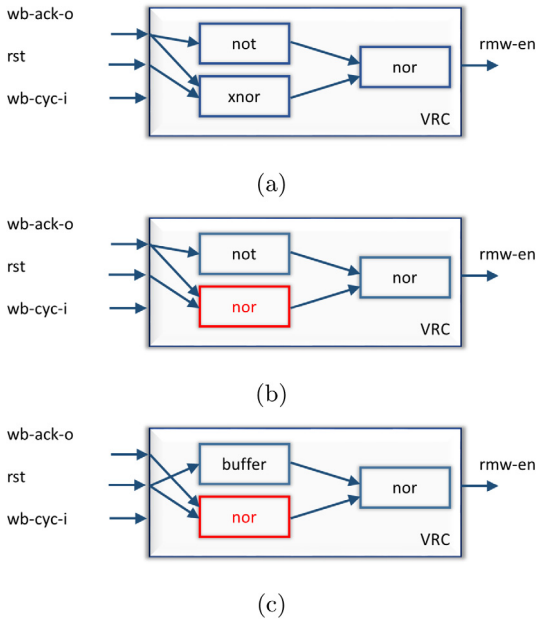


Fig. 7. (a) original circuit; (b) HTH inserted in the circuit; (c) EA-based reconfiguration to correct the output.

is a Virtex-7 FPGA of Xilinx. It has 600 IOBs, 303600 LUTs, and 2800 DSPs. Four LUTs and eight flip-flops form a slice, and two slices form a configurable logic block (CLB). We use Verilog as a hardware description language for all the proposed designs.

Tables 3 and 4 show the implementation results for the *Mem-ctrl* benchmark circuit based on the tree-based approach. Tables 5 and 6 show the results for the graph-based approach. In total, eight strategies are evaluated, called **Strat0** to **Strat7**. All strategies have same number of inputs, $n_i = 3$, and outputs, $n_0 = 1$. The depth of the tree structure in the GP approach is varied from $\#levels = 2$ to 4 in Table 3. Although the logic circuit is assumed to be organized in a two-dimensional array of cells, most configurations do not use at least some of the cells. The dimension of the array is equal to $\#levels = n_r = n_c$ for each GP strategy in Table 3. In the CGP approach based on a graph structure in Table 5, the number of feedback levels is varied from $\#fb = 1$ to 4 and the dimension is varied between $n_r * n_c = 2 * 2$ and $4 * 4$. Both Tables 3 and 5 indicate the length of the chromosome (*len*) and when the first optimal chromosome is found among all 500 populations (*#pop*) and 50 generations (*#gen*). The number of PEs used by the first found optimal chromosome is indicated as well (*#PE*). The resource consumption of the applied strategies on FPGA, expressed in number of Look-Up Tables (LUTs), is indicated in both tables in the *#LUT* column. The *#opt* parameter indicates the total number of configuration options that are possible through the applied strategy.

Tables 4 and 6 compare three different parameters for the node replacement mutation operator that is applied 2 or 3 times, respectively. The first parameter in the tables is the calculated mutation rate (*mutrate*), which is equal to the percentage of bits that are mutated in the chromosome. The second parameter in the tables is the average fitness value (*avgfit*) of all produced chromosomes, which is below 8. The success rate (*sucrate*) indicates the percentage of all optimal chromosomes with fitness value equal to 8 after 500 runs.

In **Strat0** to **Strat2**, the output of the VRC can only be connected to the last PEs. Each PE can get its inputs from each PEs in the previous columns. These three strategies have a tree structure of which the depth is indicated by *#levels*, and the number of

Table 3
Explored parameters for the implementation of *Mem-ctrl* based on a tree structure.

GP	n_i	n_o	*#levels*	*len*	First optimal chrom			*#LUT*	*#opt*
					#pop	*#gen*	*#PE*		
Strat0	3	1	2	28	5	27	3	126	8^3
Strat1	3	1	3	65	11	49	5	313	8^4
Strat2	3	1	4	114	24	5	7	2548	8^{15}

Table 4
Effect of the mutation rate on the effectiveness of the *Mem-ctrl* benchmark circuit based on a tree structure.

GP	2-node replacement mutation			3-node replacement mutation		
	mutrate	*avgfit*	*sucrate*	*mutrate*	*avgfit*	*sucrate*
Strat0	7.1	6.5	4	10.7	7	5.2
Strat1	3	6.9	8.6	4.6	7	8.8
Strat2	1.7	6.9	9.2	2.6	7.06	13

Table 6
Effect of the mutation rate on the effectiveness of the *Mem-ctrl* benchmark circuit based on a graph structure.

GP	2-node replacement mutation			3-node replacement mutation		
	mutrate	*avgfit*	*sucrate*	*mutrate*	*avgfit*	*sucrate*
Strat3	5.7	5	0.6	8.5	5.5	0.8
Strat4	1.9	6.4	1.2	2.9	6.7	4
Strat5	1.3	6.5	2.6	2	6.8	5.2
Strat6	1.1	6.7	3.4	1.7	6.8	4.6
Strat7	1.1	6.7	3.4	1.7	6.8	5.2

levels back is zero. The VRC of **Strat0** is shown in Fig. 8. It consists of a tree structure with three PEs. The component generates the "rmw-en" signal, one of the memory control signals, based on three inputs: "Rst", "Wb-ack-o", and "Wb-cyc-i". Each PE has two inputs and one output. In addition to three inputs/outputs for each PE, three bits are necessary to define the functionality (i.e., to select one of the eight functions listed in Section 4) for the VRC to adhere to the truth table of the desired circuit. Moreover, two configuration bits are needed for each input to determine the connectivity. This leads to seven configuration bits for each PE. As a result, for 4 PEs, the length of the chromosome is equal to 28 bits.

In **Strat3** to **Strat7**, the output of the VRC can be connected to each of the PEs in all rows and columns. In these strategies, the PEs can get their inputs from other PEs on their left side or from the primary inputs based on the number of levels back (*#fb*). In **Strat4** and **Strat7**, which have full feedback, each PE can get its inputs from all PEs on its left side or primary inputs. In **Strat5**, each PE can get its input from the PEs in the first column on its left or from the primary inputs. In **Strat6**, each PE can get its input from the PEs in the first and second column on its left or from the primary inputs.

These different strategies result in a different level of security. Enabling more variety will cause a higher level of security, but will come at the cost of a higher energy and resource consumption. According to the evaluated results, **Strat0** is the best solution with respect to FPGA resource consumption. However, in terms of effectiveness, it is not the best solution, since it can generate only 8^3 different VRC configurations, which limits the HTH recovery capabilities of the circuit. **Strat5** offers the best trade-off with 8^{16} possible VRC configurations, but a high FPGA resource consumption of 4016 LUTs is not desirable. Moreover, increasing resource consumption also increases the dynamic power consumption of the chip. The implementation results of both **Strat0** and **Strat5** are shown in Table 7. In general, the implementation results of the explored strategies show that increasing dimensions of the VRC, improve the success rate and the average fitness value. Larger dimensions also lead to larger chromosome length, which results in a higher number of LUTs. Moreover, an increased mutation rate leads to a higher success rate and average fitness value. The comparison between GP and CGP shows that the VRC circuits implemented with CGP have a higher success rate and average fitness value at the expense of a larger number of LUTs.

To show the added value of the GP and CGP approach over a method that randomly searches the solution space, we performed a random search for both the tree and the graph structure. Because the tree structure is simple and the outputs are independent of each other, it was possible to find optimal solutions through a random search, but it took more clock cycles in comparison to the GP approach. For the graph structure, the random search did not find any optimal solution after searching 30 000 options. Therefore, we conclude that GP and CGP are more efficient and effective for the considered use case.

5.3. Attack analysis

The proposed VRC structure can protect against HTH insertion by changing the circuit to a new configuration that is found at run-time through the use of EAs. The approach prevents the propagation of the effect of an HTH by repairing the circuit at fixed time intervals or whenever the subcircuit is idle. Since the proposed method evaluates the functionality of new configurations on the fly and directly on the configurable hardware, any malfunctioning due to HTH insertion will be repaired. That means that we protect against:

- HTHs inserted during FPGA fabrication: this type of HTH introduces a permanent change in the FPGA fabric. Nevertheless, the introduced "error" will only have an effect on the functional behavior of the protected subcircuit for a subset of possible configurations. Our proposed technique will dynamically find the configurations that lead to the correct functionality, canceling out the effect of the HTH. This means that the permanent HTH will still be there in the FPGA fabric, but the effect of the HTH will not be noticed.

Table 5
Explored parameters for the implementation of *Mem-ctrl* based on a graph structure.

CGP	n_i	n_o	$n_r * n_c$	*#fb*	*len*	First optimal chrom			*#LUT*	*#opt*
						#pop	*#gen*	*#PE*		
Strat3	3	1	2*2	2	35	2	32	3	141	8^4
Strat4	3	1	3*3	3	103	34	26	2	459	8^9
Strat5	3	1	4*4	1	148	3	9	4	4016	8^{16}
Strat6	3	1	4*4	2	175	2	16	5	5582	8^{16}
Strat7	3	1	4*4	4	175	3	36	6	5900	8^{16}

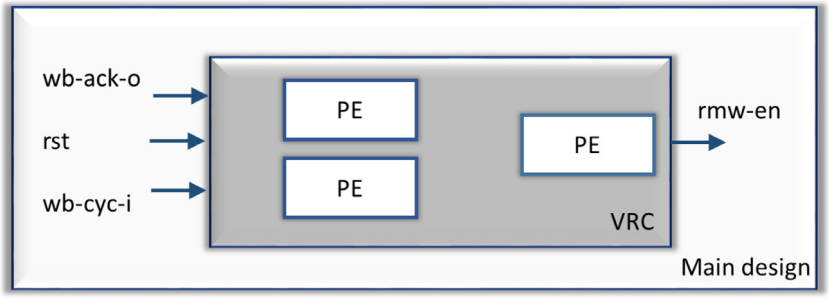


Fig. 8. Schematic of the tree structure used to implement the *Mem-ctrl* benchmark.

Table 7
FPGA resource occupation of the four implemented benchmark circuits based on **Strat0** and **Strat5**.

Bench-marks	Resource occupation (LUTs + FFs)			Power consumption (static + dynamic)		
	Without EA	With EA (strat0)	With EA (strat5)	Without EA	With EA (strat0)	With EA (strat5)
Mem-ctrl	1041 + 990	1167 + 990	5057 + 990	0.2 + 0.03	0.2 + 0.04	0.2 + 0.09
AES-core	627 + 406	753 + 406	4643 + 406	0.2 + 0.11	0.2 + 0.14	0.2 + 0.23
AC97-ctrl	939 + 1032	1065 + 1032	4955 + 1032	0.2 + 0.02	0.2 + 0.05	0.2 + 0.07
Ethernet	1949 + 2348	2075 + 2348	5965 + 2348	0.2 + 0.03	0.2 + 0.06	0.2 + 0.08

- HTHs inserted in the EDA flow (during synthesis and placement & routing, or during bitstream generation): this type of HTH results in a modified bitstream that ends up in the configuration memory. Because we dynamically reconfigure the content of the configuration memory, we make the HTH disappear after reconfiguration.

The shortcomings of our approach can be summarized as follows:

- Since we only repair the circuit periodically, an attacker can aim at temporarily inserting malicious functionality into the FPGA during the interval in which no repairing is done. Nevertheless, we suggest to always reconfigure in the idle time of the circuit, such that we only leave time for an attacker to insert an HTH while the circuit is active. But reconfiguring the circuit while it is active is not possible, leaving almost no possibility for an attacker to leave the insertion of HTHs undetected.
- The subcircuits that are replaced by VRCs are relatively small, which leaves room for an attacker to focus on the other parts of the FPGA. Nevertheless, the VRCs are well-positioned in places where HTHs are most likely to be inserted, i.e. at points with low testability or high time slack, or in components that process sensitive data, like cryptographic cores. This way, we maximize the effectiveness of our approach.
- We rely on a golden reference to evaluate the functionality of the VRC configuration. An attacker can undermine the security of the proposed system by tampering with the golden reference. Nevertheless, we assume that the golden reference is, e.g., stored in an embedded memory block. The likeliness that a memory block contains well-chosen malicious functionality is very low. Thus, it is reasonable to assume that the memory is free from HTHs. The added value of implementing the required functionality in the configurable logic of the FPGA over just implementing it in embedded memory is the lower latency that can be achieved with FPGA logic; it takes one or more cycles to read/write from/to embedded memory.

5.4. Resource and power consumption

Table 7 shows the FPGA resources and the power consumption of the design without the proposed protection mechanism and with the proposed protection mechanism, including the Genetic Unit. The timing constraint used for all the examples is a clock period of 10 ns. One of the control signals in the *Mem-ctrl* code is driven by a VRC. The other implemented IWLS benchmarks are also shown in Table 7. In the considered sample codes, one of the full adders is implemented by two different VRCs, one following **Strat0** and another one following **Strat5**. It must also be noticed that we can use one EA module for securing different subcircuits with the same truth table. They can be reconfigured simultaneously without any additional hardware overhead.

We are using the same number of PEs for all benchmarks. During reconfiguration, the functionality of the PEs (different logic gates or a simple pass-through) and the connections between the layers are changed, but the overhead with respect to resource occupation is the same for all benchmarks: 126 LUTs for **Strat0** and 4016 LUTs for **Strat5**. As indicated in Table 7, the EA, which consists of the Genetic Unit and the VRC, only uses LUTs. The Genetic Unit, excluding the LFSR, is a combinational circuit, without any clock input. The VRC is a combinational circuit as well because we use it to replace a combinational subcircuit in the main design, not a sequential one. In fact, we secure a combinational part of the main design. As a result, the EA does not consume any Flip-Flops. The implementation results in Table 7 shows that the smallest VRC structure (**Strat0**) that converges to the predefined truth table is the best choice in terms of low resources and low dynamic power consumption while offering an acceptable level of security as indicated in Table 3.

The operating frequency for producing random chromosomes is 83 MHz and is determined by the critical path of the EA module, which is a combinational circuit when **Strat0** is used. The hardware overhead of our proposed method by using **Strat0** is significantly lower than the overhead in work presented in [23], which also secures a portion of the circuits, just like our mechanism. The results reported in [23] lead to an increase of factor 5, while our work shows an increase in hardware overhead of less than 50%. The overall power overhead in [23] is doubled, while the power overhead of our method shows an increase of less than 50%. Note that we did not take into account the hardware overhead of the LFSR for the generation of the chromosomes.

6. Conclusions

In this paper, we propose a new protection mechanism against Hardware Trojan Horse (HTH) insertion. The proposed method is based on a Virtual Reconfigurable Circuit (VRC), implemented on a Field-Programmable Gate Array (FPGA). The VRC is reconfigured with an on-chip genetic unit that implements an Evolutionary Algorithm (EA). The VRC is implemented in selected parts of the FPGA that are susceptible to HTH insertion. Through periodic reconfiguration into valid alternative configurations, the VRC automatically recovers from HTH insertion. As this is the first work proposing the use of VRCs for HTH protection, we explore a tree-based approach (GP) as well as a graph-based approach (CGP) with different parameter sets. We evaluate the implementation results of four representative benchmark circuits and conclude that our approach offers a valid solution for effective HTH protection with an acceptable overhead in FPGA resource occupation.

Declaration of competing interest

The authors declare that they have no known competing financial interests or personal relationships that could have appeared to influence the work reported in this paper.

CRediT authorship contribution statement

Mansoureh Labafniya: Software, Writing - original draft, Conceptualization, Methodology. **Stjepan Picek:** Methodology, Writing - review & editing, Validation. **Shahram Etemadi Borujeni:** Writing - review & editing, Validation. **Nele Mentens:** Methodology, Project administration, Supervision, Writing - review & editing, Validation.

References

[1] United States. Defense Science Board. Task Force on High Performance Microchip Supply, Defense Science Board Task Force on High Performance Microchip Supply, Office of the Under Secretary of Defense For Acquisition, Technology, and Logistics, 2005.

[2] J. Robertson, M. Riley, The big hack: How China used a tiny chip to infiltrate US companies, Bloomberg Businessweek (2018).

[3] A. Thompson, An evolved circuit, intrinsic in silicon, entwined with physics, in: International Conference on Evolvable Systems, Springer, 1996, pp. 390–405.

[4] R. Salvador, A. Otero, J. Mora, E. de la Torre, T. Riesgo, L. Sekanina, Self-reconfigurable evolvable hardware system for adaptive image processing, IEEE Trans. Comput. 62 (8) (2013) 1481–1493.

[5] L. Sekanina, Š. Friedl, An evolvable combinational unit for FPGAs, Comput. Inform. 23 (5–6) (2012) 461–486.

[6] J. Rajendran, H. Zhang, C. Zhang, G.S. Rose, Y. Pino, O. Sinanoglu, R. Karri, Fault analysis-based logic encryption, IEEE Trans. Comput. 64 (2) (2015) 410–424.

[7] M.S. Samimi, E. Aerabi, Z. Kazemi, M. Fazeli, A. Patooghy, Hardware enlightening: No where to hide your hardware trojans!, in: 2016 IEEE 22nd International Symposium on on-Line Testing and Robust System Design (IOLTS), IEEE, 2016, pp. 251–256.

[8] S. Dupuis, P.-S. Ba, G. Di Natale, M.-L. Flottes, B. Rouzeyre, A novel hardware logic encryption technique for thwarting illegal overproduction and hardware trojans, in: 2014 IEEE 20th International on-Line Testing Symposium (IOLTS), IEEE, 2014, pp. 49–54.

[9] S. Trimberger, Trusted design in FPGAs, in: Proceedings of the 44th Annual Design Automation Conference, in: DAC '07, ACM, New York, NY, USA, 2007, pp. 5–8, http://dx.doi.org/10.1145/1278480.1278483.

[10] S. Drimer, Volatile FPGA design security–a survey, IEEE Comput. Soc. Annu. Vol. (2008) 292–297.

[11] A.E. Eiben, J.E. Smith, Introduction to Evolutionary Computing, Springer-Verlag, Berlin Heidelberg New York, USA, 2003.

[12] T. Bäck, D. Fogel, Z. Michalewicz (Eds.), Evolutionary Computation 1: Basic Algorithms and Operators, Institute of Physics Publishing, Bristol, 2000.

[13] J.R. Koza, Genetic Programming, MIT press, 1994.

[14] J.H. Holland, et al., Adaptation in Natural and Artificial Systems: an Introductory Analysis with Applications to Biology, Control, and Artificial Intelligence, MIT press, 1992.

[15] J.F. Miller, An empirical study of the efficiency of learning boolean functions using a Cartesian genetic programming approach, in: W. Banzhaf, J.M. Daida, A.E. Eiben, M.H. Garzon, V. Honavar, M.J. Jakiela, R.E. Smith (Eds.), GECCO, Morgan Kaufmann, 1999, pp. 1135–1142.

[16] J.F. Miller, P. Thomson, CaRtesian genetic programming, in: R. Poli, W. Banzhaf, W.B. Langdon, J.F. Miller, P. Nordin, T.C. Fogarty (Eds.), EuroGP, in: Lecture Notes in Computer Science, vol. 1802, Springer, 2000, pp. 121–132.

[17] J.A. Walker, J.F. Miller, R. Cavill, A multi-chromosome approach to standard and embedded Cartesian genetic programming, in: Proceedings of the 8th Annual Conference on Genetic and Evolutionary Computation, in: GECCO '06, ACM, New York, NY, USA, 2006, pp. 903–910.

[18] A. Swarnalatha, A. Shanthi, Complete hardware evolution based SoPC for evolvable hardware, Appl. Soft Comput. 18 (2014) 314–322.

[19] A.P. Johnson, S. Patranabis, R.S. Chakraborty, D. Mukhopadhyay, Remote dynamic partial reconfiguration: A threat to internet-of-things and embedded security applications, Microprocess. Microsyst. 52 (2017) 131–144.

[20] D. Keymeulen, R.S. Zebulum, Y. Jin, A. Stoica, Fault-tolerant evolvable hardware using field-programmable transistor arrays, IEEE Trans. Reliab. 49 (3) (2000) 305–316.

[21] Xilinx, 7 series FPGAs memory resources - user guide, 2019, URL https://www.xilinx.com/support/documentation/user_guides/ug473_7Series_Memory_Resources.pdf.

[22] R.E. Lyons, W. Vanderkulk, The use of triple-modular redundancy to improve computer reliability, IBM J. Res. Dev. 6 (2) (1962) 200–209.

[23] S. Mal-Sarkar, R. Karam, S. Narasimhan, A. Ghosh, A. Krishna, S. Bhunia, Design and validation for FPGA trust under hardware trojan attacks, IEEE Trans. Multi-Scale Comput. Syst. 2 (3) (2016) 186–198, http://dx.doi.org/10.1109/TMSCS.2016.2584052.

[24] M. Labbafniya, R. Saeidi, Secure FPGA design by filling unused spaces, ISC Int. J. Inf. Secur. 11 (1) (2019) 47–56.

[25] R. Karam, T. Hoque, S. Ray, M. Tehranipoor, S. Bhunia, Mutarch: Architectural diversity for fpga device and ip security, in: Design Automation Conference (ASP-DAC), 2017 22nd Asia and South Pacific, IEEE, 2017, pp. 611–616.

[26] Z. Zhang, L. Njilla, C.A. Kamhoua, Q. Yu, Thwarting security threats from malicious FPGA tools with novel FPGA-oriented moving target defense, IEEE Trans. Very Large Scale Integr. (VLSI) Syst. 27 (3) (2019) 665–678.

[27] G.T. Becker, F. Regazzoni, C. Paar, W.P. Burleson, Stealthy dopant-level hardware trojans, in: Cryptographic Hardware and Embedded Systems - CHES, 2013, pp. 197–214.

[28] S. Ghandali, G.T. Becker, D. Holcomb, C. Paar, A design methodology for stealthy parametric trojans and its application to bug attacks, in: Cryptographic Hardware and Embedded Systems - CHES, 2016, pp. 625–647.

[29] M. Tehranipoor, C. Wang, Introduction to Hardware Security and Trust, Springer Science & Business Media, 2011.

[30] V. Jyothi, M. Thoonoli, R. Stern, R. Karri, FPGA Trust zone: Incorporating trust and reliability into fpga designs, in: 2016 IEEE 34th International Conference on Computer Design (ICCD), IEEE, 2016, pp. 600–605.

[31] E. Dubrova, M. Näslund, G. Carlsson, B. Smeets, Keyed logic BIST for trojan detection in soc, in: 2014 International Symposium on System-on-Chip (SoC), IEEE, 2014, pp. 1–4.

[32] R. Salvador, A. Otero, J. Mora, E. de la Torre, T. Riesgo, L. Sekanina, Self-reconfigurable evolvable hardware system for adaptive image processing, IEEE Trans. Comput. 62 (8) (2013) 1481–1493, http://dx.doi.org/10.1109/TC.2013.78.

[33] K. Glette, Design and implementation of scalable online evolvable hardware pattern recognition systems, 2008.

[34] B. López, J. Valverde, E. de la Torre, T. Riesgo, Power-aware multi-objective evolvable hardware system on an fpga, in: Adaptive Hardware and Systems (AHS), 2014 NASA/ESA Conference on, IEEE, 2014, pp. 61–68.

[35] J. Wang, C.H. Piao, C.H. Lee, Implementing multi-VRC cores to evolve combinational logic circuits in parallel, in: International Conference on Evolvable Systems, Springer, 2007, pp. 23–34.

[36] Z. Bao, T. Watanabe, A new approach for circuit design optimization using genetic algorithm, in: 2008 International SoC Design Conference, Vol. 1, IEEE, 2008, I–383.

[37] F.Z. Hadjam, C. Moraga, M.K. Rahmouni, Evolutionary design of digital circuits using improved multi expression programming (IMEP), Mathw. Soft Comput. 14 (2) (2007) 103–123.

[38] R.E. Keller, W. Banzhaf, The evolution of genetic code in genetic programming, in: Proceedings of the 1st Annual Conference on Genetic and Evolutionary Computation-Volume 2, Morgan Kaufmann Publishers Inc., 1999, pp. 1077–1082.

[39] D. Dechev, R. Ashraf, F. Luna, R. DeMara, Designing digital circuits for FPGAs using parallel genetic algorithms., Tech. Rep., Sandia National Lab.(SNL-NM), Albuquerque, NM (United States), 2012.

[40] K.S. Babu, N. Balaji, Approximation of digital circuits using cartesian genetic programming, in: 2016 International Conference on Communication and Electronics Systems (ICCES), IEEE, 2016, pp. 1–6.
[41] S. Asha, R. Hemamalini, Synthesis of adder circuit using cartesian genetic programming, Middle-East J. Sci. Res. 23 (6) (2015) 1181–1186.
[42] J.E. da Silva, H. Bernardino, CaRtesian genetic programming with crossover for designing combinational logic circuits, in: 2018 7th Brazilian Conference on Intelligent Systems (BRACIS), IEEE, 2018, pp. 145–150.
[43] M. Irfan, Q. Habib, G.M. Hassan, K.M. Yahya, S. Hayat, Combinational digital circuit synthesis using Cartesian genetic programming from a NAND gate template, in: 2010 6th International Conference on Emerging Technologies (ICET), IEEE, 2010, pp. 343–347.
[44] I. Brajer, D. Jakobović, Automated design of combinatorial logic circuits, in: 2012 Proceedings of the 35th International Convention MIPRO, IEEE, 2012, pp. 823–828.
[45] S. Kazarlis, J. Kalomiros, V. Kalaitzis, A Cartesian genetic programming approach for evolving optimal digital circuits., J. Eng. Sci. Technol. Rev. 9 (5) (2016).
[46] A.K. Srivastava, A. Gupta, S. Chaturvedi, V. Rastogi, Design and simulation of virtual reconfigurable circuit for a fault tolerant system, in: International Conference on Recent Advances and Innovations in Engineering (ICRAIE-2014), IEEE, 2014, pp. 1–4.
[47] S. Zamanzadeh, A. Jahanian, Security path: An emerging design methodology to protect the FPGA ips against passive/active design tampering, J. Electron. Test. 32 (3) (2016) 329–343.
[48] C. Albrecht, Iwls 2005 benchmarks, in: International Workshop for Logic Synthesis (IWLS), 2005, http://Www.Iwls.Org.