Information systems. Computer sciences. Issues of information security
Информационные системы. Информатика. Проблемы информационной безопасности

UDC 004.31
https://doi.org/10.32362/2500-316X-2024-12-4-23-39
EDN DRCIUV

RESEARCH ARTICLE

Identification of digital device hardware vulnerabilities based on scanning systems and semi-natural modeling

Evgeniy F. Pevtsov,
Tatyana A. Demenkova [@],
Alexander O. Indrishenok,
Vladimir V. Filimonov

MIREA – Russian Technological University, Moscow, 119454 Russia
[@] Corresponding author, e-mail: demenkova@mirea.ru

Abstract

Objectives. The development of computer technology and information systems requires the consideration of issues of their security, various methods for detecting hardware vulnerabilities of digital device components, as well as protection against unauthorized access. An important aspect of this problem is to study existing methods for the possibility and ability to identify hardware errors or search for errors on the corresponding models. The aim of this work is to develop approaches, tools and technology for detecting vulnerabilities in hardware at an early design stage, and to create a methodology for their detection and risk assessment, leading to recommendations for ensuring security at all stages of the computer systems development process.

Methods. Methods of semi-natural modeling, comparison and identification of hardware vulnerabilities, and stress testing to identify vulnerabilities were used.

Results. Methods are proposed for detecting and protecting against hardware vulnerabilities: a critical aspect in ensuring the security of computer systems. In order to detect vulnerabilities in hardware, methods of port scanning, analysis of communication protocols and device diagnostics are used. The possible locations of hardware vulnerabilities and their variations are identified. The attributes of hardware vulnerabilities and risks are also described. In order to detect vulnerabilities in hardware at an early design stage, a special semi-natural simulation stand was developed. A scanning algorithm using the Remote Bitbang protocol is proposed to enable data to be transferred between *OpenOCD* and a device connected to the debug port. Based on scanning control, a verification method was developed to compare a behavioral model with a standard. Recommendations for ensuring security at all stages of the computer systems development process are provided.

Conclusions. This paper proposes new technical solutions for detecting vulnerabilities in hardware, based on methods such as FPGA system scanning, semi-natural modeling, virtual model verification, communication protocol analysis and device diagnostics. The use of the algorithms and methods thus developed will allow developers to take the necessary measures to eliminate hardware vulnerabilities and prevent possible harmful effects at all stages of the design process of computer devices and information systems.

Keywords: hardware vulnerability, digital components, half-life modeling, diagnostics, scanning, verification

• **Submitted:** 30.08.2023 • **Revised:** 13.01.2024 • **Accepted:** 03.06.2024

For citation: Pevtsov E.F., Demenkova T.A., Indrishenok A.O., Filimonov V.V. Identification of digital device hardware vulnerabilities based on scanning systems and semi-natural modeling. *Russ. Technol. J*. 2024;12(4):23–39. https://doi.org/10.32362/2500-316X-2024-12-4-23-39

Financial disclosure: The authors have no a financial or property interest in any material or method mentioned.

The authors declare no conflicts of interest.

НАУЧНАЯ СТАТЬЯ

Выявление аппаратных уязвимостей цифровых устройств на основе систем сканирования и полунатурного моделирования

Е.Ф. Певцов,
Т.А. Деменкова [@],
А.О. Индришенок,
В.В. Филимонов

МИРЭА – Российский технологический университет, Москва, 119454 Россия
[@] *Автор для переписки, e-mail: demenkova@mirea.ru*

Резюме

Цели. Развитие вычислительной техники и информационных систем требует рассмотрения вопросов их безопасности, различных методов обнаружения аппаратных уязвимостей цифровых компонентов устройств и защиты от несанкционированного доступа. Важным аспектом данных проблем является исследование существующих методов на возможность и способность выявить аппаратные ошибки или произвести поиск ошибок на соответствующих моделях. Цель работы – разработка подходов, инструментов и технологии для обнаружения уязвимостей в аппаратном обеспечении на ранней стадии проектирования, создание методики их обнаружения и оценки риска, рекомендаций по обеспечению безопасности на всех этапах процесса разработки вычислительных систем.

Методы. Использованы методы полунатурного моделирования, сравнения и выявления аппаратных уязвимостей, стресс-тестирования для выявления уязвимостей.

Результаты. Предложены методы обнаружения и защиты от аппаратных уязвимостей, являющихся критически важным аспектом в обеспечении безопасности вычислительных систем. Для обнаружения уязвимостей в аппаратном обеспечении использованы методы сканирования портов, анализа протоколов связи и диагностики устройств. Определены возможные места нахождения аппаратных уязвимостей, их вариации, описаны атрибуты аппаратных уязвимостей и риски. Для обнаружения уязвимостей в аппаратном обеспечении на ранней стадии проектирования разработан специальный стенд полунатурного моделирования. Предложен алгоритм сканирования с использованием протокола Remote Bitbang, который позволяет передавать данные между *OpenOCD* и подключенным к отладочному порту устройством. На основе управления сканированием

разработан метод верификации, реализующий сравнение поведенческой модели с эталоном. Приведены рекомендации по обеспечению безопасности на всех этапах процесса разработки вычислительных систем.
Выводы. В данной работе предложены новые технические решения для обнаружения уязвимостей в аппаратном обеспечении, основанные на таких методах, как сканирование системы на программируемой логической интегральной схеме, полунатурное моделирование, верификация по виртуальной модели, анализ протоколов связи и диагностика устройств. Применение разработанных алгоритмов и способов позволит разработчикам предпринять необходимые меры по устранению аппаратных уязвимостей и предотвращению возможных вредоносных воздействий на всех этапах процесса проектирования устройств вычислительной техники и информационных систем.

Ключевые слова: аппаратная уязвимость, цифровые компоненты, полунатурное моделирование, диагностика, сканирование, верификация

• **Поступила:** 30.08.2023 • **Доработана:** 13.01.2024 • **Принята к опубликованию:** 03.06.2024

Для цитирования: Певцов Е.Ф., Деменкова Т.А., Индришенок А.О., Филимонов В.В. Выявление аппаратных уязвимостей цифровых устройств на основе систем сканирования и полунатурного моделирования. *Russ. Technol. J*. 2024;12(4):23–39. https://doi.org/10.32362/2500-316X-2024-12-4-23-39

Прозрачность финансовой деятельности: Авторы не имеют финансовой заинтересованности в представленных материалах или методах.

Авторы заявляют об отсутствии конфликта интересов.

INTRODUCTION

Hardware vulnerability is an error in the technical implementation of hardware which can allow malicious intruders to gain access to a system or its application. Advanced hardware vulnerabilities can have serious security implications for computer systems, including compromising sensitive data, disruption of systems, and potential threat to human life and health, if vulnerabilities in medical devices or automotive equipment are exploited.[1]

Hardware vulnerabilities can be diverse and include bugs in various computer components: processor, chips, memory, hardware modules, and drivers. They are also contained in other hardware complex function blocks (CF-blocks). Hardware vulnerabilities can occur as a result of interaction with software, particularly drivers, databases, and other applications that contain bugs or malicious code.

Checking for vulnerabilities is a complex process and requires the involvement of information security experts.[2] Hardware vulnerabilities can be identified by scanning the system under investigation, in order to find problem areas and then fix them [1]. One option for finding vulnerabilities in existing computing devices is to use specialized software and tools. For example, many utilities and applications for searching and analyzing vulnerabilities can also be used to check the security of network devices, etc. A comprehensive approach of model comparison in the form of *OpenOCD*[3] binding and a software model of the device obtained using *Verilator* [2] from hardware description languages[4] can also be applied.

Analysis and reverse engineering techniques are also used to find vulnerabilities. These methods can be used to find vulnerabilities in hardware and software systems, operating systems, device drivers, and other computing systems [3].

Each vulnerability has certain attributes which can be used to detect it. This paper describes the attributes of hardware vulnerabilities, methods of their detection, and risk assessment. Attributes of hardware vulnerabilities are considered in accordance with the accepted taxonomy proposed in a number of works which categorize the identification of hardware vulnerabilities and assessment of their risk or severity [4]. This taxonomy identifies four multi-parametric attributes (vectors) corresponding to hardware vulnerability identification, hardware vulnerability detection identification, hardware vulnerability risk or severity, and hardware vulnerability detection efficiency.

[1] Zantout S. *Hardware Trojan Detection in FPGA through Side-Channel Power Analysis and Machine Learning*. MSc Thesis. University of California, Irvine. 2018. https://escholarship.org/uc/item/7hk8x6rb. Accessed May 15, 2023.

[2] Oberg J. *Testing Hardware Security Properties and Identifying Timing Channels*. UC San Diego. 2014. P. 5–38. https://escholarship.org/uc/item/8b530988. Accessed May 15, 2023.

[3] Documentation regarding OpenOCD RISC-V Debug Configuration Commands. https://openocd.org/doc/html/Architecture-and-Core-Commands.html#RISC_002dV-Debug-Configuration-Commands. Accessed May 15, 2023.

[4] Andrianov A.V. *Realization of possibility of step-by-step debugging at debugging of test scenarios on VLSI SonC model*. https://www.module.ru/uploads/media/1534156062-2018-833a272aac.pdf (in Russ.). Accessed May 15, 2023.

TASK STATEMENT. TAXONOMY OF HARDWARE BOOKMARKS

Each hardware vulnerability has a number of attributes, so methods have been developed to perform their detection by means of some or all of the attributes. Information about vectors and attributes is further needed to select methods for vulnerability detection.

Attributes in each category may be ranked based on their importance, uniqueness, weighting, or other criteria determined by the specific Trojan detection project or task. A hardware Trojan is a malicious module embedded in an integrated circuit or a malicious modification of the circuit to alter its operation or add additional functionality, for example, to organize an information leakage channel.

A TI (Trojan Identification) vector includes attributes that are used to identify Trojans, such as Trojan size, location in the circuitry, defense mechanisms used by the Trojan, and other characteristics that may be unique to a particular Trojan.

TDI (Trojan Detection Identification) vector includes attributes that can be detected by a particular Trojan detection method, such as changes in Trojan signatures, anomalies in Trojan behavior, attack features used by the Trojan, and other attributes that can be detected by a particular method.

TR vector (Trojan Risk or Severity) includes attributes that assess the risk or severity of a Trojan, such as the ability of the Trojan to cause harm, how stealthy the Trojan is, how likely it is to spread, the potential consequences of its actions, and other factors that may affect the severity of the Trojan.

TDE vector (Trojan Detection Effectiveness) includes attributes that evaluate the effectiveness of a particular Trojan detection method, such as detection accuracy, false positives and false negatives of the method, detection rate, complexity of the method implementation, and other factors that may affect the effectiveness of the method.

These vectors can be used to evaluate Trojans and select the most effective method of detecting it, identifying it, and assessing its risk or severity.

A combination of TI, TDI, CR, and TDE vectors can be used to select the most effective Trojan detection method based on certain criteria and requirements of a project or task. For example, a method with high values of the TDI and TDE vectors indicating a high Trojan detection capability and method effectiveness may be preferred over a method with lower values in these vectors. Attributes in the TR vector can also be used to assess the severity of a Trojan and its priority when selecting a detection method.

Attributes in each category are used to identify Trojans and assess their risk or severity. They can also be used to identify Trojan detection methods and evaluate their effectiveness. Two vectors TI and TR are assigned to each Trojan. The former identifies the corresponding attributes and the latter represents the attributes in terms of their risk or severity. Each Trojan detection method also has two vectors TDI and TDE. TDI identifies the attributes that can be detected and TDE represents the attributes in terms of the effectiveness of the method. Thus, there are four vectors corresponding to: 1) TI Trojan identification, 2) TDI Trojan detection identification, 3) TR Trojan risk or severity, 4) TDE Trojan detection effectiveness [5].

The proposed approach based on ranking the attributes of Trojans in each category and using TI, TR, TDI, and TDE vectors can be a useful tool for identification and risk assessment of Trojans, as well as for selecting effective detection methods when developing more robust security and defense against attacks.

However, it should be noted that the proposed Trojan attribute ranking system may be limited because new types of Trojans and their attributes may emerge as a result of evolving technologies and attack methods. Therefore, the database of Trojan attributes and detection methods need to be continuously updated and supplemented, in order to better protect against them.

Detecting vulnerabilities is only the first step. Steps must also be taken to remediate the vulnerabilities and prevent potential attacks. In this process, tools such as computer-aided design (CAD) systems can play an important role in verifying devices and detecting vulnerabilities early in the design process. A typical example of such a system is the CAD tool suite of Cadence Design Systems [6]. In general, security is a critical aspect in the development of computing systems, detection and protection against hardware vulnerabilities should be considered at all stages of the development and operation process.

In addition to the implemented function, the vulnerability can be upgraded for remote interaction. Here it is necessary to program the ability to access the network card and go online. In addition, the bookmark can be configured to corrupt not only the output data at a given time, but also to modify already recorded data. To do this, the bookmark must have access to memory storages, as well as the ability to detect certain data or the ability to change all data by a certain amount. Writing and modernizing a bookmark is limited only by the skills of the attacker and the size of the bookmark [7–9].

HARDWARE BOOKMARK IMPLEMENTATION EXAMPLE

An important aspect of the methodology for identifying hardware bookmarks is the procedure for creating possible examples. Usually, the main difficulty

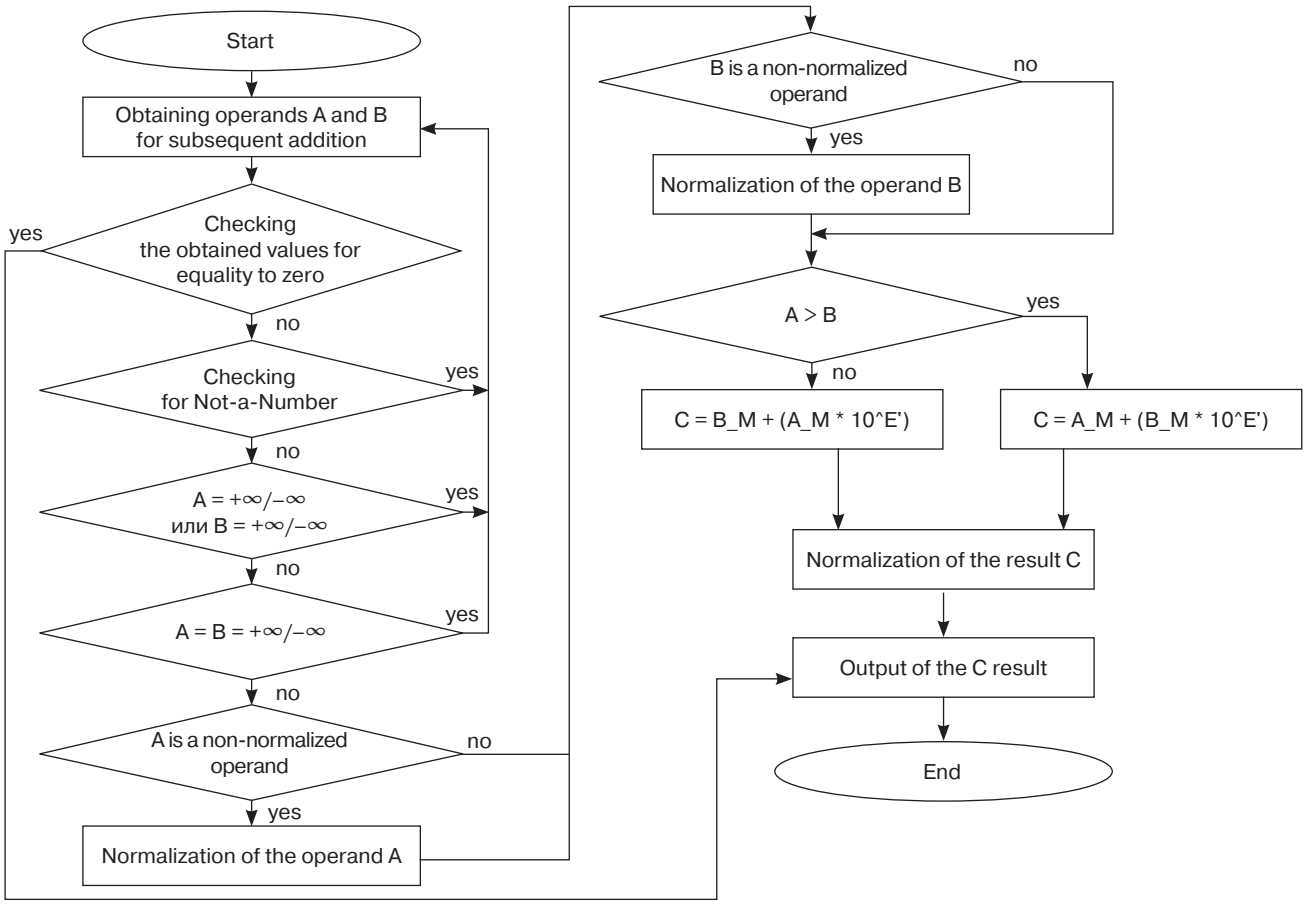

Fig. 1. Algorithm of adder operation

arises with regard to what exactly to show and how to present it. The example can be useful for developers of computer security devices during design in terms of identifying possible vulnerabilities.

Let us consider hardware bookmarks which violate the accuracy of calculations in the kernels, also known as the hardware gas pedal of mathematical calculations. When adding real numbers at the adder, multiplier or divider, you can create a hardware bookmark that violates the correctness of calculations. Let us introduce the bookmark into the addition algorithm and adder code using a separate module and synthesize the resulting computation block.

This module performs the following procedure: when the adder receives a certain number, the bookmark triggers and replaces this number with the one needed by the intruder. It then passes this number directly to the summation. If the bookmark has not received the number necessary for its transition, it remains inactive. Thus, the bookmark spoils output data, while giving its appearance at a particular moment of receipt of a certain number at the input. Figure 1 shows the algorithm of the adder block operation for practicing the skills of introducing and detecting bookmarks in the most common nodes of digital computer systems.

A hardware tab in the presence of the fractional part in the result will change its value, and affect calculation accuracy. The presence of a fractional part in a number in the encoded 32-bit IEEE 754 standard format[5] can be detected by the presence of units in the digits of the number:

$$s + e + \exp + 1 \le n, \quad (1)$$

where s is the number of bits allocated to the sign of the number, e is the number of bits allocated to the shifted exponent of the number, exp is the exponent of the number, and n is the total number of bits.

In the 32-bit representation (1) can be written as:

$$10 + \exp \le 32. \quad (2)$$

If the number has a fractional part, then we invert the digit $s + e + \exp + 2$. In other words, the number $5.5_{10} = 101.1\mathbf{0}0_2$ after inverting this digit will become equal to $5.75_{10} = 101.1\mathbf{1}0_2$.

[5] 2754-2019 IEEE Standard for Floating-Point Arithmetic. July 22, 2019. Electronic ISBN 978-1-5044-5924-2. https://ieeexplore.ieee.org/document/8766229. Accessed May 15, 2023.

Block diagram of the hardware bookmarking algorithm is shown in Fig. 2 [10, 11].

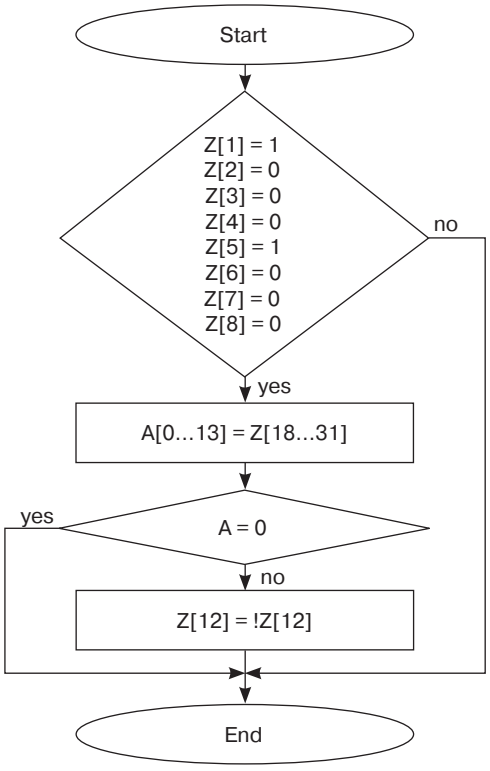

Fig. 2. Block diagram of the hardware bookmarking algorithm

In order to reduce the frequency of this event which triggers a hardware tab, we will take only numbers that, in addition to having a fractional part, have values only in the range $512 \le x < 1024$. The prerequisite for this condition can be met from the value of the exponent of the binary number. If the exponent of the binary number is equal to $9_{10} = 1001_2$, which corresponds to the shifted exponent 10001000_2, then this number lies in the interval $512 \le x < 1024$.

As a result, this hardware Trojan will change the result on the adder only if this result belongs to the interval $512 \le x < 1024$ and has a fractional part.

Since we have determined the order of numbers required for distortion to be equal to 9_{10}, the previously described step of finding the right digit in a number to check whether the number has a fractional part is no longer required. Any 32-bit IEEE 754 numbers lying in the interval $512 \le x < 1024$ with one in any of $18 \le n \le 31$ digits will have a fractional part according to formulas (1) and (2):

$$1 + 8 + 9 + 1 = 19 \le n.$$

The encoded fractional part of the number will be located in the interval from 19 digits to 32 digits, and the encoded integer part of the number is located in digits 9 to 17. Therefore, in order to check whether the number has a fractional part, at least one unit in digits 18–31 needs to be found.

From the adder result bus, the value of the resulting encoded number is read into register Z of the hardware Trojan. If the shifted exponent of the number is equal to 10001000_2, then the remainder of the mantissa of the encoded number is written to register A of the hardware tab. Further, if this register A is non-empty, then in register Z the z value of the 12th digit is inverted, corresponding to the second digit of the fractional part of the number. As a result, the number is either increased by 0.25 if there is 0 in this digit, or decreased by 0.25 if there is 1 in this digit. The developed hardware Trojan can be embedded into the real number adder core of a hardware math gas pedal. The developed bookmark is described in the Verilog language [12].

After the successful development of a hardware Trojan, verification must be performed. This is required, in order to determine whether the hardware tab works correctly depending on the specified conditions. For the purpose of this verification, we define several possible events. Each of the events must be verified both with and without the presence of the hardware Trojan.

1. Result of the adder lies in the interval $512 \le x < 1024$.
   a) result has a fractional part;
   b) result has no fractional part.
2. Result of the adder does not belong to the interval $512 \le x < 1024$.
   a) result is less than 512 and has a fractional part;
   b) result is less than 512 and has no fractional part;
   c) result is greater than or equal to 1024 and has a fractional part;
   d) result is greater than or equal to 1024 and has no fractional part.
3. The adder result is an exceptional number.
   a) result is equal to 0;
   b) result is equal to $+\infty$;
   c) result is equal to $-\infty$.

Such hardware vulnerability can be detected using the following methods.

1. **Anomalous behavior analysis.** The presence of a bookmark can lead to abnormal behavior of the floating-point unit (FPU), such as unexpected calculation errors, incorrect results, or unusual activity during mathematical operations.
2. **Application of Linpack performance test** [13]. A hardware tab introduced into FPU is difficult to detect using this test because its size has a negligible effect on performance. However, this method will be effective, if it is possible to compare two hardware blocks: with and without the bookmark. One model has an intentionally introduced hardware vulnerability and is loaded into a programmable logic integrated circuit (PLIC). The other model

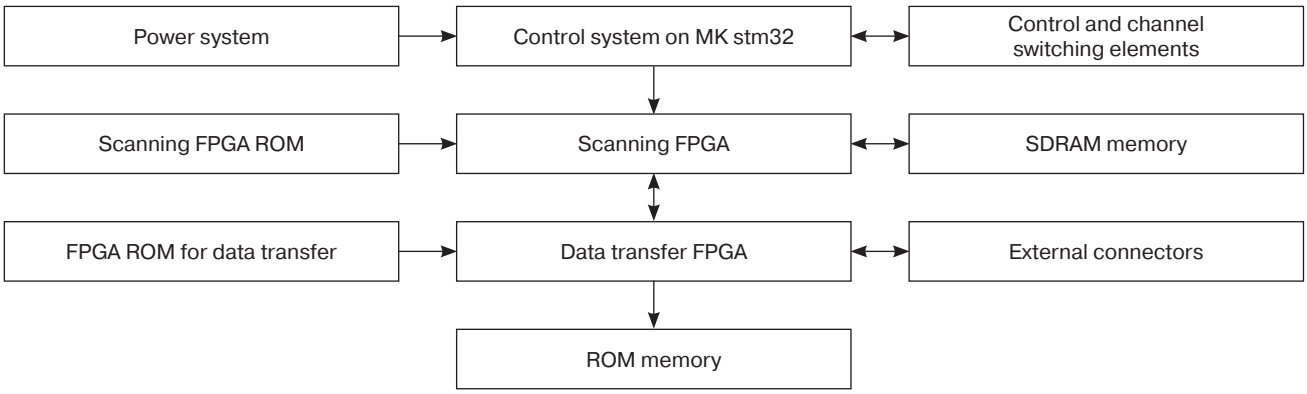

Fig. 3. The structure of the bench for conducting experiments to identify hardware vulnerabilities. ROM, read-only memory; SDRAM, synchronous dynamic random access memory; MK stm32, STM32 microcontroller

is behavioral and is implemented with some simplifications in a simulator on a computer using special CAD tools for integrated circuits, such as Cadence Design Systems tools.

3. **Checking the instruction set** inside the processor using the Joint Test Action Group (JTAG) interface[6] and the Remote Bitbang protocol-based scanning system. Specifically, this meant auditing the FPU microcode for unusual or suspicious instructions which may indicate the presence of a hardware bookmark.

   The Remote Bitbang protocol is a protocol used in *OpenOCD* (an open source program for debugging embedded systems) which allows data to be transferred between *OpenOCD* and a device that is connected to the debug port. The Remote Bitbang protocol is used to control I/O and other device interfaces.

   The Remote Bitbang protocol uses asynchronous data transfer, represented as a simple message format. Commands sent through this protocol are of two types: write commands and read commands. Write commands allow you to control pins, transfer data, and set interface modes, while read commands allow you to receive data from the device via inputs.

   Remote Bitbang protocol is used in conjunction with protocols such as JTAG, in order to enable full debugging and programming of embedded systems. This protocol can be used to scan registers on a remote device connected to *OpenOCD*, and in the automation of testing, fault diagnosis, and device programming in production environments [14–16].

4. **The use of specialized tools** (microcode analyzers and specialized stands for detecting hardware bookmarks) which can be useful in detecting such threats. However, their use requires specialized knowledge and experience.

[6] https://ru.wikipedia.org/wiki/JTAG (in Russ.). Accessed May 15, 2023.

SEMI-NATURAL SIMULATION BENCH

When developing a stand designed to detect hardware vulnerabilities, it needs to ensure the reliability of the results obtained and fulfill the following functions:

1. Ability to test various types of hardware, including processors, chipsets, I/O controllers, network cards, and other components.
2. Ability to use a variety of tools to detect vulnerabilities in hardware, including vulnerability scanners, debuggers, emulators, and other tools.
3. Ability to experiment in a variety of scenarios, including malware attacks on hardware, memory and peripheral manipulation, and vulnerability exploits.
4. Ensue security and data protection during experiments, including protection against leakage of confidential information and ensuring confidentiality of test results.
5. Ensure that test results can be reproduced and all steps of the experiment can be documented, including test automation and tools for analyzing results.

The structural diagram of the stand is shown in Fig. 3.

The feature is a special FPGA structure that enables verification for the repeated firmware in order to ensure that there are no changes to the complex programmable logic device (CPLD) circuit. It is also possible to use the FPGA's built-in resources for basic firmware verification using built-in controls.

Verification of the Verilog code used to generate the firmware file (bitstream) can be performed using various methods and tools such as simulation, formal verification, and emulation on hardware.

1. **Simulation.** After creating test vectors which represent different bitstream loading scenarios in the FPGA, the Verilog simulator should be used to perform a simulation of these test vectors. As a result, the signal values within the bitstream comparison circuitry can be analyzed and compared to the expected values. This enables error detection

and verification that the bitstream comparison methods are working correctly.

2. **Formal verification** is a method of verifying the correctness of Verilog code based on mathematical algorithms. Formal verification tools, such as Model Checking or Equivalence Checking, can be used to verify that bitstream comparison methods work correctly. This can include checking the correctness of the comparison logic, detecting potential errors, and finding unexpected execution paths.
3. **Emulation on hardware.** Load the bitstream into the FPGA and run it on a physical device. Test vectors and physical signals can then be used to verify that the bitstream comparison methods work correctly in real time. This can help identify possible problems when running on the real hardware.
4. **Manual validation.** The code of bitstream comparison methods can be carefully analyzed and manual validation performed against requirements and expected behavior. This can include analyzing logic, checking boundary conditions, and testing different bitstream loading scenarios.

Another feature of the semi-natural simulation bench architecture is the ability to load into internal memory via the internal FPGA reconfiguration interface. If such an interface is not available, the standard external reconfiguration interface can be used. In order to avoid problems when verifying and loading into memory from the FIFO (first in, first out) buffer, verification blocks need to be placed in strictly allocated FPGA cells when writing the configuration and layout on-chip. In addition, this module has multi-stage verification to detect threats, report them, and eliminate possible negative effects. A report of the validation results is saved and can be used to further analyze and improve the module's performance.

Since FPGA architectures can vary from model to model and manufacturer to manufacturer, this structure enables the verification algorithms to be adapted to different architectures and manufacturers. Compared to the closest analogs, this module is more relevant because it has the ability to connect to various external devices. This provides for its versatility and the possibility of it being used as a master device in information switching systems, as well as in systems with high-performance processors [17, 18].

Before scanning, device operation modes need to be selected which will allow you to identify possible vulnerabilities in the RS-485 data transmission system. For example, you can configure the device to send malicious commands or to intercept and analyze data transmitted over the network. It should be taken into account that such actions can lead to disruption of the device or the network as a whole. Therefore, the experiments should be conducted in a controlled environment and with prior coordination with those responsible for the operation of the system [1, 19].

An example of implementation of the semi-natural simulation stand is shown in Fig. 4.

Fig. 4. Implementation of semi-natural simulation bench

A configuration according to the target model is loaded into the data transfer FPGA, describing the expected results of the device. Then the process begins of obtaining readings from the external interfaces of the unit under investigation, such as RS-485 or Ethernet[7], and comparing them with the target model. If the results do not correspond to the expected results, an error is generated and recorded in the report. Then the indicator on the device signals the occurrence of the error[8].

When connecting the bench and debugging the FPGA firmware, in order to obtain the necessary data for comparison with the target model deployed on a computer, it is important to note that the connection scheme of the JTAG programmer to the FPGA may differ depending on the specific model of the programmer and FPGA, as well as the task to be solved by this connection[9, 10, 11]. In this case, if additional peripherals

[7] Andrianov A.V. *Realization of possibility of step-by-step debugging at debugging of test scenarios on VLSI SonC model*. https://www.module.ru/uploads/media/1534156062-2018-833a272aac.pdf (in Russ.). Accessed May 15, 2023.

[8] Fern N.C. *Verification Techniques for Hardware Security*: Ph.D. Thesis (Comput.). USA: UC Santa Barbara; 2016. P. 10–25. https://escholarship.org/uc/item/2ch6f44s. Accessed May 15, 2023.

[9] Cadence documentation. https://www.cadence.com/content/cadence-www/global/en_US/home/support/documentation.html. Accessed May 15, 2023.

[10] Yang P. *Assessing VeSFET Monolithic 3D Technology in Physical Design, Dynamic Reconfigurable Computing, and Hardware Security*: Ph.D. Thesis (Comput.). USA: UC Santa Barbara; 2017. P. 62–81. https://escholarship.org/uc/item/5s9833kz. Accessed May 15, 2023.

[11] Farinholt B.R. *Understanding the Remote Access Trojan malware ecosystem through the lens of the infamous DarkComet RAT*: Ph.D. Thesis (Comput.). USA: UC San Diego; 2019. P. 17–29. https://escholarship.org/uc/item/3vv544n5. Accessed May 15, 2023.

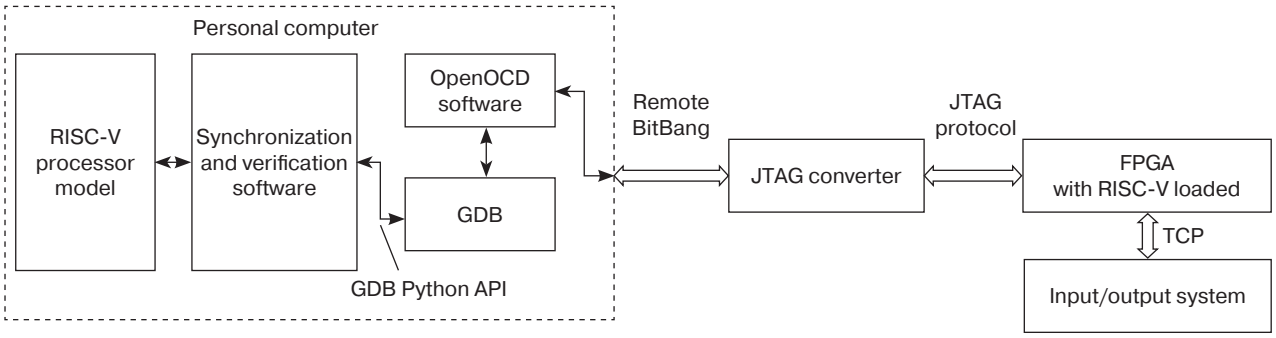

Fig. 5. A model for finding vulnerabilities in a RISC-V processor-based system. API, application programming interface; TCP, transmission control protocol; GVB, Gnu DeBugger, open source debugger

need to be connected to the FPGA, for example, to debug a system containing a microcontroller, a special debug module may also need to be connected.

EXPERIMENTAL

A semi-natural simulation test bench is performed on the basis of verification of a processor design, e.g., RISC-V,[12] in whose behavioral description in the Verilog language code of an intentional hardware vulnerability is introduced. This vulnerability can be represented as a modified SF block ready to be embedded into a system with RISC-V architecture for hardware vulnerability detection experiments[13] [20].

After implementing the modified hardware NF block, the performance needs to be compared, and the node with vulnerability first identified using error detection software. Then, using virtual model analysis, analysis needs to be performed on the FPGA using a combination of Remote Bitbang and JTAG protocols, as well as *OpenOCD* software. Then the results need to be compared and possible deviations from the behavioral model of the device identified. The general concept of the scanning idea is presented in Fig. 5.

The device behavioral model is an abstract model which describes the functional behavior of a device and its interaction with other system components. It describes how a particular unit should behave under certain input signals and conditions. It does not specify the internal implementation of the device, and how this behavior is reflected in the overall picture of the device's interaction with external sources of information.

A description in unified modeling language, used to describe the behavior of the entire device, should be used for verification, but other options are possible.

You can use automated security checkers to check Verilog code for vulnerabilities or hardware flaws. These systems analyze the code for bugs and look for possible vulnerabilities. Third-party tools such as *Verilator* and *VeriSim*[14] can also be used to analyze and check Verilog code for vulnerabilities.

Verilator is an open source translator from Verilog to C/C++ which generates files for profiling and debugging. It allows users to analyze code for hardware bookmarks and vulnerabilities. This tool can be used to automatically check Verilog code for bugs.

VeriSim is an another tool for analyzing and verifying Verilog code. It allows users to analyze code using simulation and profiling. It also enables identification of hardware bookmarks and vulnerabilities in Verilog code.

Alternative tools for analyzing and inspecting Verilog code for hardware bookmarks and vulnerabilities include *Vivado HLS*[15], *Synopsys VCS*[16], and *Mentor Questa*[17].

Automated vulnerability scanning tools such as *SonarQube*[18] and *Coverity*[19] can be used to find

[12] RISC-V is an extensible open and free instruction system and processor architecture based on the RISC (reduced instruction set computer) concept for processor/microcontroller design and software development.

[13] Li C. *Securing Computer Systems Through Cyber Attack Detection at the Hardware Level*: Ph.D. Thesis (Comput.). USA: UC Irvine; 2020. P. 13–26. https://escholarship.org/uc/item/8vr8f0dq. Accessed May 15, 2023.

[14] Wei S. *Minimizing Leakage Energy in FPGAs Using Intentional Post-Silicon Device Aging*: Master Sci. Thesis. USA: UC Los Angeles; 2013. P. 16–35. https://escholarship.org/uc/item/75h4m6qb. Accessed May 15, 2023.

[15] https://docs.xilinx.com/r/en-US/ug949-vivado-design-methodology/Vivado-Design-Suite-User-and-Reference-Guides. Accessed May 15, 2023.

[16] https://users.ece.utexas.edu/~patt/10s.382N/handouts/vcs.pdf. Accessed May 15, 2023.

[17] https://www.orcada.ru/product/mentor-graphics/proektirovanie-zakaznyh-ims/products_106.html (in Russ.). Accessed May 15, 2023.

[18] https://www.sonarsource.com/products/sonarqube/. Accessed May 15, 2023.

[19] https://devguide.python.org/development-tools/coverity/. Accessed May 15, 2023.

vulnerabilities in SystemC program code. These tools enable automated scanning of the source code for vulnerabilities. If vulnerabilities or hardware bookmarks are not found, there is a risk that the application will be vulnerable to attack by malicious attackers who can exploit the vulnerability to make unwanted changes to the code or gain access to data[20, 21, 22, 23, 24] [21, 22].

In order to perform synchronization between a virtual processor and a real processor undergoing debugging, *OpenOCD* capabilities can be used. *OpenOCD* needs to be connected to the debug interface of the real processor. This can be done, for example, via the JTAG interface *OpenOCD* must then be configured to work with a virtual processor undergoing debugging, such as a processor emulated in *QEMU* [23] or through the Functional Safety Simulator[25].

Next, synchronization between the virtual and real processor must be performed. This can be done with the resume command which allows execution of the process on the virtual processor to be continued while synchronizing it with the real processor. Thus, it is possible to debug the virtual processor while being able to monitor its operation in real time.

For correct synchronization the correct *OpenOCD* configuration is needed, in order to work with a specific virtual processor and debugging interface of a real processor. The particular features of working with certain types of processors and debug interfaces also need to be taken into account.

In order to build a project for testing for hardware vulnerabilities, the cross-platform automatic build system *CMake* can be used. This allows you to create, test and package a build system for source code, as well as freely distributed compilers[26].

Figure 6 shows a code fragment of building a test program using *CMake* for the RISC-V processor. In order to execute the procedure, you need to install *CMake* and RISC-V compiler. To do this, you can use the operating system package manager or download them from the official websites.

To start it, the commands 'mkdir build && cd build && cmake ... && make' need to be executed. Then the project will be built, and the result of the build will be loaded into the stand and the virtual testing system, the code description of which is given in Fig. 7.

Since a hardware vulnerability in the FPU compute unit is under consideration here, one option for identifying vulnerabilities is to directly stress test systems related to floating-point computing.

Identifying hardware vulnerabilities by means of stress testing can be quite complex, unproductive, and depends on the type of vulnerability. However, the following approaches can be distinguished:

1. Changing operating conditions: stress testing can be used to identify vulnerabilities associated with prolonged operation of the device under high load conditions. For example, you can increase the number of requests to the device, increase the duration of operation, or change the temperature, humidity, or other operating parameters.
2. Attack simulation: with stress testing, attacks on the device can be simulated.
3. Congestion testing: stress testing can be used to test the device's resistance to congestion, for example, to check how the device handles high traffic or a situation where the number of users on the network increases dramatically.
4. Use of random test data: by generating random test data, the robustness of the device against data errors can be tested, e.g., data transmission errors or data storage errors.

It is important to realize that stress testing can only help identify specific hardware vulnerabilities. For a complete check, a combination of different methods and tests must be used, and the security recommendations of the device manufacturer must be followed. An evolution of this concept is the identification of vulnerabilities through software scanners.

The creation of a stand for identifying hardware vulnerabilities, as well as the technology of working with it, can be considered as one result of this research. This is because the available scientific literature provides no clear definition and description of the steps required to create similar hardware and developing examples for practicing possible vulnerable situations.

[20] Architecture and Core Commands. https://openocd.org/doc/html/Architecture-and-Core-Commands.html#RISC_002dV-Authentication-Commands. Accessed May 15, 2023.

[21] Verilog-Mode Help. https://veripool.org/verilog-mode/help/. Accessed May 15, 2023.

[22] Shepherd C., Markantonakis K. *Vulnerabilities analysis and attack scenarios description.* 2021. https://exfiles.eu/wp-content/uploads/2022/07/EXFILES-D5.1-Vulnerabilities-analysis-and-attack-scenarios-description-PU-M06.pdf. Accessed May 15, 2023.

[23] Wang B. *Improving and Securing Machine Learning Systems*: Ph.D. Thesis (Comput.). USA: UC Santa Barbara; 2019. P. 10–14. https://escholarship.org/uc/item/1nv8m9nb. Accessed May 15, 2023.

[24] Guo Z. *Security of Internet of Things Devices and Networks*: Ph.D. Thesis (Comput.). USA: UC Irvine; 2016. P. 1–30. https://escholarship.org/uc/item/4rq8s4jx. Accessed May 15, 2023.

[25] Spear Ch. *System Verilog for Verification: A Guide to Learning the Testbench Language Features*. Springer. 2018. https://3ec1218usm.files.wordpress.com/2016/12/book_systemverilog_for_verification.pdf. Accessed May 15, 2023.

[26] Qualys platform. https://www.qualys.com/solutions/pci-compliance/. Accessed May 15, 2023.

```
set(MCU_RISCV RISCV)
set(START_FILE startup.s) # Startup file s

add_compile_options(-Wall -Wextra)
add_compile_options(-O2 -ggdb)
add_link_options(-mthumb -mfpu=fpv4-sp-dl6 -mfloat-abi=hard
                 -T${RISCV_LDSCRIPT} --specs=nosys.specs --specs=nano.spec

# set the project name
project(Test_firs_prj VERSION 0.1)

include_directories(
                    "${PROJECT_BINARY_DIR}"
                    "${PROJECT_SOURCE_DIR}/inc"
                    "${PROJECT_SOURCE_DIR}/library/CMSIS"
                     )

set_property(SOURCE ${START_FILE} PROPERTY LANGUAGE C)

add_subdirectory(library/CMSIS)

configure_file(inc/version.h.in inc/version.h)

list(APPEND TARGET_SOURCE ${PROJECT_SOURCE_DIR}/inc/main.h)
list(APPEND TARGET_SOURCE ${PROJECT_SOURCE_DIR}/src/main.c)
```

Fig. 6. Example of program test assembly system

```
    delay_lms(100);
    printf("linpackc test \r\n");
    rtc_print_current_time();
    printf("Start >> linpackc test \r\n");

    rtc_print_current_time();
    printf("Stop >> linpackc test \r\n");

    while(1) {
        time_start = rtc_get_subsecond();
        linpackc_test();
        uint16_t real_len = read_string(test_string, TEST_STR_SIZE);
        printf("len read data = %d \r\n",real_len);

        if(real_len){
            printf("read data = %s \r\n",test_string);
            memset(test_string, 0, TEST_STR_SIZE);
        }

        time_stop = rtc_get_subsecond();
        uint64_t runtime = (uint64_t)time_stop-(uint64_t)time_start;
        printf("runtime = %ld \r\n",(uint32_t)runtime);
        rtc_print_current_time();
    }
}
```

Fig. 7. Code for running a stress test to verify the RISC-V behavioral description

IDENTIFYING VULNERABILITIES THROUGH SOFTWARE VULNERABILITY SCANNERS

A comparison of the most commonly used software vulnerability scanning systems is shown in the table below.

One of the tools specializing in searching for errors and vulnerabilities in C, C++, C#, and Java code is *PVS-Studio*, static code analyzer. However, for a more complete C code scanning you may need other tools such as *Gitleaks*, *Trivy*[27], *Burp Suite*[28], and *MobSF*[29]. Here is a brief description of these tools and their capabilities for detecting vulnerabilities in C code.

1. *Gitleaks* is a tool for finding sensitive data in Git repositories which can be used to scan C code stored in Git.
2. *Trivy* is a tool for scanning Docker containers and images for vulnerabilities in the packages and dependencies used. It can be used to scan Docker images containing C code.
3. *Burp Suite* is a popular web application security testing tool used to scan web applications written in the C language.
4. *MobSF* is a mobile application vulnerability scanning tool used to scan mobile applications written in C language (e.g., using Native Development Kit).

Table. Comparison of different software vulnerability scanning systems

No.	Scanner Name	Open code	C/C++ support	Main purpose
1	*Sn1per*[30]	no	no	It is used to scan for vulnerabilities in web applications, ports, and network devices, supports dictionary brute force scanning and allows you to customize various scanning parameters
2	*Wapiti3*[31]	yes	no	It is a vulnerability scanner for web applications that is open source and can automatically search for vulnerabilities in various parts of a web application such as URL parameters, forms, headers, and scripts
3	*Nikto*[32]	yes	no	It performs database and extensible script-based vulnerability scanning, enables scanning various vulnerabilities such as cross-site scripting (XSS), SQL-injections, header spoofing, unauthorized access, etc.
4	*OWASP ZAP*[33]	yes	no	It provides automatic or manual scanning and penetration tests. It is used to search for vulnerabilities such as SQL injection, XSS, cross-site request forgery, invalid authorization, etc.
5	*Sqlmap*[34]	yes	no	It is a tool for automatic scanning of SQL injection vulnerabilities in web applications. It supports multiple databases including *MySQL, Oracle*, *PostgreSQL*, *Microsoft SQL Server*, etc.
6	*Acunetix WVS*[35]	no	no	It is a web application vulnerability scanning tool that provides automatic or manual scanning options. It supports detection of vulnerabilities such as XSS, SQL injection, information leaks, file security breaches, etc.

[27] https://trivy.dev/. Accessed May 15, 2023.
[28] https://portswigger.net/burp. Accessed May 15, 2023.
[29] https://github.com/MobSF/Mobile-Security-Framework-MobSF. Accessed May 15, 2023.
[30] https://github.com/1N3/Sn1per/releases. Accessed May 15, 2023.
[31] https://pypi.org/project/wapiti3/ (in Russ.). Accessed May 15, 2023.
[32] https://github.com/sullo/nikto. Accessed May 15, 2023.
[33] https://www.zaproxy.org/docs/. Accessed May 15, 2023.
[34] https://sqlmap.org/. Accessed May 15, 2023.
[35] https://allsoft.ru/software/vendors/acunetix/acunetix-web-vulnerability-scanner-/ (in Russ.). Accessed May 15, 2023.

Table. Continued

No.	Scanner Name	Open code	C/C++ support	Main purpose
7	*Vega*[36]	yes	no	It is a tool for scanning vulnerabilities in web applications. It supports dictionary brute-force scanning and provides penetration tests and enables detection of vulnerabilities such as XSS, SQL-injections
8	*PVS-Studio*[37]	no	yes	A static code analyzer in C, C++, C#, and Java designed to search for errors and vulnerabilities
9	*Gitleaks*[38]	no	yes	It is a tool for scanning open source Git repositories for sensitive information and other security vulnerabilities. It operates by analyzing source code, commits, and change history in the repository for lines of code containing sensitive information such as passwords, secure shell keys, access tokens, API secrets
10	*QARK*[39]	yes	no	Java application scanner for Android and IOS

Each of these tools is designed to detect vulnerabilities in different areas. When combined, they can provide more comprehensive code security coverage.

However, it should be noted that most of the scanners on the list can be used to scan code in a variety of programming languages, including C. However, depending on your specific needs and the type of vulnerabilities you need to detect, a combination of several tools may need to be used.

CONCLUSIONS

The research carried out herein has shown that the proposed approach based on scanning systems and semi-natural modeling can successfully identify the hardware vulnerabilities of digital systems when other methods prove ineffective and the analysis of the results is difficult to interpret. Only in a synthetic experiment using semi-natural modeling is it possible to narrow the search area and identify a system with embedded malicious code with a vulnerability. The results of the experiments allowed us to develop a methodology and define a set of tools for identifying vulnerabilities in digital devices of computing systems, as well as to create a library of ready-made solutions for implementing an optimal solution.

The results obtained and the stand we developed for conducting experiments can be used in perspective projects linked to the creation of digital devices on a modern element base. It offers the possibility of transition to new technologies for detection of hardware vulnerabilities. Hardware security should be considered as a priority task in various industries and spheres of activity. The detection and elimination of vulnerabilities of digital components of devices should be carried out both at early stages of development and at the stage of operation.

[36] https://subgraph.com/vega/. Accessed May 15, 2023.
[37] https://pvs-studio.ru/ru/pvs-studio/ (in Russ.). Accessed May 15, 2023.
[38] https://github.com/gitleaks/gitleaks. Accessed May 15, 2023.
[39] https://github.com/linkedin/qark. Accessed May 15, 2023.

ACKNOWLEDGMENTS

This work was supported by the Ministry of Science and Higher Education of the Russian Federation (State task for universities No. FGFZ-2023-0005) and using the equipment of the Center for Collective Use of RTU MIREA (agreement dated September 01, 2021, No. 075-15-2021-689, unique identification number 2296.61321X0010).

Authors' contributions

E.F. Pevtsov—the research idea, consultations on research issues, writing the text of the article.

T.A. Demenkova—the research idea, research planning, scientific editing of the article.

A.O. Indrishenok—the research idea, conducting research, writing the text of the article, interpretation and generalization of the results.

V.V. Filimonov—consultations on research issues, writing the text of the article.

REFERENCES

1. Smetana D. FPGA-Enabled Trusted Boot Is Part of Building Security into Every Aspect of Trusted Computing Architectures. *Military & Aerospace Electronics Journal*. September 25, 2019. Available from URL: https://www.militaryaerospace.com/trusted-computing/article/14040672/trustedcomputing-embedded-computing-realworld
2. Sesin I.Yu., Bolbakov R.G. Comparative analysis of software optimization methods in context of branch predication on GPUs. *Russ. Technol. J.* 2021;9(6):7–15 (in Russ.). https://doi.org/10.32362/2500-316X-2021-9-6-7-15
3. Shayan M., Basu K., Karri R. Hardware Trojans Inspired Hardware IP Watermarks. *IEEE Design & Test*. 2019;36(6):72–79. https://doi.org/10.1109/MDAT.2019.2929116
4. Hennessy J.L., Patterson D.A. A new golden age for computer architecture: Domain-specific hardware/software co-design, enhanced security, open instruction sets, and agile chip development. In: *Proceedings of the 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture* (*ISCA*). IEEE; 2018. https://doi.org/10.1109/ISCA.2018.00011
5. Li D., Zhang Q., Zhao D., Li L., He J., Yuan Y., Zhao Y. Hardware Trojan Detection Using Effective Property-Checking Method. *Electronics*. 2022;11(17):2649. https://doi.org/10.3390/electronics11172649
6. Alekhin V.A. Designing electronic systems using SystemC and SystemC-AMS. *Russ. Technol. J.* 2020;8(4):79–95 (in Russ.). https://doi.org/10.32362/2500-316X-2020-8-4-79-95
7. Yang K., Zhang K., Ren J., Shen X. Security and privacy in mobile crowdsourcing: Challenges and opportunities. *IEEE Commun. Mag*. 2015;53(8):75–81. https://doi.org/10.1109/MCOM.2015.7180511
8. Lou X., Zhang T., Jiang J., Zhang Y. *A Survey of Microarchitectural Side-channel Vulnerabilities, Attacks and Defenses in Cryptography*. Vol. 1. No. 1. March 2021. Available from URL: https://arxiv.org/pdf/2103.14244
9. Skorobogatov S., Woods C. Breakthrough Silicon Scanning Discovers Backdoor in Military Chip. In: Prouff E., Schaumont P. (Eds.). *Cryptographic Hardware and Embedded Systems – CHES 2012*. *Lecture Notes in Computer Science*. 2012. V. 7428. Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-642-33027-8_2
10. Tasiran S., Keutzer K. Coverage metrics for functional validation of hardware designs. *IEEE Des. Test. Comput.* 2001;18(4):36–45. https://doi.org/10.1109/54.936247
11. Mukhopadhyay D., Chakraborty R.S. *Hardware Security: Design, Threats, and Safeguards*. CRC Press; 2014. 542 p. ISBN 978-1-4398-9584-9
12. Tarasov I.E. *PLIS Xilinx. Yazyki opisaniya apparatury VHDL i Verilog, SAPR, priemy proektirovaniya* (*FPGA Xilinx. Hardware Description Languages VHDL and Verilog, CAD, Design Techniques*). Moscow: Goryachaya liniya – Telekom; 2024. 538 p. (in Russ.). ISBN 978-5-9912-0802-4
13. Turkington K., Masseios K., Constantinides G.A., Leong P. FPGA Based Acceleration of the Linpack Benchmark: A High Level Code Transformation Approach. In: *2006 International Conference on Field Programmable Logic and Applications*. IEEE; 2007. INSPEC Accession Number: 9604301. https://doi.org/10.1109/FPL.2006.311240
14. Tamuly S., Joseph A. Chandrasekharan J. Deep Learning Model for Image Classification. In: Smys S., Tavares J., Balas V., Iliyasu A. (Eds.). *Computational Vision and Bio-Inspired Computing. ICCVBIC 2019. Advances in Intelligent Systems and Computing*. Springer, Cham; 2019. V. 1108. P. 312–320. https://doi.org/10.1007/978-3-030-37218-7_36
15. Majeric F., Gonzalvo B., Bossuet L. JTAG Fault Injection Attack. *IEEE Embed. Syst. Lett.* 2018;10(3):65–68. https://doi.org/10.1109/LES.2017.2771206
16. Abdalhag B., Awad A., Hawash A. A fast Binary Decision Diagram (BDD)-based reversible logic optimization engine driven by recent meta-heuristic reordering algorithms. *Microelectron. Reliab*. 2021;123:114168. https://doi.org/10.1016/j.microrel.2021.114168
17. Pevtsov E.F., Demenkova T.A., Shnyakin A.A. Design for Testability of Integrated Circuits and Project Protection Difficulties. *Russ. Technol. J*. 2019;7(4):60–70 (in Russ.). https://doi.org/10.32362/2500-316X-2019-7-4-60-70

18. Kuo M.-H., Hu Ch.-M., Lee K.-J. Time-Related Hardware Trojan Attacks on Processor Cores. In: *IEEE International Test Conference in Asia* (*ITC-Asia*). IEEE; 2019. https://doi.org/10.1109/ITC-Asia.2019.00021
19. Komolov D., Zolotukho R. Using special memory chips to ensure FPGA copy protection. *Komponenty i tekhnologii = Components & Technologies*. 2008;12:24–26 (in Russ.). Available from URL: https://kit-e.ru/wp-content/uploads/2008_12_24.pdf
20. Becker A., Hu W., Tai Y., Brisk P., Kastner R., Ienne P. Arbitrary Precision and Complexity Tradeoffs for Gate-Level Information Flow Tracking. In: *Proceedings of the 54th ACM/EDAC/IEEE Design Automation Conference* (*DAC*). IEEE, 2017. Part 128280. https://doi.org/10.1145/3061639.3062203
21. Polychronou N.F., Thevenon P.H., Puys M., Beroulle V. A Comprehensive Survey of Attacks without Physical Access Targeting Hardware Vulnerabilities in IoT/IIoT Devices, and Their Detection Mechanisms. *ACM Trans. Design Automat. Electron. Syst*. 2022;27(1):1–35. https://doi.org/10.1145/3471936
22. Erata F., Deng Sh., Zaghloul F., Xiong W., Demir O., Szefer J. Survey of Approaches and Techniques for Security Verification of Computer Systems. *ACM J. Emerg. Technol. Comput. Syst.* 2022;1(1):Article 1. https://doi.org/10.1145/3564785
23. Yang X., Zhao D., Jiang Y., Zhang X., Yuan Y. Fault Simulation and Formal Analysis in Functional Safety CPU FMEDA Campaign. *J. Phys.: Conf. Ser*. 2021;1769:012061. https://doi.org/10.1088/1742-6596/1769/1/012061

СПИСОК ЛИТЕРАТУРЫ

1. Smetana D. FPGA-Enabled Trusted Boot Is Part of Building Security into Every Aspect of Trusted Computing Architectures. *Military & Aerospace Electronics Journal*. September 25, 2019. URL: https://www.militaryaerospace.com/trusted-computing/article/14040672/trustedcomputing-embedded-computing-realworld
2. Сесин И.Ю., Болбаков Р.Г. Сравнительный анализ методов оптимизации программного обеспечения для борьбы с предикацией ветвлений на графических процессорах. *Russ. Technol. J*. 2021;9(6):7–15. https://doi.org/10.32362/2500-316X-2021-9-6-7-15
3. Shayan M., Basu K., Karri R. Hardware Trojans Inspired Hardware IP Watermarks. *IEEE Design & Test*. 2019;36(6):72–79. https://doi.org/10.1109/MDAT.2019.2929116
4. Hennessy J.L., Patterson D.A. A new golden age for computer architecture: Domain-specific hardware/software co-design, enhanced security, open instruction sets, and agile chip development. In: *Proceedings of the 2018 ACM/IEEE 45th Annual International Symposium on Computer Architecture* (*ISCA*). IEEE; 2018. https://doi.org/10.1109/ISCA.2018.00011
5. Li D., Zhang Q., Zhao D., Li L., He J., Yuan Y., Zhao Y. Hardware Trojan Detection Using Effective Property-Checking Method. *Electronics*. 2022;11(17):2649. https://doi.org/10.3390/electronics11172649
6. Алехин В.А. Проектирование электронных систем с использованием SystemC и SystemC–AMS. *Russ. Technol. J*. 2020;8(4):79–95. https://doi.org/10.32362/2500-316X-2020-8-4-79-95
7. Yang K., Zhang K., Ren J., Shen X. Security and privacy in mobile crowdsourcing: Challenges and opportunities. *IEEE Commun. Mag*. 2015;53(8):75–81. https://doi.org/10.1109/MCOM.2015.7180511
8. Lou X., Zhang T., Jiang J., Zhang Y. *A Survey of Microarchitectural Side-channel Vulnerabilities, Attacks and Defenses in Cryptography*. Vol. 1. No. 1. March 2021. URL: https://arxiv.org/pdf/2103.14244
9. Skorobogatov S., Woods C. Breakthrough Silicon Scanning Discovers Backdoor in Military Chip. In: Prouff E., Schaumont P. (Eds.). *Cryptographic Hardware and Embedded Systems – CHES 2012*. *Lecture Notes in Computer Science*. 2012. V. 7428. Berlin, Heidelberg: Springer. https://doi.org/10.1007/978-3-642-33027-8_2
10. Tasiran S., Keutzer K. Coverage metrics for functional validation of hardware designs. *IEEE Des. Test. Comput.* 2001;18(4):36–45. https://doi.org/10.1109/54.936247
11. Mukhopadhyay D., Chakraborty R.S. *Hardware Security: Design, Threats, and Safeguards.* CRC Press; 2014. 542 p. ISBN 978-1-4398-9584-9
12. Тарасов И.Е. *ПЛИС Xilinx. Языки описания аппаратуры VHDL и Verilog, САПР, приемы проектирования.* М.: Горячая линия – Телеком; 2024. 538 с. ISBN 978-5-9912-0802-4
13. Turkington K., Masseios K., Constantinides G.A., Leong P. FPGA Based Acceleration of the Linpack Benchmark: A High Level Code Transformation Approach. In: *2006 International Conference on Field Programmable Logic and Applications*. IEEE; 2007. https://doi.org/10.1109/FPL.2006.311240
14. Tamuly S., Joseph A. Chandrasekharan J. Deep Learning Model for Image Classification. In: Smys S., Tavares J., Balas V., Iliyasu A. (Eds.). *Computational Vision and Bio-Inspired Computing. ICCVBIC 2019. Advances in Intelligent Systems and Computing*. Springer, Cham; 2019. V. 1108. P. 312–320. https://doi.org/10.1007/978-3-030-37218-7_36
15. Majeric F., Gonzalvo B., Bossuet L. JTAG Fault Injection Attack. *IEEE Embed. Syst. Lett.* 2018;10(3):65–68. https://doi.org/10.1109/LES.2017.2771206
16. Abdalhag B., Awad A., Hawash A. A fast Binary Decision Diagram (BDD)-based reversible logic optimization engine driven by recent meta-heuristic reordering algorithms. *Microelectron. Reliab*. 2021;123:114168. https://doi.org/10.1016/j.microrel.2021.114168
17. Певцов Е.Ф., Деменкова Т.А., Шнякин А.А. Тестопригодное проектирование интегральных схем и проблемы защиты проектов. *Russ. Technol. J*. 2019;7(4):60–70. https://doi.org/10.32362/2500-316X-2019-7-4-60-70
18. Kuo M.-H., Hu Ch.-M., Lee K.-J. Time-Related Hardware Trojan Attacks on Processor Cores. In: *IEEE International Test Conference in Asia* (*ITC-Asia*). IEEE; 2019. https://doi.org/10.1109/ITC-Asia.2019.00021

19. Комолов Д., Золотухо Р. Использование микросхем специальной памяти для обеспечения защиты ПЛИС FPGA от копирования. *Компоненты и технологии*. 2008;12:24–26. URL: https://kit-e.ru/wp-content/uploads/2008_12_24.pdf
20. Becker A., Hu W., Tai Y., Brisk P., Kastner R., Ienne P. Arbitrary Precision and Complexity Tradeoffs for Gate-Level Information Flow Tracking. In: *Proceedings of the 54th ACM/EDAC/IEEE Design Automation Conference* (*DAC*). IEEE, 2017. Part 128280. https://doi.org/10.1145/3061639.3062203
21. Polychronou N.F., Thevenon P.H., Puys M., Beroulle V. A Comprehensive Survey of Attacks without Physical Access Targeting Hardware Vulnerabilities in IoT/IIoT Devices, and Their Detection Mechanisms. *ACM Trans. Design Automat. Electron. Syst*. 2022;27(1):1–35. https://doi.org/10.1145/3471936
22. Erata F., Deng Sh., Zaghloul F., Xiong W., Demir O., Szefer J. Survey of Approaches and Techniques for Security Verification of Computer Systems. *ACM J. Emerg. Technol. Comput. Syst.* 2022;1(1):Article 1. https://doi.org/10.1145/3564785
23. Yang X., Zhao D., Jiang Y., Zhang X., Yuan Y. Fault Simulation and Formal Analysis in Functional Safety CPU FMEDA Campaign. *J. Phys.: Conf. Ser*. 2021;1769:012061. https://doi.org/10.1088/1742-6596/1769/1/012061

About the authors

Evgeniy F. Pevtsov, Cand. Sci. (Eng.), Director of Center for the Design of Integrated Circuits, Nanoelectronics Devices and Microsystems, MIREA – Russian Technological University (78, Vernadskogo pr., Moscow, 119454 Russia). E-mail: pevtsov@mirea.ru. Scopus Author ID 6602652601. ResearcherID M-2709-2016, RSCI SPIN-code 1410-2483, http://orcid.org/0000-0001-6264-1231

Tatyana A. Demenkova, Cand. Sci. (Eng.), Associated Professor, Computer Technology Department, Institute of Information Technologies, MIREA – Russian Technological University (78, Vernadskogo pr., Moscow, 119454 Russia). E-mail: demenkova@mirea.ru. Scopus Author ID 57192958412, ResearcherID AAB-3937-2020, RSCI SPIN-code 3424-7489, http://orcid.org/0000-0003-3519-6683

Alexander O. Indrishenok, Postgraduate Student, Computer Technology Department, Institute of Information Technologies, MIREA – Russian Technological University (78, Vernadskogo pr., Moscow, 119454 Russia). E-mail: indrishenoksasha@mail.ru. RSCI SPIN-code 2308-7140, http://orcid.org/0000-0003-1471-9043

Vladimir V. Filimonov, Senior Lecturer, Department of Physics and Technical Mechanics, Institute for Advanced Technologies and Industrial Programming, MIREA – Russian Technological University (78, Vernadskogo pr., Moscow, 119454 Russia). E-mail: filimonov@mirea.ru. Scopus Author ID 7102525379. http://orcid.org/0000-0003-1118-6608

Об авторах

Певцов Евгений Филиппович, к.т.н., директор структурного подразделения «Центр проектирования интегральных схем, устройств наноэлектроники и микросистем», ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: pevtsov@mirea.ru. Scopus Author ID 6602652601. ResearcherID M-2709-2016, SPIN-код РИНЦ 1410-2483, http://orcid.org/0000-0001-6264-1231

Деменкова Татьяна Александровна, к.т.н., доцент, кафедра вычислительной техники, Институт информационных технологий, ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: demenkova@mirea.ru. Scopus Author ID 57192958412, ResearcherID AAB-3937-2020, SPIN-код РИНЦ 3424-7489, http://orcid.org/0000-0003-3519-6683

Индришенок Александр Олегович, аспирант, кафедра вычислительной техники, Институт информационных технологий, ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: indrishenoksasha@mail.ru. SPIN-код РИНЦ 2308-7140, http://orcid.org/0000-0003-1471-9043

Филимонов Владимир Викторович, старший преподаватель, кафедра физики и технической механики, Институт перспективных технологий и индустриального программирования, ФГБОУ ВО «МИРЭА – Российский технологический университет» (119454, Россия, Москва, пр-т Вернадского, д. 78). E-mail: filimonov@mirea.ru. Scopus Author ID 7102525379, http://orcid.org/0000-0003-1118-6608

Translated from Russian into English by Lyudmila O. Bychkova
Edited for English language and spelling by Dr. David Mossop