# Checkable FPGA-Based Components of Safety-Related Systems

Oleksandr Drozd[a], Kostiantyn Zashcholkin[a], Anatoliy Sachenko[b], Oleksandr Martynyuk[a], Olena Ivanova[a] and Julia Drozd[a]

[a] *Odessa Polytechnic State University, 1, Ave. Shevchenko, Odesa, 65044, Ukraine*
[b] *West Ukrainian National University, 11, Lvivska Str., Ternopil, 46009, Ukraine*

**Abstract**

The paper is devoted to the problem of checkability of circuits in FPGA components of safety-related systems, which are designed to operate in two modes: normal and emergency for providing their own functional safety and the safety of control facilities in order to prevent accidents and reduce losses in case of their occurrence. Functional safety is ensured through the use of fault-tolerant solutions that are sensitive to sources of multiple failures, including hidden faults. They can accumulate during a prolonged normal mode with limited checkability of the circuits and simultaneously manifest themselves with the beginning of the emergency mode. A fault-tolerant structure becomes fail-safe if it is checkable. The problem of hidden faults manifests itself in the memory of the LUT units of FPGA components with LUT-oriented architecture. The program code written in the memory of the LUT units is checked with a checksum, but it can be corrupted when reading its bits on the outputs of the LUT units. Bits observed only in emergency mode reduce the checkability of FPGA components and are potentially hazardous. Checkability can be increased by the operation of circuits on successively replaced versions of the program code that can be obtained for the same hardware implementation. Versions move potentially hazardous bits to checkable positions observed in normal mode. However, the set of these versions are significantly limited by connecting the inputs of the LUT units to the inputs of the FPGA component. The proposed method overcomes this limitation by introducing an additional scheme. Experimental studies of library FPGA designs show a low level of their checkability and efficiency of the proposed method, which provides totally checkable circuits.

**Keywords**

Safety-related system, normal and emergency modes, hidden faults, FPGA component, LUT-oriented architecture, memory bits of LUT unit, program code version, checkability

## 1. Introduction

FPGA design is becoming the most advanced direction in the development of digital components for computer systems. The advantages of FPGA design, first of all, are based on the combination of the technological hardware basis of circuit solutions and the possibility of their programming [1, 2].

Recognition of the advantages of FPGA design (Field Programmable Gate Array) is reflected in its widespread use in the most responsible areas, including the domain of critical application of computer systems for monitoring high-risk facilities [3].

Power plants and power grids, transport infrastructures, chemical and other high-risk industries are directly related to safety problems, which manifests itself in two aspects. On the one hand, these facilities provide energy, transport, food and other types of security important for the survival and

7th International Workshop on Theory of Reliability and Markov Modeling for Information Technologies (WS TheRMIT 2021 at the 17th International Conference ICTERI2021), September 28, 2021, Kherson, Ukraine
EMAIL: drozd@ukr.net (O. Drozd); const-z@te.net.ua (K. Zashcholkin); as@wunu.edu.ua (A. Sachenko); anmartynyuk@ukr.net (O. Martynyuk); en.ivanova.ua@gmail.com (O. Ivanova); yuliia.drozd@opu.ua (J. Drozd)
ORCID: 0000-0003-2191-6758 (O. Drozd); 0000-0003-0427-9005 (K. Zashcholkin); 0000-0002-0907-3682 (A. Sachenko); 0000-0003-1461-2000 (O. Martynyuk); 0000-0002-4743-6931 (O. Ivanova); 0000-0001-5880-7526 (J. Drozd)

development of mankind. For this reason, humanity is unable to refuse the further development of these objects. On the other hand, this development is accompanied by the quantitative and qualitative growth of high-risk objects, which become an integral part of our habitat and constantly increase their power, demonstrating it in the event of accidents [4, 5].

In these conditions, it is necessary to control the risks, which are assessed by two factors: the probability of an accident and the cost of its consequences [6, 7].

The development of high-risk facilities in the direction of increasing losses caused by accidents focuses attention on the problem of containing risks by reducing the probability of an accident. The solution to this problem, which is important for the survival of mankind, is provided by the development of information technologies implementing in computers, which, taking into account such specialization, acquire features that define them as safety-related systems. An important feature of critical systems is their designing for operation in two modes: normal and emergency [8, 9].

According to international standards, these instrumentation and control systems are aimed at ensuring the functional safety of the controlled object in combination with its own functional safety to prevent accidents and reduce losses in case of their occurrence [10].

The provision of functional safety, which is carried out on the basis of the use of fault-tolerant solutions, faces the problem of multiple failures, including hidden faults. Their accumulation during normal operation can lead to multiple failures in emergency mode. Faults can accumulate in a digital circuit due to its limited checkability as another manifestation of the features of modern safety-related systems. In FPGA components with a LUT-oriented (Look-Up Table) architecture, the lack of checkability manifests itself in the memory of the LUT units and does not allow fault-tolerant circuits to become fail-safe.

These studies are aimed at designing checkable circuits that convert fault-tolerant FPGA components into fail-safe ones. Section 2 examines related works that show the sources of multiple failures and disclose the impact of critical system features on the checkability and fail-safety of digital circuits. Section 3 describes the main provisions of the proposed method for designing checkable circuits for fault-tolerant FPGA components of safety-related systems. Section 4 demonstrates the use of the method on the example of iterative array multipliers implemented in FPGA projects. Section 5 is devoted to a discussion of the results obtained.

## 2. Related works

Ensuring functional safety is associated with the consideration of various factors of its violation, among which failures occupy a special place. The problem of failures is solved by using fault-tolerant structures, which in the design process acquire the ability to withstand a certain number of failures. Most often, a fault tolerant solution is limited to one or two failures that can be mitigated. This limitation is explained by considerations of cost savings and a significant decrease in the probability of the simultaneous occurrence of independent failures with an increase in their number [11, 12].

It should be noted that the low probability of the simultaneous occurrence of many failures, including independent ones, does not negate the likelihood of their simultaneous manifestation. For this reason, the greatest danger to fault-tolerant circuits and the functional safety provided by them comes from sources of multiple faults, for which the assumption of a low probability of multiple failures is violated.

International standards related to functional safety focus on one such source, which combines common cause failures [13].

This reason is the copying of erroneous decisions that can arise, for example, as a result of design errors. The standards propose limiting such copying through the development of multi-version technologies that reduce many common causes [14, 15].

It is important to note that versions are one of the features of the existing universe, since nothing in this world is stamped the same. This feature is demonstrated by majority structures, which are fault-tolerant with respect to the failure of one channel when they are built by stamping three channels. Fault tolerance of the majority system is achieved only because each of these channels has a natural version redundancy. They cannot be made the same. However, the available natural types of diversity are not enough to withstand common cause failures, for example, in the case of using the same

software that contains a design error in the channels. Therefore, the natural types of diversity are supplemented by the types created artificially in the process of development of multi-version technologies [16].

Ensuring functional safety requires close attention to the sources of multiple failures based on a deep analysis of already manifested sources and the search for new sources.

One of these sources is associated with deliberate malicious interference in computer systems through the organization of cyber-attacks. This source has been deeply studied by methods and means of information security, which, thus, manifests itself as an important component in ensuring the functional safety of critical systems, in particular, in maintaining their integrity [17, 18].

It should be noted that common cause failures are primarily a problem for fault-tolerant systems and are only indirectly related to functional safety. Multiple failures caused by cyber-attacks are also related to more than just critical systems. Moreover, all of these failures are not independent in nature.

The next source of multiple failures is inherent only in safety-related systems, although it has not yet been adequately reflected in the relevant international standards. This source is associated with the problem of hidden faults that can be independent and accumulate in safety-related systems during long normal mode. This mode can last for many years and decades. As a rule, detection of faults in a digital circuit is carried out by logical checking methods by identifying an error in the calculated result. In this case, the fault becomes hidden due to the lack of input data that can manifest it in the form of an error in the calculated result. In normal mode, the variety of input data is often limited by noise. The emergency mode increases the activity of the input data, which manifests itself in the expansion of the range of their change. The manifestation of accumulated faults on new input data can lead to numerous failures and disrupt the functional safety of the system in the most responsible emergency mode [19, 20].

Hidden faults are not a problem for conventional computers which work in the same operating mode. Indeed, the absence of input data showing a malfunction throughout the entire operating mode completely excludes the hidden fault from the computational process. Thus, hidden faults occupy a special place among multiple failures related to critical systems.

The presence of hidden faults in the circuit indicates its insufficient checkability. Typically, digital circuits are characterized by logical checkability, i.e., suitability for logical checking.

The limitation of checkability can have various reasons. Testability, which is the simplest form of checkability, can be limited by the structural redundancy of a digital circuit. The ability to write tests for detection of the faults is completely determined by the circuit design. Therefore, testability is structural checkability [21, 22].

In the operating mode, the checkability of a digital circuit is limited not only by the structure of the digital circuit, but also by its input data, the variety of which may be insufficient for the manifestation of some faults. Methods and means of on-line testing using logical checking [23, 24] can detect faults in digital circuits only in the case of calculating an erroneous result, i.e., within the framework of logical checkability.

Different conditions for the functioning of digital circuits in the normal and emergency modes of the safety-related system are determined, first of all, by different input data, which also determine the difference in the checkability of the circuit for these modes.

The increase in checkability in the emergency mode, which occurs due to the greater variety of input data, leads to the problem of hidden faults, which have more opportunities for their manifestation in the form of errors in the result.

Focusing the problem of hidden faults only on safety-related systems does not mean that they are inextricably linked. At the same time, such a connection is certainly inherent in modern safety-related systems. To free critical systems from this problem, it is necessary to pay attention to its causes, which are of an evolutionary nature [25, 26].

The resource-based approach that analyzes the integration of the computer world into the natural one identifies the problem of hidden faults as a challenge of growth. The evolutionary development of models, methods and tools that make up the resources necessary to support integration processes can be represented by three levels: replication, diversification and self-sufficiency as the goal of development. Replication is characterized by the absence of rigid contact with the natural world in the conditions of open resource niches: ecological, market, technological and others, which allow churning out resources within these niches without restrictions. Integration is carried out at the

expense of productivity, the growth of which is stimulated up to the closure of resource niches as a result of their filling. From this point on, stamped clones lose the prospect of integration and can only survive by exhibiting characteristics that allow them to rise to the level of diversification. At this level, integration occurs at the expense of trustworthiness, which manifests itself in the adequacy of the natural world [27, 28].

In the computer world, all levels of resource development are represented, but replication dominates. Open resource niches of performance and memory capacity of modern computers allow assembling software products from stamped redundant program modules [29, 30].

For such solutions, resource niches are closed in energy consumption with the advent of mobile computer systems that encourage the development of green technologies, tending to the level of diversification [31–33].

In hardware, replication is manifested in the growth of matrix structures from homogeneous elements. Matrix structures show a number of significant drawbacks due to their low level of development.

For example, an iterative array multiplier that performs an operation on $n$-bit binary codes in one clock cycle consists of $n^2$ operational elements, each of which is used for a small part of the clock cycle, since its duration is determined by the serial connection of $2n - 2$ elements. For $n = 64$, the useful use of each of the more than 4 thousand operational elements is only 0.8% [34, 35].

The rest of the time is spent on parasitic switching, which occurs due to the competition of signals propagating along paths of different paths. Parasitic transitions are quantitatively many times greater than functional switching and therefore mainly determine the dynamic component of power consumption, just as the static component is due to the large size of the matrices [36, 37].

The most significant drawback of matrix structures, which manifests itself in relation to safety-related systems, is their low checkability, due to the processing of data in parallel codes. The iterative array multiplier input word, composed of two $n$-bit binary codes, contains $2n$ bits and a variety of $2^{2n}$ values. The digital multiplier circuitry can accept input data that changes at the noise level and use only a few such values throughout normal mode.

The processing of data in sequential codes constitutes an input word of two bits, taking 4 different values, on which the complete checkability of the digital circuit is ensured. This way does not lead to a loss in the ratio of performance to complexity when performing bitwise pipelining, where the matrix structures of modern pipelines' sections are reduced to a single operational element. The truncated execution of arithmetic operations also increases the checkability of matrix structures and provides advantages in on-line testing [38, 39].

However, over several decades of dominance, matrix structures have created a powerful infrastructure for their resource support, including modern CAD systems [40, 41]. FPGA design offers built-in iterative array multipliers and accelerated addition of parallel binary codes, as well as library solutions, which are mainly represented by matrix circuits [42, 43].

Under these conditions, the problem of hidden faults associated with limited checkability of digital components in safety-related systems should be solved by raising matrix structures to the diversification level, using the capabilities of FPGA-designing with LUT-oriented architecture [44].

The increasing demands on the functional safety of critical systems require the development of checkable FPGA components that enable the transformation of fault-tolerant circuits into fail-safe solutions.

## 3. Main Provisions of the Method

The purpose of the method is to improve the checkability of digital circuits in FPGA components used of the safety-related systems. The problem of hidden faults is manifested in the distortion of the values of the bits read from the memory of the LUT units. The change of program code versions allows in the normal mode to address to the memory bits of the LUT unit, which in the original version are observed only with the beginning of the emergency mode. Improving circuit checkability is based on the use of the program code versions for the LUT-oriented architecture. The proposed method provides a choice of versions from the maximum possible set.

## 3.1. Prerequisites for the method

The initial data for the method are the features of the LUT-oriented architecture of FPGA components. The main feature consists in diversification of the FPGA project by dividing its resources into hardware and software parts. The hardware used to organize computations has a regular structure formed by a matrix of identical operational nodes, called configurable logical blocks or logical elements containing LUT units and one-bit registers [45].

The LUT unit contains SRAM memory, supplemented by a multiplexer for reading a bit from this memory to the output of the LUT unit. The bit is read at the address that is fed to the inputs of the LUT unit. The number of $m$ inputs is varied, defining an $m$-LUT with a memory size of $2^m$ bits and a $2^m$:1 multiplexer. The memory bits are numbered with an address code from 0 to $2^m - 1$. The widely used 4-LUT unit, containing 4 inputs: A, B, C D, addresses 16-bit SRAM memory with bit numbering from $0_{16}$ to $F_{16}$ using a 16:1 multiplexer. Bits $a$, $b$, $c$ and $d$ arriving at these inputs form the binary code of the $dcba_2$ address, which is also the bit number in the memory of the LUT unit. The 16:1 multiplexer is implemented as a tree of 2:1 multiplexers. FPGA project programming is performed by writing the program code from the configuration file to the memory of the LUT units.

The program code is verified using a checksum and therefore the SRAM of the FPGA project is checkable. However, the bits of this memory, read by the multiplexer, can be swapped for other bits as a result of an addressing error caused by faulty 2:1 multiplexer. An error occurs when the values in the bits located at the correct and erroneous addresses do not match. Otherwise, the switch fault remains hidden for this program code and can be manifested as an error at the output of the LUT unit only when this code is changed. Within the framework of the use of the same program code, the manifestation of the addressing error additionally depends on the input data of the circuit. In this case, the 2:1 multiplexer failure also becomes hidden in the absence of a corresponding erroneous memory access. Thus, the LUT memory of units, considered in conjunction with 2:1 multiplexers, can be a carrier of hidden fault in case of limited use of input data in the normal mode of safety-related systems.

In practice, a limited change in the input data in the normal mode is overcome by their manual regulation, which for power blocks of nuclear power plants is performed no more than once every six months. Experiments show that memory bits of LUT units, addressed in normal and emergency mode, can form non-overlapping sets. This eliminates the possibility of observing the bits used only in emergency mode in the period before the start of this mode and encourages the use of simulation mode for testing circuits in emergency conditions.

The proposed method uses version redundancy inherent in FPGA circuits with LUT-oriented architecture. This redundancy is manifested in the existence of many versions of the program code that support the specified functionality for the same hardware implementation of the FPGA component. The basis for creating versions is a pair of LUT units, connecting the output of the first unit of the pair to the input of the second unit. Each such pair allows to create two versions of the program code. The original version is supplemented by another version, which differs in the inverse value of the bit transmitted between the LUT units of the pair. The inverse value of this bit is provided by inverting the program code written in the memory of the first LUT unit of the pair. The inversion that occurs at the input of the second LUT unit of the pair is compensated for by changing the places in the bits of its memory. The change of places occurs between the bits, the numbers of which in the corresponding bit of the address take on the values 0 and 1.

The presence of two versions of the program code creates conditions for optimizing the circuit by inverting the memory of individual LUT units or moving their bits. It should be noted that versions can be created independently for each inverted input of the LUT unit, i.e., an input connected to the output of the previous unit. For example, a 4-LUT unit can have up to 16 code versions.

In the case of circuit branches, the number of pairs is determined by the number of their first LUT units. Versions are numbered with a binary code that contains the values of one in bits corresponding to the inverted LUT inputs of the node. In the case of a 4-LUT unit, the version can be indicated by a hexadecimal character. For example, version $0_{16} = 0000_2$ is the original version, and version $0_{16} = 0101_2$ is provided by inverting inputs A and C [46, 47].

An important circumstance is the simplicity of changing the versions of the program code in the LUT units of the FPGA component.

## 3.2. The essence of the method

The method offers the organization of FPGA components operation with a sequential change of program code versions as an alternative to manual regulation of input data for their most complete change in normal mode and as an alternative to the use of simulation mode that increases the checkability of circuits by recreating emergency operating conditions for safety-related systems and their components. The purpose of the version change is to improve the checkability of the circuit by moving bits in the memory of the LUT nodes.

The main idea of the method is to develop the ability to move the memory LUT bits, observed only in emergency mode, to positions that can be observed during normal mode.

According to the ability of a pair of LUT units to create two versions of the program code, moving bits is possible only for the memory of the second LUT units of the pair. This limitation excludes a significant portion of the memory LUT bits from the set of relocatable circuit bits. First of all, the opportunity to improve the checkability of the circuit is lost in its first-level LUT units, since they cannot be the second units of the pair.

In pyramidal schemes, the first level is the most numerous. In the absence of branches after the first and subsequent levels, the use of all inputs of the $m$-LUT units to connect to the units of the previous level reduces the number of each next level by a factor of $m$. In this case, the first level contains $T_1 = m\,k - 1$ $m$-LUT units, where $k$ is the number of levels in the circuit. The number of $m$-LUT units in all subsequent levels of the circuit is determined as $T_{2\,\ldots\,k} = (m^{k-1} - 1) / (m - 1)$, $m > 1$. Difference $\Delta T = T_1 - T_{2\,\ldots\,k}$ and ratio $\delta T = T_1 / T_{2\,\ldots\,k}$ are defined as $\Delta T = (m^{k-1}\,(m - 2) + 1) / (m - 1)$ and $\delta T = m - 1 + (m^{k-1} - 1)^{-1}$, respectively.

For example, for $m = 4$ and $k = 5$, the number of 4-LUT units in levels increases from outputs to inputs of the circuit as follows: 1, 4, 16, 64 and 256. The number of 4-LUT units in the first level and other levels is $T_1 = 256$ and $T_{2\,\ldots\,k} = 85$, i.e., the first level outnumbers the other levels by more than 3 times ($\delta T = 3.01$).

Connecting individual level inputs bypassing any level reduces the $m$ value for the corresponding LUT units. However, the inequality $T_1 > T_{2\,\ldots\,k}$ is preserved even for $m = 2$, since in this case $\Delta T = 1$.

The pyramidal scheme was considered without branching. However, it should be noted that branching the outputs of the LUT units does not increase the number of pairs, since they are counted by their first LUT units.

In addition, it is necessary to take into account the reduction in the number of versions when connecting the second LUT units of the pair to the inputs of the circuit. Each such connection makes it impossible to invert the corresponding input of the LUT unit and accordingly halves the number of versions that can be created for it.

Thus, many versions of the program code are limited by the pyramidal structure of digital circuits, the presence of branches and connections to the inputs of the circuit.

It is also important to take into account the peculiarities of such matrix circuits as iterative array multipliers and dividers, which connect $n$-bit operands to the $n^2$ and $(n+1)^2$ inputs of the operating elements. This significantly reduces the number of versions in the case of operands coming from the inputs of a digital circuit.

The proposed method is aimed at significantly expanding a set of versions by introducing a $Z$-bit shift register and $Z$ adders modulo two, where $Z$ is the number of information inputs of the digital circuit implemented in the FPGA project. A one-to-one correspondence is established between the bits of the shift register, the adders modulo two and the information inputs of the digital circuit. In the new FPGA project, its inputs are connected to the inputs of the original circuit by means of the corresponding adders modulo two, i.e., each input of the FPGA project is connected to the first input of the corresponding adder modulo two, the second input of which is connected to the output of the corresponding register bit, and the output serves as the input to the original circuit. Information input of register and its sync input are additional inputs of the FPGA project, with the help of which the register code can be set or changed.

Thus, all LUT units following those implementing modulo two adders become the second LUT units of the pair. They do not have connections to the FPGA inputs of the project and therefore ensure the creation of the maximum possible number of versions for them. For this, modulo two adders must

provide direct and inverse values at the inputs of the next LUT units. These values are formed by writing the corresponding sequential code to a register.

The additional circuit increases the computation time by the delay of one LUT unit that implements the modulo two adder, and therefore does not significantly affect the performance of the FPGA component"

## 3.3. Method steps

The proposed method is carried out using the results of FPGA designing of the digital component under study and a program for simulating calculations performed in the LUT-oriented architecture of the project, by the sequence of the following steps.

Step 1. The original FPGA project is finalized by introducing an additional circuit consisting of a shift register and modulo two adders.

Step 2. A description of the resulting LUT-oriented architecture is compiled. Files of description of inputs, modified FPGA project, connection diagrams of LUT units, their program codes, circuit outputs are formed.

Step 3. The computation simulation program is supplemented with the compiled description files. General data on the simulated circuit and its operating modes are established, including the number of LUT units, circuit inputs and outputs, as well as the ranges of input data change in normal and emergency modes.

Step 4. The program ranks the circuit by renumbering the LUT units in the course of processing the input data to perform the logical functions described in the memory of the LUT units in the order of the assigned numbers.

Step 5. The program iterates over all the input data of the circuit, marking their belonging to the normal or emergency modes.

Step 6. For each input word, the program simulates the execution of computations and forms an array of results.

Step 7. For the same input data, the simulation continues by examining the LUT units one by one for observability of the addressable memory bits. The output of the LUT unit under investigation is inverted to trace the inverted value in its propagation to the outputs of the circuit. The results obtained are compared with the results calculated in step 6 to assess the observability of the memory bits addressed in normal and emergency modes. Arrays N and E are formed, which determine the sets of $M_N$ and $M_E$ memory bits of the LUT units observed on the normal mode input data and on the emergency mode only input data, respectively.

Step 8. Arrays ON and OE of intersection and difference of sets $M_E$ and $M_N$ are formed. These arrays identify the bits that are checkable in normal mode and the potentially hazardous bits, respectively. The distortion of potentially hazardous bits is hidden in normal mode and can manifest itself as a result error with the onset of emergency mode.

Step 9. Versions of the program code are determined that allow to swap the bits of all second LUT units of the pair, ensuring that each potentially hazardous bit is moved to the position of the checkable bit when the register code is zero. In this case, the register and modulo two adders of the complementary circuit do not affect the computations that are performed according to the original FPGA project. The version number is determined by the modulo-two sum of the binary codes for the numbers of two mutually relocatable bits. For example, moving the potentially hazardous bit $3_{16} = 0011_2$ to the checkable bit position $D_{16} = 1101_2$ is done using version $0011_2 \oplus 1101_2 = 1110_2 = E_{16}$.

Step 10. An array NE is formed that identifies a plurality of $M_{NE}$ bits that are not versioned to move to checkable positions with the register code zero unchanged. Code versions are defined that allow the bits of multiple $M_{NE}$ bits to be moved when the register code is changed. The received versions are assigned to the corresponding register code.

Step 11. An array EE is formed, marking the set of $M_{EE}$ bits that remain potentially hazardous. They cannot be moved to checkable positions under any version and register codes, since they belong to the unit's LUT memory, which does not contain the checkable bits.

Thus, the checkability of the scheme is achieved by its operation on a sequence of replaceable versions, which are provided taking into account the possibility of changing the register code. In this

case, the method allows the possibility of storing a certain number of potentially hazardous bits in the memory of the LUT units. The $M_{EE}$ sat of such bits is identified in the EE array, and information about a non-empty $M_{EE}$ set is reported.

## 3.4. Experimental approbation of the method

The method has been tested on a number of FPGA projects, including iterative array multipliers, which showed the most typical results. Iterative array multiplier circuits taken from the standard library LPM_mult CAD Quartus [48] were implemented in the Intel Cyclone 10 LP FPGA chip: 10CL025YU256I7G [49] using CAD Quartus Prime 20.1 Lite Edition [50]. Iterative array multipliers were investigated for size $n$ = 4, 6 and 8 in FPGA projects before and after the introduction of an additional circuit composed of modulo two adders and registers. Their introduction increased the number of LUT units from 30, 61 and 101 to 38, 70 and 116, i.e., by 26.7%, 14.8%, 14.9%, respectively. The trend towards a decrease in the percentage of additionally introduced LUT units is explained by the quadratic and linear dependence of the number of LUT units used, respectively, in iterative array multipliers and additional circuits.

FPGA projects were simulated using a program developed in the Delphi environment on a free demo version [51].

For each FPGA project, the program conducts 8 experiments, differing by the threshold $S$, which separates the ranges of variation of the factors in normal and emergency modes.

For a 4-bit multiplier, the threshold varies from 2 to 9 with a step of 1, taking into account the change in the factors from 1 to 15. In the case of the threshold $S = 2$, the factors take only two values 0 and 1 in normal mode and the remaining values from 2 to 15 – in emergency mode. With an increase in the size $n$, the initial value of the threshold $S$ retains the value 2, and the step of changing the threshold increases in accordance with the formula that defines it $\Delta = 2n - 4$, i.e., it grows to values 4, 16, and 64 for $n$ = 6, 8 and 10, respectively.

The input data of the factors is selected from a $2n$-bit code that takes all values from 0 to $2^{2n} - 1$.

The program allows to view the memory of all LUT units for all values of the threshold $S$ with a distinction between the bits observed during normal mode and observed only in emergency mode. In addition, the memory of the LUT units shows non-relocatable potentially hazardous bits when present.

All this information is shown on the main panel of the program, which is presented in Fig. 1 for FPGA project of 4-bit multiplier with additional circuit introduced.

The main panel contains controls for exiting the program and starting the simulation: "EXIT" and "Start", as well as the "LUT # 20" key, which determines the number of the LUT unit considered after the completion of the simulation. Pressing this key allows to go to the next LUT unit. Scan of LUT units starts from number 1 and, if necessary, is repeated after reaching the highest number.

The LUT unit under consideration is represented by its memory in the form of bit matrices shown for eight threshold $S$ values from 2 to 9. The matrix contains 16 squares in which the bit values are shown. The bit numbers are determined by their position at the intersection of the rows "DC" and columns "BA" of the matrix, which are numbered $00_2$, $01_2$, $10_2$ and $11_2$. These codes constitute the $dcba_2$ address, which is also the bit number. For example, the bit at the intersection of row $10_2$ and column $01_2$ is $1001_2 = 9_{16}$.

The image of the memory matrices makes it possible to distinguish between the bits observed in both modes and only in one of them: normal or emergency. These bits are colored blue, green, and yellow, respectively. The values of the bits that are observed only in the emergency mode are highlighted in red or blue, respectively, for cases when these bits are movable and non-movable when the register code is zero.

The memory matrices show the change in bits in their observability, which occurs with increasing threshold $S$ for the LUT unit 20. For $S = 2$, only one bit $0_{16}$ is observed in normal mode. The rest of the bits are observed only in emergency mode, including bits $4_{16}$, $6_{16}$, $C_{16}$ and $E_{16}$, which refer to non-relocatable bits when the register code is zero and are stored as such for all $S < 9$.

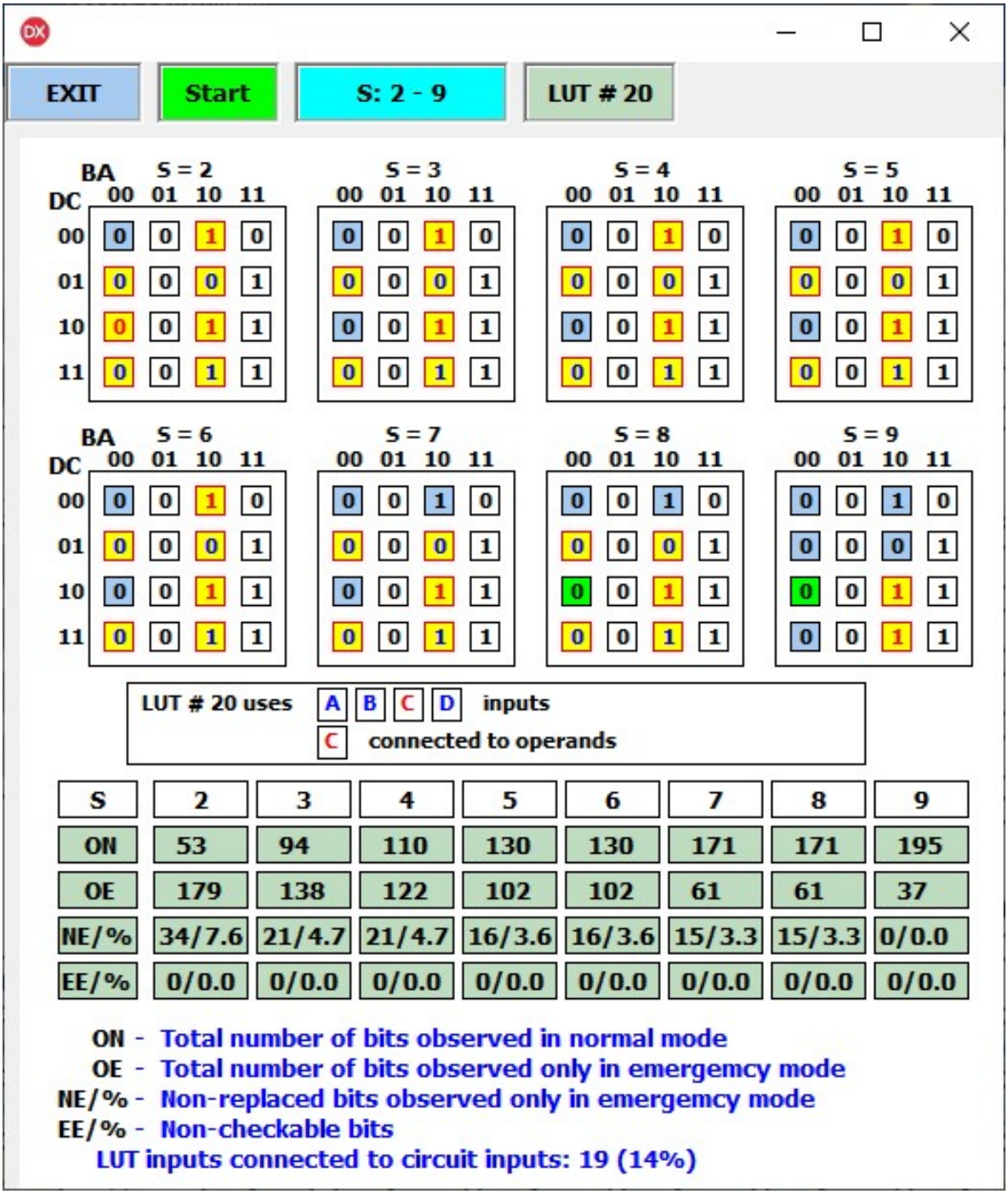


**Figure 1**: Main panel of the program

As the threshold $S$ increases from 3 to 6, bit $8_{16}$ becomes observed in normal mode. For $S = 7$, bit $8_{16}$ is also appended to the bits observed in normal mode. Reaching the threshold $S = 8$ excludes bit $8_{16}$ from being monitored in emergency mode. In the case of $S = 9$, the number of bits observed only in emergency mode is reduced to two bits: $A_{16}$ and E*16*, which are relocatable when the register code is zero.

Below the memory matrices, the main panel shows the used inputs of the LUT unit in question. Its inputs, which are the inputs of the project's FPGA circuit, are highlighted in red. The next line shows the circuit inputs connected to the operands.

The table located on the main panel shows the change in the number of bits in the entire FPGA project, observed during normal mode (ON line) and only in emergency mode (OE line) when the threshold $S$ increases from 2 to 9. The number of bits in the ON line increases from 53 to 195, and the

number of bits in the OE line decreases from 179 to 37. The next line shows the number and percentage of the bits of the set $M_{NE}$, which have no version to move to checkable positions when the zero code of register remains unchanged. The last row of the table shows the number and percentage of bits in the $M_{EE}$ set that remain potentially hazardous because they cannot be moved to checkable positions due to their absence.

The last line of the main panel shows the number and percentage of the inputs of the LUT units connected to the inputs of the FPGA of the project.

A set of $M_{EE}$ bits that could be potentially hazardous turned out to be empty in all FPGA projects studied for all values of the threshold *S*.

The main results of experiments carried out for the original and advanced iterative array multipliers are shown in Table. 1, where NE indicates the number and percentage of bits of a plurality of $M_{NE}$ observed only in emergency mode. The indices denote the size of the iterative array multipliers, and the "*" symbol refers to the results to advanced multipliers, whose FPGA projects contain additional scheme.

**Table 1**
Experiment Results

| *S* | 2 | 2+Δ | 2+2Δ | 2+3Δ | 2+4Δ | 2+5Δ | 2+6Δ | 2+7Δ |
|---|---|---|---|---|---|---|---|---|
| $NE_4$ / % | 188 / 47.4 | 147 / 37.1 | 130 / 32.8 | 100 / 25.2 | 157 / 21.0 | 80 / 20.2 | 80 / 20.2 | 47 / 11.8 |
| $NE_{4*}$ / % | 34 / 7.4 | 21 / 4.7 | 21 / 4.7 | 16 / 3.6 | 16 / 3.6 | 15 / 3.3 | 15 / 3.3 | 0 / 0 |
| $NE_6$ / % | 272 / 36.3 | 227 / 30.3 | 205 / 27.4 | 175 / 23.4 | 175 / 23.4 | 157 / 21.0 | 157 / 21.0 | 128 / 17.1 |
| $NE_{6*}$ / % | 52 / 5.7 | 40 / 4.4 | 40 / 4.4 | 36 / 3.9 | 36 / 3.9 | 35 / 3.8 | 35 / 3.8 | 25 / 2.7 |
| $NE_8$ / % | 480 / 36.7 | 425 / 32.5 | 397 / 30.4 | 359 / 27.5 | 359 / 27.5 | 337 / 25.8 | 337 / 25.8 | 300 / 22.9 |
| $NE_{8*}$ / % | 69 / 4.3 | 56 / 3.6 | 56 / 3.6 | 48 / 3.1 | 48 / 3.1 | 46 / 3.0 | 46 / 3.0 | 37 / 2.4 |

In addition, the results of the experiments carried out show a significant effect on them by the number of inputs of the LUT units connected to the inputs of the FPGA of the project (Table 2).

**Table 2**
Experiment Results

| Multipliers | 4 | 4* | 6 | 6* | 8 | 8* |
|---|---|---|---|---|---|---|
| Inputs | 57 | 19 | 85 | 37 | 132 | 45 |
| % | 52 | 14 | 41 | 13 | 37 | 10 |

For all projects, the number of LUT inputs connected to the inputs of the FPGA circuit monotonically increases in absolute terms and decreases in percentage with an increase in the size of the iterative array multipliers. However, the original multipliers increase the number of such connections from 57 to 132, and advanced projects from 19 to 45, i.e., almost 3 times less.

## 4. Discussion of experimental results

The results of the experiments showed the capabilities of the proposed method to significantly increase the checkability of the FPGA project.

For the original multipliers, the bits of the $M_{NE}$ set are potentially hazardous, since they are observed only in emergency mode and have no versions to move to checkable positions. With an increase in the threshold *S*, their number decreases from 188, 272 and 480 to 47, 128 and 300 for the size $n = 4$, 5 and 8, respectively. In percentage terms, potentially hazardous points range from 47.4%, 36.3% and 36.7% to 11.8%, 17.1% and 22.9%. These estimates indicate low checkability of library matrix circuits and a real threat to functional safety from potentially hazardous bits of FPGA designs.

For advanced FPGA projects, the bits of the MNE set are also observed only in emergency mode, but they are not potentially hazardous, since they receive versions for moving to checkable positions at various non-zero values of the register code. With an increase in the threshold *S*, the number of

such bits decreases from 34, 52 and 69 to 0, 25 and 37 for size $n$ = 4, 5 and 8, respectively. The percentage of these bits is reduced from 7.4%, 5.7% and 4.3% to 0%, 2.7% and 2.4%. Since the experiments carried out did not reveal the bits of the $M_{EE}$ set, then all the advanced FPGA projects investigated do not contain potentially hazardous points and, therefore, are totally checkable.

The significant reduction in the number of bits in the $M_{NE}$ set of advanced FPGA projects can be explained by a threefold decrease in the number of LUT inputs connected to the inputs of the FPGA circuit. This effect, revealed experimentally, is also positive from the standpoint of reducing restrictions on the creation of versions of program code.

## 5. Conclusions

Safety-related systems base their own functional safety and the safety of control objects on the use of fault-tolerant solutions, for which the main threat comes from sources of multiple failures.

Hidden faults, which can be accumulated during a long normal operation and manifest as multiple failures with the onset of an emergency mode, form one of such sources that require improving the checkability of systems and their components.

For safety-related systems, FPGA checkability is essential to transform fault-tolerant solutions into fail-safe ones.

The LUT-oriented architecture of FPGA projects allows the accumulation of hidden faults that disrupt the reading of data from the memory of the LUT units.

The proposed method makes it possible to significantly increase the checkability of FPGA components by changing the versions of the program code and additional possibilities to invert the inputs of the LUT units.

Experiments have shown the effectiveness of the proposed method, which provided the transformation of FPGA projects with low checkability into totally checkable solutions.

## 6. References


[1] S. F. Tyurin, Hyper redundancy for super reliable FPGAs, Radioelectronic and computer systems 1 (2021) 119–132. http://nti.khai.edu/ojs/index.php/reks/article/view/reks.2021.1.11/1449

[2] O. Drozd, V. Romankevich, M. Kuznietsov et. al., Using natural version redundancy of FPGA projects in area of critical applications, in: Proceedings of the International Conference DESSERT, 2020, pp. 58–63. doi: 10.1109/DESSERT50317.2020.9125050.

[3] M. Farias, R.H.S. Martins, P.I.N. Teixeira, P.V. Carvalho, FPGA-based I&C systems in nuclear plants, Chemical, Engineering Transactions 53 (2016) 283–288. doi: 10.3303/CET1653048.

[4] J. Bergström, R. van Winsen, E. Henriqson, On the rationale of resilience in the domain of safety: A literature review, Reliability Engineering and System Safety, 141 (2015) 131–141.

[5] J.-C. Le Coze, Outlines of a sensitising model for industrial safety assessment, Safety Science, 51 (2013) 187–201.

[6] T. Aven, P.Baraldi, R. Flage, E. Zio, Uncertainties in risk assessments, Wiley, Chichester (2014).

[7] Y. Ben-Haim, Doing our best: Optimization and the management of risk**,** Risk Analysis, 32 (8) (2012) 1326–1331.

[8] O. Odarushchenko, V. Kharchenko, D. Butenko, V. Butenko, Assessment of the Reactor Trip System Dependability Two Markov Chains - based Cases, in: Proceedings of the 10th International Conference on Digital Technologies, Zilina, Slovakia, 2014, pp. 103–109.

[9] J. C. Jung, Improved design architecture to minimize functional complexity of plant protection system for nuclear power plant, Nucl. Eng. Des., 309 (2016) 97–103.

[10] IAEA Safety Standards, Specific Safety Guide No.SSG-39, Design of Instrumentation and Control Systems for Nuclear Power Plants, 2016.

[11] I. Atamanyuk, Y. Kondratenko, Computer's analysis method and reliability assessment of fault-tolerance operation of information systems, in: CEUR Workshop Proceedings, vol. 1356, 2015, pp. 507–522.

[12] A. Romankevich, A. Feseniuk, V. Romankevich, T. Sapsai, About a fault-tolerant multiprocessor control system in a pre-dangerous state, in: Proceedings of the 9th IEEE International Conference DESSERT, Kyiv, Ukraine, 2018, pp. 215–219.
[13] C. Wang, L. Xing and G. Levitin, Explicit and Implicit Methods for Probabilistic Common-Cause Failure Analysis, Reliability Engineering and System Safety, vol. 131, 2014, pp. 175–184.
[14] V. Kharchenko, E. S Bakhmach, A. A. Siora at. al., Diversity-Oriented FPGA-Based NPP I&C Systems: Safety Assessment, Development, Implementation, in: Proceedings of the 18th International Conference on Nuclear Engineering, 2010, pp. 755–764.
[15] V. Kharchenko, Dependable Systems and Multi-version Calculations: Aspects of Evolution, Radioelectronic and Computer Systems, 7 (41) 2009 46–59.
[16] A. A. Gashi, L. Povyakalo, M. Strigini et. al., Diversity for safety and security in embedded systems, in: Faste Abstracts of the IEEE International Conference on Dependable Systems and Networks, Atlanta, GA, USA, 2014.
[17] S. Lysenko, K. Bobrovnikova, O. Savenko, A. Kryshchuk, BotGRABBER: SVM-Based Self-Adaptive System for the Network Resilience Against the Botnets' Cyberattacks, Communications in Computer and Information Science 1039 (2019) 127–143.
[18] S. Lysenko, K. Bobrovnikova, O. Savenko, R. Shchuka. A Cyberattacks Detection Technique Based on Evolutionary Algorithms, in: Proceedings of the 11th IEEE International Conference DESSERT, Kyiv, Ukraine, 2020, pp. 127–132.
[19] A. Drozd, V. Kharchenko, S. Antoshchuk et. al., Checkability of the digital components in safety-critical systems: problems and solutions, in: Proceedings of IEEE East-West Design & Test Symposium, Sevastopol, Ukraine, 2011, pp. 411–416. doi: 10.1109/EWDTS.2011.6116606.
[20] A. Drozd, M. Drozd, V. Antonyuk, "Features of Hidden Fault Detection in Pipeline Components of Safety-Related System," CEUR Workshop Proceedings, vol. 1356, 2015, pp. 476–485.
[21] G. S. Spandana, K Hari Kishore, A contemporary approach for fault diagnosis in testable reversible circuits by employing the CNT gate library, International Journal of Pure and Applied Mathematics, 115 (7) (2017) 537–542.
[22] T. Shah, A. Matrosova, V. Singh, PDF testability of a combinational circuit derived by covering ROBDD nodes using Invert-And-Or circuits, in: Proceedings of the International Symposium on VLSI Design and Test, 2015, pp. 1–2.
[23] A.V. Drozd, M.V. Lobachev, Efficient On-line Testing Method for Floating-Point Adder, in: Proceedings of the Design, Automation and Test in Europe Conference, Munich, Germany, 2001, pp. 307–311. doi: 10.1109/DATE.2001.915042.
[24] A. Drozd, J. Drozd, S. Antoshchuk et. al., Objects and Methods of On-Line Testing: Main Requirements and Perspectives of Development, in: Proceedings of the IEEE East-West Design & Test Symposium, Yerevan, Armenia, 2016, pp. 72–76. doi:10.1109/EWDTS.2016.7807750.
[25] TR 026764, ReSIST: Resilience for Survivability, in: IST. Deliverable D12. Resilience-Building Technologies: State of Knowledge, 2006.
[26] C. Cachin, S. Tessaro, Optimal Resilience for Erasure Coded Byzantine Distributed Storage, in: Proceedings of International Conference on Dependable Systems and Networks, 2006.
[27] J. Drozd, A. Drozd, M. Al-dhabi, A resource approach to on-line testing of computing circuits, in: Proceedings of the IEEE East-West Design & Test Symposium, Batumi, Georgia, 2015, pp. 276–281. doi: 10.1109/EWDTS.2015.7493122.
[28] O. Drozd, K. Zashcholkin, R. Shaporin, J. Drozd, Y. Sulima, Development of ICT Models in Area of Safety Education, in: Proceedings of the IEEE EWDT Symposium, Varna, Bulgaria, 2020, pp. 212–217. doi: 10.1109/EWDTS50664.2020.9224861.
[29] T. Hovorushchenko, O. Pavlova. Evaluating the Software Requirements Specifications Using Ontology-Based Intelligent Agent, in: Proceedings of IEEE International Scientific and Technical Conference "Computer Science and Information Technologies", Lviv, Ukraine, 2018, pp. 215–218.
[30] T. Hovorushchenko, O. Pomorova, Evaluation of Mutual Influences of Software Quality Characteristics Based ISO 25010:2011, in: Proceedings of the IEEE International Conference, CSIT-2016, Lviv, Ukraine, 2016, pp. 80–83. doi: 10.1109/STC-CSIT.2016.7589874.
[31] V. Kharchenko, A. Gorbenko, V. Sklyar, C. Phillips, Green Computing and Communications in Critical Application Domains: Challenges and Solutions, in: Proceedings of 2016 IX International Conference of Digital Technologies, Zhilina, Slovak Republic, 2013.

[32] J. Drozd, A. Drozd, S. Antoshchuk, Green IT engineering in the view of resource-based approach," In book: Green IT Engineering: Concepts, Models, Complex Systems Architectures, vol. 74. Berlin, Heidelberg: Springer International Publishing, 2017, pp. 43–65. doi: 10.1007/978-3-319-44162-7_3.
[33] V. Hahanov, E. Litvinova, S. Chumachenko, Green Cyber-Physical Computing as Sustainable Development Model, in: Green IT Engineering: Components, Networks and Systems Implementation, vol. 105, Springer, Berlin, 2017, pp. 65–86. doi: 10.1007/978-3-319-55595-9_4.
[34] Palagin, A., Opanasenko, V.: The implementation of extended arithmetic's on FPGA-based structures, in: IEEE International IDAACS Conference, vol. 2, Bucharest, Romania, pp. 1014–1019 (2017) doi: 10.1109/IDAACS.2017.8095239
[35] J. Drozd, A. Drozd, S. Antoshchuk, at. al., Effectiveness of Matrix and Pipeline FPGA-Based Arithmetic Components of Safety-Related Systems, in: Proceedings of the IDAACS Conference, Warsaw, Poland, 2015, pp. 785–789. doi: 10.1109/IDAACS.2015.7341410.
[36] A. Drozd, J. Drozd, S. Antoshchuk et. al., Green Experiments with FPGA. In book: Green IT Engineering: Components, Networks and Systems Implementation, vol. 105. Berlin, Heidelberg: Springer International Publishing, 2017, pp. 219–239. doi: 10.1007/978-3-319-55595-9_11.
[37] S. Warren, J. Anderson, FPGA Glitch Power Analysis and Reduction, in: Proceedings of the International Symposium on Low power electronics and design, Fukuoka, Japan, 2011, pp. 27–32.
[38] J. Drozd, A. Drozd, S. Antoshchuk, V. Kharchenko, Natural Development of the Resources in Design and Testing of the Computer Systems and their Components, in: Proceedings of the IDAACS Conference, Berlin, Germany, 2013, pp. 233–237. doi: 10.1109/IDAACS.2013.6662656.
[39] A. Drozd, M. Lobachev, W. Hassonah, Hardware check of arithmetic devices with abridged execution of operations, in: Proceedings of the European Design & Test Conference, Paris, France, 1996, p. 611. doi: 10.1109/EDTC.1996.494375.
[40] V. Kulanov, V. Kharchenko, A. Perepelitsyn, Parameterized IP Infrastructures for fault-tolerant FPGA-based systems: Development, assessment, case-study, in: IEEE East-West Design and Test Symposium EWDTS'10, St. Petersburg, Russia, 2010, pp. 322–325.
[41] C. Unsalan, B. Tar, Digital System Design with FPGA. New-York, McGraw-Hill, 2017.
[42] Intel FPGA Architecture, 2019. URL: https://www.intel.com/content/dam/www/programmable /us/en/pdfs/literature/wp/wp-01003.pdf
[43] S. F. Tyurin, LUT's Sliding Backup, IEEE transactions on device and materials reliability, 2019 19(1), pp. 221-225. doi: 10.1109/TDMR.2019.2898724.
[44] O. Drozd, M. Kuznietsov, O. Martynyuk, M. Drozd. A method of the hidden faults elimination in FPGA projects for the critical applications, in: Proceedings of the IEEE International Conference DESSERT'2018), Kyiv, Ukraine, pp. 231–234, 2018. doi: 10.1109/DESSERT.2018.8409131.
[45] Amagasaki Motoki, Yuichiro Shibata, "FPGA Structure," in book: Principles and Structures of FPGAs, H. Amano (edits), Springer, USA, New-York, 2018. pp. 47–86.
[46] O. Drozd, K. Zashcholkin, O. Martynyuk, O. Ivanova, J. Drozd. Development of Checkability in FPGA Components of Safety-Related Systems. CEUR Workshop Proceedings, vol. 2762, pp. 30–42 (2020). URL: http://ceur-ws.org/Vol-2762/paper1.pdf.
[47] O. Drozd, K. Zashcholkin, O. Martynyuk, O. Ivanova, J. Drozd. Development of Checkability in FPGA Components of Safety-Related Systems, CEUR Workshop Proceedings 2762 (2020) 30-42. URL: http://ceur-ws.org/Vol-2762/paper1.pdf.
[48] Intel FPGA Integer Arithmetic IP Cores User Guide. https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/ug/ug_lpm_alt_mfug.pdf.
[49] Intel Cyclone 10 LP Core Fabric and General Purpose I/Os Handbook, 2020. URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyclone-10/c10lp-51003.pdf.
[50] Intel Quartus Prime Standard Edition User Guide, 2020. URL: https://www.intel.com/content/dam/altera-www/global/en_US/pdfs/literature/ug/ug-qps-getting-started.pdf.
[51] Delphi 10 Seattle: Embarcadero (2015) https://www.embarcadero. com/ru/products/delphi.