Development of Checkability in FPGA Components of Safety-Related Systems

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

[a]*Odessa National Polytechnic University, Ave. Shevchenko 1, Odesa, 65044, Ukraine*

Abstract

The paper is dedicated to the development of FPGA-designing (Field Programmable Gate Array) components for safety-related systems as an important direction in improving the functional safety of high-risk facilities and the control systems themselves in order to counter accidents and their consequences. The critical application of the computer system diversifies its operating mode into normal and emergency, as well as increases the requirements for fault tolerance of circuits as a basis for functional safety. Fault-tolerant solutions do not become fail-safe in conditions of insufficient checkability, which is inherent in modern safety-related systems and manifests itself in the problem of hidden faults. They can accumulate during normal mode and eliminate fault tolerance in emergency mode. FPGA projects with LUT-oriented (Look-Up Table) architecture inherit this problem in the LUT memory, which is used only in emergency mode. The proposed method develops the FPGA components' checkability by using the version redundancy of their program code. Periodic change of the program code version in normal mode allows to address the memory, which was previously used only with the transition to emergency mode. All versions support the component's FPGA functionality while maintaining its hardware implementation. The method evaluates the controllability and observability of the LUT memory and determines versions that increase its checkability.

Keywords

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

1. Introduction

Information technology, implemented in computer systems, has already outgrown the most daring predictions of its usefulness and, in the form of safety-related systems, has occupied the niche of humanity's defender from risky activities. Indeed, we have created an extensive infrastructure of high-risk objects in energy and transport by placing them on the ground, water, air and space. These facilities are represented by power plants and power networks, ground and air high-speed transport systems, chemical industries and warehouses for storing their dangerous products, as well as various types of weapons [1, 2].

ICT&ES-2020: Information-Communication Technologies & Embedded Systems, November 12, 2020, Mykolaiv, Ukraine
drozd@ukr.net (O. Drozd); const-z@te.net.ua (K. Zashcholkin); anmartynyuk@ukr.net (O. Martynyuk); en.ivanova.ua@gmail.com (O. Ivanova); yuliia.drozd@opu.ua (J. Drozd)
0000-0003-2191-6758 (O. Drozd); 0000-0003-0427-9005 (K. Zashcholkin); 0000-0002-3043-5924 (O. Martynyuk); 0000-0002-4743-6931 (O. Ivanova); 0000-0001-5880-7526 (J. Drozd)

We are all in the zone of defeat in the event of technological disasters, but we do not plan to abandon the development of this infrastructure.

Risk is considered as the product of two factors, the first of which is determined by the probability of an accident, and the second factor is estimated by the cost of consequences from the expected accident [3, 4].

How do these factors develop? The second factor is growing along with the quantitative and qualitative growth of high-risk facilities. We see not only an increase in the number of these objects, but also their density of placement, as well as a progressive proximity to densely populated regions. No less dangerous is the qualitative growth of energy facilities in increasing power characteristics. All this increases the possible losses from the accident. Containment of risk growth is based on the balance in the development of factors, when the growth of the second factor is compensated by a decrease in the probability of an accident.

The mission of risk limitation is fully dedicated to safety-related systems, which, according to international standards, are aimed at ensuring the functional safety of both the facility and the system itself to prevent and mitigate accidents [5, 6].

Achieving the functional safety necessary and sufficient for further development in conditions of expansion by high-risk facilities requires a permanent analysis of the associated problems, their origins, advanced state and solutions. We refer to such problems the limited checkability of circuits in digital components of safety-related systems. This problem receives new content in the perspective direction of FPGA-designing (Field Programmable Gate Array) of digital components.

We propose to consider the analysis of this problem and the associated problem of hidden faults, as well as the development of a method to increase the checkability of FPGA components for critical systems. Section 2 provides an overview of the status of the problem and its features in FPGA-designing. Section 3 describes the method of developing the FPGA components' checkability when solving the problem of hidden faults. The results of the program implementation of the proposed method are shown by an example of an iterative array multiplier in section 4.

2. State of Art in Checkability Development

The main functional safety challenges are failures, and therefore the basis for functional safety is the use of fault-tolerant solutions, including various types of redundancy, reconfiguration and majority structures [7, 8].

The greatest risk comes from common cause failures and hidden faults. Common cause failures received a wide response in international standards which indicate copying decisions as a reason [9, 10]. Measures to counteract common cause failures include the introduction of restrictions on copying based on the development of multi-version technologies and various types of diversity [11, 12].

It should be noted that the danger of common cause failures lies in their hidden nature, when copying an erroneous solution in the redundant channels limits the checkability and allows to choose this solution as correct by a majority vote, which is fully trusted.

However, hidden faults that occur in addition to copying have their own history and pose no less danger to the functional safety of safety-related systems. Hidden faults are not reflected in international standards, although this type of fault is directly associated with safety-related systems, as well as their designing for operation in two modes: normal and emergency.

The problem of hidden faults is the possibility of their accumulating in digital circuits over a long normal mode due to the absence of input data in this mode that manifest these faults. Emergency mode updates input data, which show accumulated faults in reduction or elimination of fault tolerance introduced into circuit solutions during designing [13, 14].

This problem is known from unsuccessful attempts to solve it using imitation modes that recreate alarm conditions to detect hidden faults. These modes showed a dangerous character on two sides: 1) the creation of emergency conditions as a result of unauthorized access due to fault or human factor; 2) a planned imitation mode with emergency protection shutdown leaded to Chernobyl disaster [15, 16].

Dangerous imitation modes create emergency conditions for increasing the circuit checkability, which turns out to be insufficient in relation to hidden faults. Checkability is best known as testability, which is its simplest form, fully determined by the structure of the circuit. The checkability of the circuits is based on their controllability and observability [17, 18].

In the operating mode, the circuit checkability, including controllability and observability, receives additional dependence on the input data and together with them forms an area of effective use of on-line testing, the logical methods of which can detect only faults that appear in the form of an error [19, 20]. In safety-related systems, different inputs of normal and emergency mode convert the checkability into dual-mode form, also different in these modes. This difference causes a problem of hidden faults.

Safe ways to improve the checkability of circuits follow from its analysis using a resource-based approach [21, 22].

This approach analyses the integration of the computer world created by human into the natural one and identifies three levels of development of models, methods and means that form the resources to solve any problem: replication, diversification and self-sufficiency as the goal of development.

At the replication level, integration occurs by stamping resources in the absence of conflict contact with the outside world, that is, in open resource niches. The basis for such integration is productivity, with which, in the natural world, birth rate exceeds mortality, for example, in bacteria. With the closure of resource niches, stamped clones are doomed to extinction and can only survive by manifesting particularities that raise them to the level of diversification.

At this level, productivity gives way to trustworthiness, that is, adequacy to the natural world. Integration takes place in close contact with the natural world by structuring according to its features. The history of the computer world distinguishes two such features: parallelism and fuzziness.

Structuring for these features is illustrated by an example of the development of hardware support for approximate computing in personal computers from coprocessors Intel287/387 optional delivery to several floating-point pipelines in Pentium and several thousand such pipelines in a graphics processor used for parallel computing in CUDA (Compute Unified Device Architecture) technology [23, 24]. Successful structuring led to an improvement in the basic indicators of personal computers by millions of times: the clock frequency rose from kHz to GHz, and the amount of memory increased from Mbytes to Tbytes.

In the computer world, we can see all levels of resource development, but replication dominates. In hardware, replication is represented by matrix structures composed of identical elements. Arithmetic operations are performed using iterative array multipliers and dividers, parallel shifters and adders [25, 26]. Software modules are also stamped and connected to create new software products, clogging them with redundant elements [27, 28]. This replication is enabled and stimulated by the open resource niches of the growing performance and memory of today's computers. Green technologies that reflect the level of diversification limit replication in hardware and software solutions and close the resource niche of uncontrolled energy consumption [29, 30].

The efficiency of matrix structures corresponds to the characteristic of replication as a lowest level of resource development. A typical pattern of matrix structures is an iterative array multiplier performing a key operation of approximate calculations. Multiplication is such an operation, since it is used to represent numbers in floating-point formats and is inherited by all two-operand operations with mantissas, as well as translates the properties of the product to their results [31, 32]. The iterative array multiplier performs an operation with n-bit binary codes in one clock cycle with the help of a matrix of n2 operational elements, $2n - 2$ of which form a serial connection with providing alternately useful use of all elements. For $n = 32$, each of the thousand operating elements is used in the clock cycle only by 1.6%. In the case of $n = 64$, the number of operational elements reaches four thousand, and their usability is halved [33, 34].

At the same time, operational elements are actively used throughout the clock cycle, participating in parasitic switches, the number of which repeatedly exceeds the number of functional transitions of signals and determines the main part of the dynamic component of energy consumption [35, 36]. The static component is formed under the influence of large sizes of matrices.

However, the low efficiency of matrix structures in elements utilization and power consumption is only the tip of the iceberg. The main problem of matrix circuits is manifested in their use for processing data in parallel codes, which generally exhibit limited diversity. For example, a code of size $n = 64$ takes 264 different values, but can actually be used on a small set of them. Such conditions often occur in the normal mode of safety-related systems, which can operate for a long time at the noise level with a change of numerical data only in lower bits.

The resource-based approach identifies the problem of hidden faults as a challenge of growth, when the safety-related system rises to the level of diversification of the operating mode by dividing it into normal and emergency with the resulting diversification of the checkability of the circuits. At the same time, the system components continue to be stamped at the replication level based on matrix structures [37, 38].

Such a vision implies that the problem should be solved by raising the components to the level of system. The simplest solution consists in reduction of matrix structures. However, in reality, such a solution is difficult to implement, since the dominance of matrix structures has been going on for several decades, which have been spent on creating a powerful infrastructure in their defense. The most convincing argument for matrix solutions is modern CAD, including FPGA-designing, which is used in the development of components for safety-related systems [39, 40].

A characteristic feature of FPGA-designing is the organization of a computational process in LUT-oriented (Look-Up Table) architecture using matrices of the Configurable Logic Blocks or Logic Elements. FPGA-designing is supported by built-in matrix multipliers and prepared carry propagation paths for fast addition of parallel codes, as well as libraries of ready-made solutions based on matrix structures [41, 42].

The problem of hidden faults can be solved within matrix structures by analogy with opposing common cause failures, which, as noted above, also show a hidden nature indicating a deficiency of checkability. The common cause failures issue addressed for safety-related systems is also a growth challenge because it is based on a copy that reflects the replication level. Therefore, improving the checkability of matrix structures can be achieved through the use of multi-version technologies by developing the version redundancy of FPGA components.

3. Main Provisions of the Method

The proposed method is aimed at development of checkability of circuits in LUT-oriented architecture for FPGA components of safety-related systems. In this architecture, calculations are performed by decomposing them into Boolean functions performed by LUT units. The description of the function in the form of program code is recorded in the LUT memory of the unit during its programming as part of the FPGA project. The architecture that uses LUT units to generate functions of four variables is most common. These units contain 16 bits of memory addressed by 4-bit code $dcba_2$ on inputs D, C, B and A [43].

The problem of hidden faults is manifested in LUT-oriented architecture in LUT memory bits, addressed only in emergency mode. These bits are not observed during the normal mode and can therefore accumulate faults, i.e., they are uncheckable in that mode.

In practice, the checkability of the LUT memory is improved by manually adjusting the input data by changing them throughout the normal mode range. This adjustment is used to detect faults in components with a slight change in input data during normal mode. However, this approach does not apply to LUT bits addressed only in emergency mode. They remain uncheckable.

The version redundancy of LUT-oriented architecture consists in the ability to program it with many versions while preserving the hardware solution and its functionality. The version redundancy carrier is a pair of LUT units with the output of the first unit connected to the input of the second unit. The program code of such a pair has two versions, which are formed with a direct and inverse value of the signal propagating between the LUT units of the pair. The transition to the inverse value is achieved by inverting all the memory bits of the first LUT unit. The inversion generated at the input of the second LUT unit is compensated by permutation of bits in its memory [44, 45]. The second LUT unit, all inputs of which are connected to outputs of LUT nodes, forms with each of them two versions of program code and 16 versions in total.

Versions are numbered using hexadecimal characters. The values of 1 in their respective binary codes denote the inverse value of the signal at the LUT inputs. For example, codes 0000_2 and 1010_2 number initial version 0_{16} and version A_{16} with inversion at inputs D and C, respectively. The LUT memory bits are numbered with hexadecimal characters according to the $dcba_2$. Codes 0000_2 and 1111_2 are provided to the LUT inputs to access bits 0_{16} and F_{16}, respectively.

An important feature of the versions is the ability to move bits in the memory of the second LUT unit.

Statement. If all the used inputs of the second LUT unit are connected to the outputs of the preceding LUT units, then there is always a version that provides for the exchange of places between any two memory bits of the second LUT unit.

Indeed, the $dcba_2$ code values 0001_2, 0010_2, 0100_2, and 1000_2 change the memory bit number to 1, 2, 4, and 8 upwards or downwards at a bit value of 0 or 1, respectively. Changing the number consists in storing or inverting its individual bits and can be performed by adding modulo two with a code denoting inversion by a value of 1.

For example, we want to change the bit number $a = 1011_2$ to $a = 1001_2$. The code $c = a \oplus b$ takes the value 0010_2, which determines the version number 2_{16} and changes the bit number a to the number $b = a \oplus c$, as well as the bit number b to the number $a = b \oplus c$. Thus, the modulo-two sum of numbers for bits that are swapped uniquely determines the version number that provides such exchange.

As an alternative to manual regulation, the method suggests changing not the input data, but the versions of the program code. This approach can be useful not only for critical systems, but also for testing FPGA projects of any purpose. In this case, at least one addressable bit for at least one input word must be defined for each LUT unit. The addressable bits can then be swapped with the remaining bits by changing the program code.

Thus, the method provides controllability of all bits in the memory of the second LUT units. The first LUT units of the pair are closer to or directly connected to the inputs of the circuit and therefore do not create difficulties in verifying controllability.

For safety-related systems, the method is performed in four steps.

At the first step, FPGA component operation is simulated on input data of normal and emergency mode. For each second LUT unit, the simulation defines N and E sets of memory bits addressed in normal and only emergency mode, respectively. Bits a and b, $a \in N$, $b \in E$ are controllable and uncontrollable in normal mode.

In the second step, the method evaluates the observability of bits of the set N in normal mode by simulating calculations performed by the FPGA component on the input data of that mode. For the input data addressing bit a, the FPGA operation of the component is simulated at the initial and inverse values of the bit. The simulation determines the results of the calculations for each value of bit a and compares them. The erroneous result refers the bit a to a set of N_O bits observed in normal mode.

In the third step, the method increases the controllability of the LUT memory in normal mode. The method defines a table whose rows and columns are denoted by bit numbers b and a of sets E and N_O, respectively. At the intersection of row b and column a , the table is populated with the version number $c = a \oplus b$, which provides exchange for bits b and a. To control the LUT memory in normal mode, the b bits of each row must be exchanged with bit a of at least one column, that is, select the versions at their intersection. The table can contain duplicate versions, since the modulo-two sum can be the same for different arguments. Therefore, the choice of versions is aimed at finding their minimum number necessary to move all bits of the set E. Such a problem can always be solved if the inputs of the project FPGA circuit can take direct and inverse values, for example, if these inputs are outputs of LUT units of previous circuits. Then all LUT units of the FPGA component can be considered as second LUT units of a pair with a full set of versions of program code. The sequential change of the selected versions ensures the memory controllability of all LUTs in the normal mode and their full checkability, taking into account that bits b change places with bits of a set of N_O observed in the normal mode. In the case of using non-invertible LUT inputs, each such input reduces the number of versions by half, and part of the bits of the set Emay not be provided by the version. Such bits b constitute a multiple of uncontrollable E_{NC} bits that can accumulate faults during normal mode.

In the fourth step, the FPGA component is simulated on input data of the emergency mode to estimate the observability of the LUT memory used. For input data addressing bit b of the E_{NC}set, the FPGA operation of the component is simulated at an initial and inverse value of the bit. The simulation determines the results of the calculations for each value of bit b and compares them. Matching the results for all the input data considered and presence at least one erroneous result refers the b bit to the sets of E_{NO} and E_O bits, which are respectively unobservable and observable in the emergency mode. Such unobservable b bits should be excluded from the E_{NC} set because they have no effect on the emergency mode. The $E_O = E_{NC} \setminus E_{NO}$ set contains uncheckable bits of the FPGA component. The initial FPGA component contains $E_{NI} = E \setminus E_{NO}$uncheckable memory bits. Thus, the proposed method reduces the number of uncheckable bits by $e = e_{NI} - e_N$, where e_{NI} and e_N the capacity of the sets E_{NI} and E_N, respectively. The increase in checkability can be estimated as $C = e/e_{NI}$.

The erroneous result can also be analyzed for the hazard it poses in emergency mode for the FPGA component and system. Hazard bits b form the E_H set of dangerous uncheckable bits. Other uncheckable bits relate to non-dangerous bits.

4. Case Study of the Method

The proposed method was tested on a number of FPGA projects, among which the most characteristic pattern is the project of iterative array multiplier considered for n = 4. The FPGA project was obtained using CAD Quartus Prime 20.1 Lite Edition on the Intel Cyclone 10 LP FPGA chip: 10CL025YU256I7G [46, 47]. The resulting circuit comprises 8 inputs, 8 outputs and 30 LUT units, 28 of which have connections to the inputs of the circuit and 22 are the second LUT units of the pair. They allow us to create versions of code: 2, 3, 11 and 6 LUT units can use 16, 8, 4 and 2 versions, respectively.

The program implementation of the proposed method was obtained using Delphi 10 Seattle demo version.

Simulation is done by sequentially implementing functions of LUT units pre-ordered by ranking the circuit. The method is investigated at various values of the threshold that separates the input data into those used in normal and emergency mode. Input data are generated by multiplicands that are below the threshold in normal mode. Reaching the threshold or exceeding it with at least one multiplicand is identified as going into emergency mode. The simulation is performed for eight threshold values in the set range.

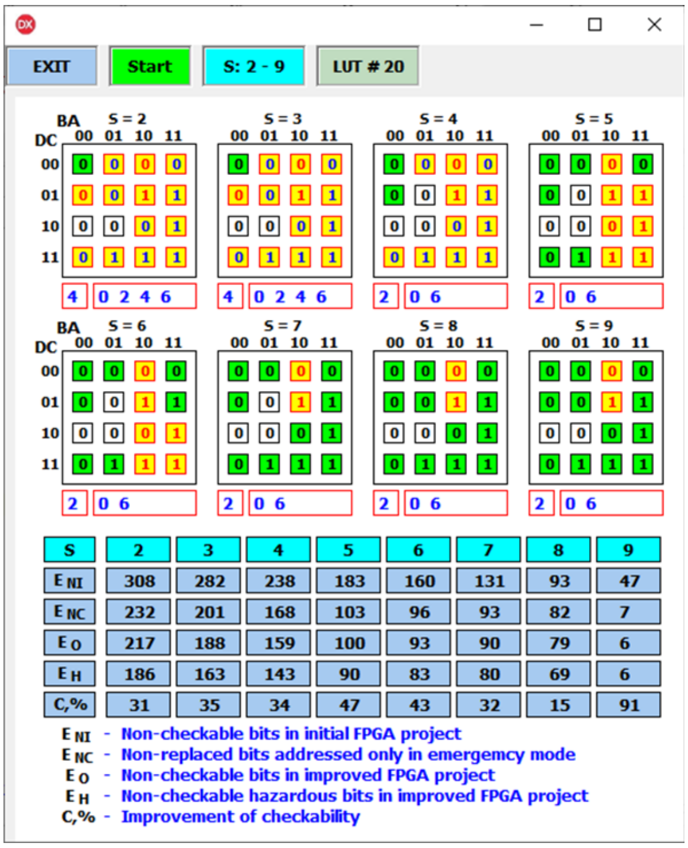
The method is executed step by step. In the first step, the simulation determines the sets N and E of controllable and uncontrollable bits addressed in the memory of each LUT unit throughout the normal mode and only in the emergency mode, respectively.

The second step narrows the N set to the N_O set of bits observed in the normal mode. Thus, the N_O set contains bits that are both controllable and observable throughout the normal mode, and therefore, they relate to checkable in this mode and do not allow the accumulation of hidden faults.

In the third step, the simulation determines the versions for moving the uncontrollable bits of the E set to the position of the checkable bits of the N_O set. Transition to these versions ensures the checkability of the moved bits of the E set. For the case of connecting LUT units to non-invertible inputs of the circuit, the simulation determines the E_{NC} set of uncontrollable bits that cannot be moved to the checkable positions and therefore cannot be prevented from accumulating faults in the normal mode.

The fourth step breaks the E_{NC} set of uncontrollable bits into the sets E_{NO} and E_O of bits, which are respectively unobserved and observed in the emergency mode. Bits of the E_{NO} set do not cause the problem of hidden faults because the faults accumulated during normal mode remain hidden also in emergency mode. In addition, the E_O set of bits are reduced to the E_H set of bits whose distortion is dangerous to the component and the system. In the conducted experiments, the danger is seen in the distortion of the most significant bits, to which the n higher bits of the result have been assigned.

The simulation results represented by the main panel of the method program implementation are shown in Figure 1.



S = 2

DC \ BA	00	01	10	11
00	🟩0	🟨0	🟨0	🟨0
01	🟨0	🟨0	🟨1	🟨1
10	⬜0	⬜0	🟨0	🟨1
11	🟨0	🟨1	🟨1	🟨1

4 | 0 2 4 6

S = 3

DC \ BA	00	01	10	11
00	🟩0	🟨0	🟨0	🟨0
01	🟨0	🟨0	🟨1	🟨1
10	⬜0	⬜0	🟨0	🟨1
11	🟨0	🟨1	🟨1	🟨1

4 | 0 2 4 6

S = 4

DC \ BA	00	01	10	11
00	🟩0	🟨0	🟨0	🟨0
01	🟩0	⬜0	🟨1	🟨1
10	⬜0	⬜0	🟨0	🟨1
11	🟨0	🟨1	🟨1	🟨1

2 | 0 6

S = 5

DC \ BA	00	01	10	11
00	🟩0	🟩0	🟨0	🟩0
01	🟩0	⬜0	🟨1	🟨1
10	⬜0	⬜0	🟨0	🟨1
11	🟩0	🟩1	🟨1	🟨1

2 | 0 6

S = 6

DC \ BA	00	01	10	11
00	🟩0	🟩0	🟨0	🟩0
01	🟩0	⬜0	🟨1	🟩1
10	⬜0	⬜0	🟨0	🟨1
11	🟩0	🟩1	🟨1	🟨1

2 | 0 6

S = 7

DC \ BA	00	01	10	11
00	🟩0	🟩0	🟨0	🟩0
01	🟩0	⬜0	🟨1	🟩1
10	⬜0	⬜0	🟩0	🟩1
11	🟩0	🟩1	🟩1	🟩1

2 | 0 6

S = 8

DC \ BA	00	01	10	11
00	🟩0	🟩0	🟨0	🟩0
01	🟩0	🟩0	🟨1	🟩1
10	⬜0	⬜0	🟩0	🟩1
11	🟩0	🟩1	🟩1	🟩1

2 | 0 6

S = 9

DC \ BA	00	01	10	11
00	🟩0	🟩0	🟨0	🟩0
01	🟩0	🟩0	🟨1	🟩1
10	⬜0	⬜0	🟩0	🟩1
11	🟩0	🟩1	🟩1	🟩1

2 | 0 6

S	2	3	4	5	6	7	8	9
E_{NI}	308	282	238	183	160	131	93	47
E_{NC}	232	201	168	103	96	93	82	7
E_O	217	188	159	100	93	90	79	6
E_H	186	163	143	90	83	80	69	6
C,%	31	35	34	47	43	32	15	91

E_{NI} - Non-checkable bits in initial FPGA project
E_{NC} - Non-replaced bits addressed only in emergemcy mode
E_O - Non-checkable bits in improved FPGA project
E_H - Non-checkable hazardous bits in improved FPGA project
C,% - Improvement of checkability

Figure 1: Main panel of the method program implementation

The main panel shows the set value range for threshold S (2 to 9) and number 20 of the LUT unit. Shift of the range and the number change of the LUT unit is carried out with step 1 in a circle when clicking on the corresponding inscription. For the selected unit 20, the panel shows the matrices of memory bits and their values for each threshold value. The two inputs A and D of this LUT unit are inputs of the FPGA project circuit. Inputs B and C are connected to the outputs of the previous LUT units and allow to create 4 versions of program code, fully used in cases $S = 2$ and $S = 3$ The versions used are shown below the LUT memory and begin with the initial version 0_{16}. The memory bits are colored green and yellow for sets N and E, respectively.

For $S = 2$ and $S = 3$, only one bit 0_{16} belongs to set N and using versions 2_{16}, 4_{16}, and 6_{16} changes places with bits 2_{16}, 4_{16}, and 6_{16} of the E set. The remaining bits 1_{16}, 3_{16}, 5_{16}, 7_{16},$A_{16} - -F_{16}$ of the E set are non-displaceable and also belong to the E_{NC} set. Bits 8_{16} and 9_{16} are not used in both modes.

For the following values of threshold S , the LUT memory increases the number of bits in the N set and reduces their presence in the E set. The bits that can be moved reduce the number of versions used for this to one version 6_{16}.

The overall results of the conducted experiments summarize the analysis of all LUT units for each threshold value S and demonstrate in the table their change with increasing threshold. In Figure 1 it is shown the reduction in the number of bits in the E_{NI}, E_{NC}, E_O, E_H sets the different level of improvement in checkability C of the FPGA project.

The E_{NI} set of uncheckable bits of the initial FPGA project decreases from 308 to 47 due to an increase in the amount of normal mode input data with an increase in the threshold S. The factor of increasing the range of input data in the normal mode contributes to a decrease in the number of bits in the E_{NC}, E_O, E_H sets too.

The E_{NC} set of non-displaceable, uncontrollable bits addressed only in emergency mode are reduced from 232 to 7. The E_O set of uncheckable bits in the improved FPGA project is reduced from 217 to 6 and shows a significant decrease in the number of uncheckable bits compared to the initial FPGA project. On average, the reduction is 64 bits and varies from 14 ($S = 8$) to 94 ($S = 3$) bits. The E_H set reduces the number of uncheckable hazardous bits from 186 to 6.

For the threshold S in the range of $2 - 9$, the proposed method provided an improvement in the checkability of the FPGA project by an average of 41% from C=15% ($S = 8$) to C=91% ($S = 9$). In the range of $2 - 7$, the checkability of the FPGA project is increased by an average of 37%.

5. Conclusions

Safety-related systems face a problem of hidden faults in FPGA designing digital components, which does not allow fault-tolerant solutions to become fail-safe in conditions of insufficient checkability. In FPGA components, this problem is manifested in the LUT memory, where faults can accumulate over a long normal mode in the absence of input data showing these faults in the form of error in results checked by on-line testing methods.

Manual adjustment of input data performed in normal mode does not apply to memory bits addressed in LUT units only in emergency mode. This part of memory is not protected from uncontrolled accumulation of faults in normal mode. Fault-tolerant solutions embedded during the FPGA designing for supporting emergency mode cannot resist the many accumulated faults that first appear in this mode. To solve this problem, it is necessary to improve the checkability of FPGA components.

The resource-based approach reveals the essence of the problem of hidden faults as a growth challenge when components lag behind the development of the system. Components are traditionally developed on the basis of matrix structures, which at the replication level do not cause problems of hidden faults, since computers are used in a single operating mode and the fault remains hidden for the entire time. In the domain of critical applications, systems rise to the level of diversification in the checkability of circuits and require solving emerging problems at this level. The matrix structures inherent in FPGA designing need to be raised to the level of diversification.

The experience gained in counteracting common cause failures shows the way to diversify matrix structures by developing version redundancy. The proposed method enhances the FPGA component checkability by developing the version redundancy in program code of the FPGA project.

The method is carried out in four steps, which allow to estimate the controllability and observability of bits in the LUT memory, to create and use versions of program code for moving uncheckable bits to checkable positions. Sequential transformation of versions allows to completely solve the problem of hidden faults in the case of invertible inputs of the circuit in the FPGA project and significantly increasing its checkability otherwise.

References

[1] D. J. Smith, K. G. Simpson, The Safety Critical Systems Handbook, Elsevier, 2020. URL: https://doi.org/10.1016/c2019-0-00966-1. doi:10.1016/c2019-0-00966-1.

[2] M. A. Yastrebenetsky, V. Kharchenko (Eds.), Nuclear Power Plant Instrumentation and Control Systems for Safety and Security, IGI Global, 2014. URL: https://doi.org/10.4018/978-1-4666-5133-3. doi:10.4018/978-1-4666-5133-3.

[3] S. Choe, F. Leite, Assessing safety risk among different construction trades: Quantitative approach, Journal of Construction Engineering and Management 143 (2017) 04016133. URL: https://doi.org/10.1061/(asce)co.1943-7862.0001237. doi:10.1061/(asce)co.1943-7862.0001237.

[4] O. Ivanchenko, V. Kharchenko, B. Moroz, L. Kabak, S. Konovalenko, Risk assessment of critical energy infrastructure considering physical and cyber assets: Methodology and models, in: 2018 IEEE 4th International Symposium on Wireless Systems within the International Conferences on Intelligent Data Acquisition and Advanced Computing Systems (IDAACS-SWS), IEEE, 2018. URL: https://doi.org/10.1109/idaacs-sws.2018.8525594. doi:10.1109/idaacs-sws.2018.8525594.

[5] Functional Safety of Electrical / Electronic / Programmable Electronic Safety Related Systems – Part 1: General requirements, Technical Report, Geneva: IEC, 2010.

[6] Safety Classification for I&C Systems in Nuclear Power Plants – Current Status & Difficulties, Technical Report 2015/008, World Nuclear Association, 2015.

[7] A. Romankevich, A. Feseniuk, V. Romankevich, T. Sapsai, About a fault-tolerant multiprocessor control system in a pre-dangerous state, in: 2018 IEEE 9th International Con-

ference on Dependable Systems, Services and Technologies (DESSERT), IEEE, 2018. URL: https://doi.org/10.1109/dessert.2018.8409129. doi:10.1109/dessert.2018.8409129.

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

[9] M. Rausand, Risk assessment. common cause failures, 2020. URL: https://www.ntnu.edu/documents/624876/1277591044/ccf.pdf/f435f724-469d-4492-860a-66eca10e6bd2.

[10] Nuclear power plants – Instrumentation and control systems important to safety – Requirements for coping with common cause failure, Technical Report IEC 62340:2007, Geneva: IEC, 2007.

[11] O. Drozd, V. Romankevich, M. Kuznietsov, M. Drozd, O. Martynyuk, Using natural version redundancy of FPGA projects in area of critical applications, in: 2020 IEEE 11th International Conference on Dependable Systems, Services and Technologies (DESSERT), IEEE, 2020. URL: https://doi.org/10.1109/dessert50317.2020.9125050. doi:10.1109/dessert50317.2020.9125050.

[12] H. Asad, I. Gashi, Diversity in open source intrusion detection systems, in: Developments in Language Theory, Springer International Publishing, 2018, pp. 267–281. URL: https://doi.org/10.1007/978-3-319-99130-6_18. doi:10.1007/978-3-319-99130-6_18.

[13] O. Drozd, V. Nikul, V. Antoniuk, M. Drozd, Hidden faults in FPGA-built digital components of safety-related systems, in: 2018 14th International Conference on Advanced Trends in Radioelecrtronics, Telecommunications and Computer Engineering (TCSET), IEEE, 2018. URL: https://doi.org/10.1109/tcset.2018.8336320. doi:10.1109/tcset.2018.8336320.

[14] A. Drozd, S. Antoshchuk, J. Drozd, K. Zashcholkin, M. Drozd, N. Kuznietsov, M. Al-Dhabi, V. Nikul, Checkable FPGA design: Energy consumption, throughput and trustworthiness, in: Green IT Engineering: Social, Business and Industrial Applications, Springer International Publishing, 2018, pp. 73–94. URL: https://doi.org/10.1007/978-3-030-00253-4_4. doi:10.1007/978-3-030-00253-4_4.

[15] D. Gillis, The apocalypses that might have been, 2007. URL: https://www.damninteresting.com/the-apocalypses-that-might-have-been/.

[16] E. Blakemore, The chernobyl disaster: What happened, and the long-term impacts, 2019. URL: https://www.nationalgeographic.com/culture/topics/reference/chernobyl-disaster.

[17] Standard Testability Method for Embedded Core-based IC, Technical Report IEEE Std1500-2005, IEEE, 2005.

[18] V. Hahanov, A. Hahanova, S. Chumachenko, S. Galagan, Diagnosis and repair method of soc memory, WSEAS transactions on circuits and systems 7 (2008) 698–707.

[19] A. Drozd, J. Drozd, S. Antoshchuk, V. Nikul, M. Al-dhabi, Objects and methods of on-line testing: Main requirements and perspectives of development, in: 2016 IEEE East-West Design & Test Symposium (EWDTS), IEEE, 2016. URL: https://doi.org/10.1109/ewdts.2016.7807750. doi:10.1109/ewdts.2016.7807750.

[20] D. Koppad, D. Sokolov, A. Bystrov, A. Yakovlev, Online testing by protocol decomposition, in: 12th IEEE International On-Line Testing Symposium (IOLTS'06), IEEE, 2006. URL: https://doi.org/10.1109/iolts.2006.45. doi:10.1109/iolts.2006.45.

[21] J. Drozd, A. Drozd, M. Al-dhabi, A resource approach to on-line testing of computing circuits, in: 2015 IEEE East-West Design & Test Symposium (EWDTS), IEEE, 2015. URL:

https://doi.org/10.1109/ewdts.2015.7493122. doi:10.1109/ewdts.2015.7493122.

[22] O. Drozd, V. Kharchenko, A. Rucinski, T. Kochanski, R. Garbos, D. Maevsky, Development of models in resilient computing, in: 2019 10th International Conference on Dependable Systems, Services and Technologies (DESSERT), IEEE, 2019. URL: https://doi.org/10.1109/dessert.2019.8770035. doi:10.1109/dessert.2019.8770035.

[23] M. Andrecut, Parallel GPU implementation of iterative PCA algorithms, Journal of Computational Biology 16 (2009) 1593–1599. URL: https://doi.org/10.1089/cmb.2008.0221. doi:10.1089/cmb.2008.0221.

[24] NVIDIA CUDA Compute Unified Device Architecture. Programming Guide, Technical Report Version 1.0, NVIDIA Corporation, 2007.

[25] A. Palagin, V. Opanasenko, The implementation of extended arithmetics on FPGA-based structures, in: 2017 9th IEEE International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), IEEE, 2017. URL: https://doi.org/10.1109/idaacs.2017.8095239. doi:10.1109/idaacs.2017.8095239.

[26] S. Chernov, S. Titov, L. Chernova, V. Gogunskii, L. Chernova, K. Kolesnikova, Algorithm for the simplification of solution to discrete optimization problems, Eastern-European Journal of Enterprise Technologies 3 (2018) 34–43. URL: https://doi.org/10.15587/1729-4061.2018.133405. doi:10.15587/1729-4061.2018.133405.

[27] O. Pomorova, O. Savenko, S. Lysenko, A. Kryshchuk, K. Bobrovnikova, A technique for the botnet detection based on DNS-traffic analysis, in: Computer Networks, Springer International Publishing, 2015, pp. 127–138. URL: https://doi.org/10.1007/978-3-319-19419-6_12. doi:10.1007/978-3-319-19419-6_12.

[28] T. Hovorushchenko, O. Pomorova, Ontological approach to the assessment of information sufficiency for software quality determination (2016).

[29] V. Hahanov, E. Litvinova, S. Chumachenko, Green cyber-physical computing as sustainable development model, in: Green IT Engineering: Components, Networks and Systems Implementation, Springer, 2017, pp. 65–85.

[30] G. Gangadharan, S. Murugesan, Harnessing green it: Principles and practices, 2012.

[31] IEEE Standard for Floating-Point Arithmetic, Technical Report IEEE Std 754™-2008, IEEE, 3 Park Avenue New York, NY 10016-5997, USA, 2008.

[32] Synopsys dwfc flexible floating-point overview, 2016.

[33] J. Drozd, O. Drozd, S. Antoshchuk, A. Kushnerov, V. Nikul, Effectiveness of matrix and pipeline FPGA-based arithmetic components of safety-related systems, in: 2015 IEEE 8th International Conference on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications (IDAACS), IEEE, 2015. URL: https://doi.org/10.1109/idaacs.2015.7341410. doi:10.1109/idaacs.2015.7341410.

[34] B. Neeraja, R. S. P. Goud, Design of an area efficient braun multiplier using high speed parallel prefix adder in cadence, in: 2019 IEEE International Conference on Electrical, Computer and Communication Technologies (ICECCT), IEEE, 2019. URL: https://doi.org/10.1109/icecct.2019.8869307. doi:10.1109/icecct.2019.8869307.

[35] W. Shum, J. H. Anderson, Fpga glitch power analysis and reduction, in: IEEE/ACM International Symposium on Low Power Electronics and Design, IEEE, 2011, pp. 27–32.

[36] V. KumarB.V.P, N. S. M. Sharma, K. L. Kishore, A technique to reduce glitch power during physical design stage for low power and less IR drop, International Journal of

Computer Applications 39 (2012) 62–67. URL: https://doi.org/10.5120/5086-7450. doi:10.5120/5086-7450.

[37] K. Vitaliy, K. Vyacheslav, P. Artem, Parameterized IP infrastructures for fault-tolerant FPGA-based systems: Development, assessment, case-study, in: 2010 East-West Design & Test Symposium (EWDTS), IEEE, 2010. URL: https://doi.org/10.1109/ewdts.2010.5742075. doi:10.1109/ewdts.2010.5742075.

[38] W. Vanderbauwhede, K. Benkrid (Eds.), High-Performance Computing Using FPGAs, Springer New York, 2013. URL: https://doi.org/10.1007/978-1-4614-1791-0. doi:10.1007/978-1-4614-1791-0.

[39] C. Unsalan, B. Tar, Digital system design with FPGA: implementation using Verilog and VHDL, McGraw-Hill Education, 2017.

[40] H. Amano (Ed.), Principles and Structures of FPGAs, Springer Singapore, 2018. URL: https://doi.org/10.1007/978-981-13-0824-6. doi:10.1007/978-981-13-0824-6.

[41] Y. Park, Y. H. Cho, K. Lee, H. Jung, H. Kim, S. Kwon, H. Park, Development of an fpga-based online condition monitoring system for railway catenary application, in: 8th World Congress on Railway Research, COEX, Seoul, Korea, 2008, pp. 18–22.

[42] J. Jung, I. Ahmed, Development of field programmable gate array-based reactor trip functions using systems engineering approach, Nuclear Engineering and Technology 48 (2016) 1047–1057. URL: https://doi.org/10.1016/j.net.2016.02.011. doi:10.1016/j.net.2016.02.011.

[43] Intel fpga architecture, 2019. URL: https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01003.pdf.

[44] O. Drozd, M. Kuznietsov, O. Martynyuk, M. Drozd, A method of the hidden faults elimination in FPGA projets for the critical applications, in: 2018 IEEE 9th International Conference on Dependable Systems, Services and Technologies (DESSERT), IEEE, 2018. URL: https://doi.org/10.1109/dessert.2018.8409131. doi:10.1109/dessert.2018.8409131.

[45] O. Drozd, I. Perebeinos, O. Martynyuk, K. Zashcholkin, O. Ivanova, M. Drozd, Hidden fault analysis of FPGA projects for critical applications, in: 2020 IEEE 15th International Conference on Advanced Trends in Radioelectronics, Telecommunications and Computer Engineering (TCSET), IEEE, 2020. URL: https://doi.org/10.1109/tcset49122.2020.235591. doi:10.1109/tcset49122.2020.235591.

[46] 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.

[47] 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.