Formation of the Interval Stego Key for the Digital Watermark Used in Integrity Monitoring of FPGA-based Systems

Kostiantyn Zashcholkin[1][0000-0003-0427-9005], Oleksandr Drozd[1][0000-0003-2191-6758], Olena Ivanova[1][0000-0002-4743-6931] and Pavlo Bykovyy[2][0000-0002-5705-5702]

[1]Odessa National Polytechnic University, Odessa, Ukraine
[2]Ternopil National Economic University, Ternopil, Ukraine
const-z@te.net.ua
drozd@ukr.net
en.ivanova.ua@gmail.com
pb@tneu.edu.ua

Abstract. The operation of programmable computer systems is determined by their program code. Possibilities of maliciously changing program code potentially pose a security risk. Therefore, monitoring the program code integrity is one of the main components of security for programmable systems. This paper is devoted to program code integrity monitoring of computer systems built on the FPGA chips. Integrity monitoring methods are considered, within which monitoring data are embedded into the program code in the form of a digital watermark. Such digital watermark does not affect the operation of the FPGA and does not change the characteristics of the system. The advantages of this approach are that the fact of the presence of monitoring data in the program code and the fact of performing integrity monitoring is hidden from an outside observer. The paper notes the problem of the need to recovery the initial state of program code when performing integrity monitoring. To perform this procedure, the digital watermark must contain the data necessary for recovery. The effective volume of a digital watermark depends on the size and structure of the FPGA program code, as well as on the limitations defined by the watermark embedding key (stego key). Most of this volume is occupied by the data necessary to recovery the initial state. Under these conditions, there is often a shortage of the effective volume of a digital watermark for storing monitoring data.

The paper proposes a solution to this problem due to a new approach to the formation of a stego key for embedding a digital watermark in the FPGA program code. An experimental assessment of the approach proposed in the paper is performed. The advantages of the proposed approach are shown in comparison with the existing methods of embedding the digital watermark in the FPGA program code.

Keywords: Integrity Monitoring, Digital Watermarks, FPGA-Based Systems, LUT-Oriented Architecture, Program Code of FPGA

1 Introduction

Most modern computer systems contain programmable components: microprocessors, microcontrollers, programmable logic integrated circuits [1, 2]. The operation of systems of this kind is ensured both by the physical links of the components, and by a set of software codes that configure the components. Changing the program code of programmable components leads to a change in the operation of systems built on the basis of such components. Therefore, an important constituent of computer systems security is to ensure the integrity of the program code for programmable components [3]. In this paper, we consider the problem of ensuring the integrity of the program code of FPGA (Field Programmable Gate Arrays) chips [4, 5]. FPGA is a two-dimensional matrix of elementary programmable units of several types [6].
The program code of the FPGA chips determines the configuration of each of the units in the matrix to perform a specific function, and also forms a system of links between the units. FPGA structure provides natural parallel computing processes. The calculations within this structure are distributed in the space of the chip [7, 8]. Because of this, FPGAs differ from microprocessors in significantly greater performance.

High performance is the reason for the frequent use of FPGA in safety-critical systems – systems that control high-risk technical objects [9-11]. Under these conditions, the integrity of the FPGA program code is one of the safety factors of this kind of system.

2 Literature Review and Goal of the Paper

The most commonly used approaches to monitoring the integrity of program code are based on the use of a hash sums [12]. Hash sums for these purposes are calculated using cryptographic hash functions [13]. The disadvantage of traditional methods for monitoring the integrity of program codes is that the hash sums is stored either openly or can be detected in the structure of the information object as a result of its analysis.
So, an approach is known [14] in which the hash sum is placed in memory next to the program code. Another frequently used approach [15] is the inclusion of a hash sum in the information object of the program code as one of its fields. These approaches do not make it possible to hide the fact that integrity monitoring is performed, and do not make it possible to hide the monitoring information.

There is also an approach [16], in the framework of which, the hash sum is attached to the information object of the program code not in open but in encrypted form. This approach hides monitoring information, but makes open to the outside observer the very fact that integrity monitoring is performed. This leads to the possibility of using a sufficiently wide range of techniques to discover and falsify the hash sum.

There is an advanced approach to storing a hash sum that eliminates the above disadvantages. This approach consists in the fact that the hash sum is not attached to the information object of the program code, but is embedded in it in the form of a digital watermark (DWM) [17, 18]. DWM is embedded in an information object using digital steganography methods [19]. Several methods [20, 21] have been developed for embedding DWM into FPGA program code. These methods are based on the use of LUT (Look Up Table) units program codes for embedding the DWM bits. The implementation of these methods is performed using equivalent transformations. These transformations do not change the logic functions implemented by the LUT units, and do not affect the operation of the FPGA.

In DWM-based methods for integrity monitoring, there is a need to recovery of initial state for the information object of the program code at the time of monitoring. In order to recovery the initial state, in addition to the monitoring data, DWM must contain data on which recovery can be performed. The effective volume of DWM depends on the size and structure of the FPGA program code, as well as on the restrictions determined by the DWM embedding stego key. Most of this volume is occupied by data, necessary to recovery the initial state. Under these conditions, there is often a shortage of the effective volume of a digital watermark for storing monitoring data.

The goal of this paper is to increase the effective volume of DWM used in integrity monitoring of FPGA program code.

3 Analysis of Factors that Affect the Effective Volume of a Monitoring Digital Watermark

Factor of the embedding path length. Within the framework of the monitoring methods under consideration, the DWM contains three fields: M_{ISRec} – information necessary to recovery the initial state of the FPGA program code at the time the integrity monitoring was performed; *Hash* – hash sum; *S* – is the service field needed to define the boundaries of the M_{ISRec} and *Hash* fields.

DWM is located in the FPGA code space along the embedding path. The amount of bits of the embedding path depends on the size of the FPGA program code and on the stego key.

The effective volume of DWM is the volume available in it to store the monitoring hash sum:

$$L_{Hash} = L_{EmbPath} - L_{ISRec} - L_S; \tag{1}$$

where L_{Hash} – effective volume of DWM, expressed in amount of bits; $L_{EmbPath}$ – amount of LUT units, which are along the embedding path (the amount of bits that are available for DWM embedding); L_{ISRec} and L_S – the lengths of fields M_{ISRec} and *S*, respectively.

In the process of embedding DWM, the program code of only part of the LUT FPGA units undergoes a change. These LUT units form an embedding path. To recovery the initial state of the program code, it is necessary to recovery only the initial state of the LUT units, which are along the embedding path.

The main approach used to recovery the initial state: 1) lossless compression [22, 23] of the target bits of program code in the LUT units, which are along the embedding path; 2) putting the compression results in the MISRec DWM field. When using this approach, expression (1) takes the following form:

$$L_{Hash} = L_{EmbPath} - LCom(EmbPath) - L_S; \qquad (2)$$

where $LCom(EmbPath)$ – target bits compression result length.
The effective volume of DWM required to perform integrity monitoring must be at least 128 (160 or 256) bits – the size of the most widely used hash sums. As a result:

$$L_{EmbPath} - LCom(EmbPath) - L_S \geq (128 \text{ or } 160 \text{ or } 256). \qquad (3)$$

The feasibility of relation (3) depends on two factors: the length of field LEmbPath and the compression ratio provided by the selected lossless compression method. The compression ratio also depends on the length of field LEmbPath, and in addition it depends on the data in this field and on the compression method used. An approach to the use of multi-method compression was proposed in [24]. This approach gives an increase in the compression ratio in this situation, however, it is applicable only in the case when a unique stego key is generated for each act of successful embedding of control data into the FPGA program code. In the case when it is necessary to use a universal stego key for many containers, this approach cannot be applied. In addition, with a low LEmbPath value, even this approach does not allow us to obtain the feasibility of relation (3).

Thus, when using effective compression methods, the main reserve for increasing the difference LEmbPath – LCom(EmbPath) for the feasibility of relation (3) is to increase the length of the embedding path (LEmbPath). However, an increase in the length of the embedding path leads to an increase in the amount of LUT units whose program code changes. Such an increase could potentially reduce the resistance of the monitoring system to attacks on monitoring data. This reduces the advantages of a steganographic approach to storing monitoring data.

Conclusion: an increase in the length of the embedding path is a positive factor for increasing the effective volume of DWM, but it is a potentially negative factor for the resistance of the monitoring system to attacks. Therefore, the length of the embedding path should be chosen exactly so as to ensure relation (3), but no more.

Factor of the stego key. The length of the embedding path depends on the stego key and on the structure of the links between the LUT units. The basic version of a stego key for a LUT-container (FPGA program code) is a set of the following components:

$$Skey = <EnumRule, DThreshold, AddrRule> \qquad (4)$$

where *EnumRule* – rule that determines the order of the enumeration for units in the LUT container to obtain an ordered set of units that are part of the embedding path; *DThreshold* – parameter that determines the limit on the number of connections to the outputs of the LUT units included in the embedding path; *AddrRule* – rule that determines the address of target bit in program code for each LUT unit of the embedding path.

The length of the embedding path depends on the parameters *EnumRule* and *DThreshold*. The *EnumRule* parameter can be described as a fixed value, an iterative rule, or a template. This parameter determines the numbering distance between the

LUT units that are included in the embedding path. The *DThreshold* parameter is a numeric threshold that is used when deciding whether to include the LUT units in the embedding path. This threshold sets the allowable amount of LUT units connected to the output of the units, regarding which such a decision is made. Reducing the EnumRule and the DThreshold leads to an increase in the length of the embedding path, as well as to an increase in the volume of modifiable FPGA program code.

Factor of stego key usage mode. Two modes of stego key use are possible.

1. Specialized stego key mode. In this mode, a stego key is generated for each individual FPGA container and DWM. If this mode is used, the stego key must be transferred to the integrity monitoring system for each monitored object via some reliable communication channel. When using this mode, you can select the key parameters that provide the most suitable length of the embedding path. However, for integrity monitoring, this mode is not practical to use because of the complexity of key distribution.
2. Universal stego key mode. In this mode, the stego key is generated once, after which it is used by the DWM embedding modules and integrity monitoring modules. This mode is the most commonly used in integrity monitoring systems. The problem of stego key formation in this mode is as follows. A situation is possible in which, when using the universal key for the next container, it will not be possible to ensure the feasibility of condition (3). In this case, it is possible to reduce the size of the hash sum, but this reduces the resistance of the monitoring system to attacks on monitoring information.

Thus, we can state that the effective DWM volume and the feasibility of condition (3) depend on the length of the embedding path. This length depends on the values of the stego key components. When using the universal stego key mode, there is a contradiction between the effective DWM volume and the fraction of LUT units included in the embedding path.

4 The Proposed Approach to Increase the Effective Volume of DWM by Using Interval Stego Key

We propose a method for formation a stego key that allows to adapt the effective DWM volume to the structure of the LUT container.

The first principle of the method is that instead of the point values of the components of the universal stego key, it is proposed to use value intervals. It is proposed to use interval values for components on whose value the length of the embedding path depends. For each such $param_i$ component, an interval of $param_{i_{min}} \dots param_{i_{max}}$ values is formed. Moreover, smaller values from these intervals correspond to a shorter embedding path and a smaller fraction of the target LUTs.

At the first attempt to embedding, the minimum value from the interval is selected as the corresponding component of the stego key. If in this case relation (3) is not true, then the next greater value is selected from the interval. The component *priority* of the key determines the order of increasing interval components.

If relation (3) is not true when the maximum values $param_{i_{max}}$ of all interval components are reached, then the actions regulated by the second principle of the method are applied. For the basic method of embedding DWM, the following components are proposed as interval components of the stego key: *EnumRule* and *DThreshold* (4).

The second principle of the method is that for the embedding and extraction of DWM, a sequence of methods *Methods* = <m_1, m_2, … m_n> is used (this sequence is a component of the stego key). The first time you try to embedding, the first method of sequence is used. If the result of the embedding (taking into account the first principle of the proposed method) does not lead to the truth of relation (3), then we proceed to the next method of the sequence *Methods*. If using this procedure the last method of the sequence *Methods* is reached and relation (3) is still not true, then the decision on further actions is applied based on the third principle of the method.

The third principle of the method determines the way of deciding on further actions if the use of any of the methods in *Methods* did not lead to the truth of relation (3). Special component m_{final} of the stego key indicates whether to apply the method of preliminary preparation of the information object [20]. Thus, the m_{final} parameter determines the admissibility of losing the initial state of an information object by switching to the functional equivalent of this state. If the m_{final} parameter is equal to zero, then the use of the preliminary preparation method is invalid. In this case, the traditional approach is used, which consists in reducing the size of the hash sum.

In accordance with the first three principles of the method, an interval stego key is defined as a tuple of the following form:

$$Skey = \langle EnumRule, DThreshold, priority, Methods, m_{final}, AddrRule \rangle \qquad (5)$$

where *EnumRule* = (*EnumRule*$_{min}$ … *EnumRule*$_{max}$); *DThreshold* = (*DThreshold*$_{min}$ … *DThreshold*$_{max}$); *Methods* = <m_1, m_2, … , m_n>; m_i = <*MethodId*$_i$, *MethodParams*$_i$>.

The fourth principle of the method determines how the Methods sequence is formed. The methods of this sequence should be arranged in order of potential increase in the embedding path length or the DWM effective volume.

The fifth principle of the method determines the way of extraction DWM in the case of the use of the interval stego key. When extraction, the procedure of searching for specific values (which were used during the embedding) from the intervals is performed. The selection of these values is carried out in accordance with the procedure defined by the first principle of the method. After that, DWM extraction is performed. Further, based on the information contained in the service field *S*, DWM is divided into three fields: *S*, M_{ISRec} and *Hash*. Next, the M_{ISRec} field is decompressed. After this, the truth of the relation is checked:

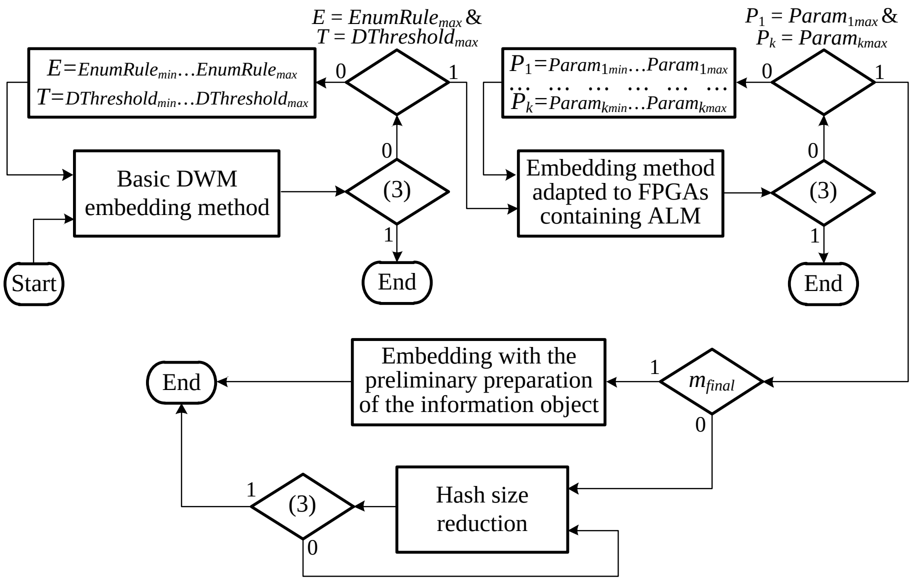
$$LDecom(M_{ISRec}) = L_{EmbPath} = L_S + L_{ISRec} + L_{Hash} \qquad (6)$$

where $LDecom(M_{ISRec})$ – length of field M_{ISRec} decompression result; $L_{EmbPath}$ – length of embedding path; L_S, L_{ISRec}, L_{Hash} – lengths of the respective fields of DWM.

In fig. 1 shows a flowchart for the implementation of the proposed embedding method. The flowchart is shown for the case of using two methods included in the Methods component: the basic DWM embedding method and the embedding method adapted to FPGAs containing Adaptive Logic Modules (ALM).

5 Experimental Assessment of the Proposed Method

The method proposed in the work was implemented as a software application. For this, software modules were used that implement: a) the basic DWM embedding method; b) the embedding method adapted to FPGAs containing ALM; c) the method of embedding DWM with the preliminary preparation of the information object of the FPGA program code. The functioning procedure of the developed software application is as follows. The application receives the interval stego key. After that, values are sequentially selected from the interval components and transferred to the corresponding DWM embedding modules. In the process of embedding, condition (3) is checked. In general, the functioning of the developed application corresponds to the flowchart shown in Fig. 1.



$E = EnumRule_{max}$ & $T = DThreshold_{max}$
E=$EnumRule_{min}$...$EnumRule_{max}$
T=$DThreshold_{min}$...$DThreshold_{max}$
0
1
$P_1 = Param_{1max}$ & $P_k = Param_{kmax}$
P_1=$Param_{1min}$...$Param_{1max}$
... ...
P_k=$Param_{kmin}$...$Param_{kmax}$
Basic DWM embedding method
(3)
Embedding method adapted to FPGAs containing ALM
Start
End
Embedding with the preliminary preparation of the information object
m_{final}
Hash size reduction

Fig. 1. Flowchart for the implementation of the proposed method

In the environment of the developed software, an experiment was carried out necessary to assessment the proposed method. The initial data of the experiment were 6 FPGA projects. These projects have a different amount of FPGA hardware resources used and different design missions. The synthesis of the projects was carried out in the CAD Intel Quartus Prime [25]. As target FPGA chips Intel Cyclone IV [26, 27] and Cyclone V [28] were applied. The experiment process was as follows. Two stego keys were formed: point key and interval key. As parameters of the point key, the minimum values of the corresponding interval components of the interval key were used. Next, DWM was formed using both keys. After that, the effective DWM volume was calculated. If the volume turned out to be sufficient to store the hash sum, then DWM was embedding in the container. After that, DWM extraction was performed and the possibility of dividing it into separate fields was checked.

Table 1 shows that when using the traditional approach for projects 1, 2, the size of the *Hash* field is insufficient to store the hash amount. When using an interval stego key with the parameter $m_{final} = 0$ (the method of preliminary preparation of the information object is not applied), the size of the *Hash* field increases, but is still insufficient. This is due to too few LUT units in the project. However, with the value of the parameter $m_{final} = 1$ (the method of preliminary preparation of the information object is allowed), the size of the Hash field allows you to store the hash sum in DWM. For projects 3–6 $m_{final} = 0$, because a sufficient size of the *Hash* field is provided without using the method of preliminary preparation of the information object.
Project 3 shows that the *Hash* field is not large enough when using the point stego key. When switching to the interval stego key, this field is enough to store the MD5 hash sum (the size is 128 bits). Projects 5 and 6 show an increase in the length of the *Hash* field when switching to the interval stego key. Due to this, it becomes possible to store a hash of a larger length: SHA1 (the size is 160 bits) for project 5 and SHA2 (the size is 256 bits) for project 6. Thus, the results of experimental assessment of the proposed method show that the method allows to increase the effective volume of DWM. For projects with a very small amount of LUT units, an increase is achieved by switching to the method of preliminary preparation of the information object. For other projects, an increase in effective volume is provided by using interval parameters of the stego key instead of point parameters.

Table 1. Experiment Results

Project No	**Total amount of LUT units**	**Traditional approach**			**Proposed method**			
		Amount of LUT units, which are along the embedding path	**Maximum possible amount of hash sum bits**	**Possibility to use hash functions (size)**	**Amount of LUT units, which are along the embedding path**	m_{final}	**Maximum possible amount of hash sum bits**	**Possibility to use hash functions (size)**
1	780	143 (18,3%)	6	—	176 (22,5%)	0	7	—
						1	143	MD5 (128)
2	4212	904 (21,4%)	35	—	1011 (24,1%)	0	43	—
						1	904	SHA2(256) or SHA1(160) or MD5 (128)
3	9856	2620 (26,5%)	119	—	3371 (34,2%)	0	130	MD5 (128)
4	10074	2851 (28,3%)	133	MD5 (128)	3275 (32,5%)	0	138	MD5 (128)
5	11839	3179 (26,8%)	141	MD5 (128)	3932 (33,2%)	0	181	SHA1(160) or MD5 (128)
6	15043	4811 (31,9%)	168	SHA1(160) or MD5(128)	6155 (40,9%)	0	262	SHA2(256) or SHA1(160) or MD5 (128)

6 Conclusions and Directions of the Further Research

The paper proposes a method that increases the effective volume of DWM used in FPGA program code integrity monitoring tasks. If the effective volume is small, the hash sum cannot be stored in the DWM. The proposed method eliminates this disadvantage.

The proposed method differs from existing methods in that instead of the point components of the stego key, interval components are used. This allows the use of a universal stego key for many containers. However, the key components in each case are adapted to obtain the most suitable effective DWM volume. When you embedding DWM, the values of the interval components are iterated sequentially. At the same time, on each iteration, the sufficiency of the effective DWM volume is checked to store the hash sum. When extracting DWM, the values of the interval components of the key are sequentially enumerated. The paper proposes a relation that allows you to make a decision about the interval element that was used when DWM embedding.
An experimental evaluation of the proposed method showed its effectiveness in comparison with existing methods. Namely, the method provides an effective volume of DWM sufficient to store hash sums (in integrity monitoring tasks). For FPGA projects that were used in the experiment, the average increase in the effective volume of the DWM was 22.8%. This, in most experimental cases, made it possible to use a hash sum with greater cryptographic strength than before applying the proposed method.

Application of the proposed method simultaneously with an increase in the effective volume of DWM increases the amount of LUT units, the program code of which changes during the embedding of DWM. We assume that this could potentially reduce the resistance of such an embedding to steganalysis. At the moment, there are no well-developed methods of steganalysis for FPGA containers. Conventional steganalysis methods are not effective for containers of this kind. Because of this, a decrease in resistance to traditional stegoanalysis can be considered insignificant. However, with the development of stegoanalysis methods oriented to FPGA containers, the problem of reducing resistance when applying the proposed method will become relevant. We consider that this question creates the direction for further research.

References

1. Andina, J.: FPGAs: Fundamentals, Advanced Features, and Applications in Industrial Electronics. CRC Press, Boca Raton, USA (2017). doi: 10.1201/9781315162133
2. Raj, A.: FPGA-Based Embedded System Developer's Guide. CRC Press, Boca Raton, USA (2018). doi: 10.1201/9781315156200
3. Lysenko, S., Bobrovnikova, K., Savenko, O.: A Botnet Detection Approach Based on the Clonal Selection Algorithm. In: DESSERT 2018 – 9th IEEE International Conference on Dependable Systems, Services and Technologies, pp. 449-453. Kyiv, Ukraine (2018) doi: 10.1109/DESSERT.2019.8770035
4. Vanderbauwhede, W., Benkrid, K. (Eds.): High-performance computing using FPGAs. Springer, New-York, USA (2016)
5. Unsalan, C., Tar, B.: Digital System Design with FPGA. New-York, McGraw-Hill (2017)
6. Tyurin, S.: LUTs Sliding Backup. IEEE Transactions on Device and Materials Reliability. 19, No 1, 221–225 (2019). doi: 10.1109/TDMR.2019.2898724
7. Amano, H.: Principles and Structures of FPGAs. Springer, Singapore (2018). doi: 10.1007/978-981-13-0824-6

8. Waidyasooriya, H., Hariyama, M., Uchiyama, K.: Design of FPGA-Based Computing Systems with OpenCL. Springer, Cham, Switzerland (2018). doi: 10.1007/978-3-319-68161-0
9. Drozd, O., Kuznietsov, M., Martynyuk, O., Drozd, M.: A method of the hidden faults elimination in FPGA projects for the critical applications. In: DESSERT 2018 – 9th IEEE International Conference on Dependable Systems, Services and Technologies, pp. 231-234. Kyiv, Ukraine (2018). doi: 10.1109/DESSERT.2018.8409131.
10. Drozd, O., Kharchenko, V., Rucinski, A. et. al.: Development of Models in Resilient Computing. In: DESSERT 2019 – 10th IEEE International Conference on Dependable Systems, Services and Technologies, pp. 2-7. Leeds, UK (2019) doi: 10.1109/DESSERT.2019.8770035.
11. Hovorushchenko, T, Pomorova, O.: Ontological approach to the assessment of information sufficiency for software quality determination. SEUR-WS, **1614**, 332–348 (2016)
12. Vacca, J.: Computer and information security handbook, 3rd edition. Morgan Kaufmann, Waltham, Mass (2017).
13. Bishop, M.: Computer Security. 2nd edition. Addison-Wesley, Boston, USA (2018).
14. Katz, J.: Introduction to Modern Cryptography, 2nd edition. CRC Press, USA (2018).
15. Conklin, W. et al.: Principles of Computer Security, 4th edition. McGraw-Hill, (2015).
16. Stallings, W.: Cryptography and Network Security: Principles and Practice, 7th edition. Pearson Education, UK, Harlow (2017)
17. Shih, F.: Digital Watermarking and Steganography: Fundamentals and Techniques. 2nd edition. CRC Press, Boca Raton, USA (2017)
18. Nematollahi, M., Vorakulpipat, C., Rosales, H.: Digital Watermarking: Techniques and Trends. Springer, Singapore (2017).
19. Yahya, A.: Steganography Techniques for Digital Images. Springer (2018)
20. Zashcholkin, K., Ivanova, O.: LUT-object Integrity Monitoring Methods Based on Low Impact Embedding of Digital Watermark. In: 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET-2018), pp. 519–523. Lviv-Slavske, Ukraine (2018). doi: 10.1109/TCSET.2018.8336255
21. Zashcholkin, K., Drozd, O.: The Detection Method of Probable Areas of Hardware Trojans Location in FPGA-based Components of Safety-Critical Systems. In: DESSERT-2018 – 9th IEEE International Conference on Dependable Systems, Services and Technologies, pp. 220–225. Kiev, Ukraine (2018). doi: 10.1109/DESSERT.2018.8409130
22. McAnlis, C., Haecky, A.: Understanding Compression: Data Compression for Modern Developers. O'Reilly Media, USA (2016).
23. Sayood, K. Introduction to Data Compression, 5th edition. Morgan Kaufmann (2018).
24. Drozd, A., Antoshchuk, S., Drozd, J., Zashcholkin, K. et. al.: Checkable FPGA Design: Energy Consumption, Throughput and Trustworthiness. In: Green IT Engineering: Social, Business and Industrial Applications, Studies in Systems, Decision and Control, V. Kharchenko, Y. Kondratenko, J. Kacprzyk (edits), vol. 171, pp. 73-94, Springer, Berlin, Heidelberg (2019). doi: 10.1007/978-3-030-00253-4_4.
25. Intel Quartus Prime. https://www.intel.com/content/www/us/en/software/programmable /quartus-prime/overview.html.
26. Intel Cyclone FPGA series. https://www.intel.com/content/www/us/en/products/programmable /cyclone-series.html.
27. Intel Cyclone IV Device Handbook, https://www.intel.com/content/dam/www /programmable /us/en/pdfs/ literature/hb/cyclone-iv/cyclone4-handbook.pdf.
28. Intel Cyclone V Device Handbook, https://www.intel.com/content/dam/www /programmable/us/en/pdfs/ literature/hb/cyclone-iv/cyclone5-handbook.pdf.