A Method to Improve FPGA Project Checkability for Safety-Related Applications

Oleksandr Drozd[0000-0003-2191-6758], Oleksandr Martynyuk[0000-0003-1461-2000], Kostiantyn Zashcholkin[0000-0003-0427-9005], Mykola Kuznietsov[0000-0002-3043-5924], Julia Drozd[0000-0001-5880-7526], Anastasiya Troynina[0000-0001-6862-1266]

Odessa National Polytechnic University, Ave. Shevchenko 1,
65044, Odessa, Ukraine
drozd@ukr.net, anmartynyuk@ukr.net, const-z@te.net.ua,
koliaodessa@ukr.net, yuliia.drozd@opu.ua,
anastasiyatroinina@gmail.com

Abstract. FPGA-designing (Field Programmable Gate Array) with LUT-oriented (Look-Up Table) architecture enjoys well-deserved recognition in the field of safety-related applications, where important tasks are solved to ensure the functional safety of high-risk objects to prevent accidents. These tasks are assigned to safety-related systems, which are the development of ordinary computers with the division of operating mode into normal and emergency ones and increased requirements for functional safety provided using fault-tolerant solutions. Under these conditions, FPGA designing encounters the problem of hidden faults that can accumulate in memory bits of LUT units in normal mode and reduce the fault tolerance of the FPGA project with the beginning of the most responsible emergency mode. This problem is due to the limited checkability of FPGA projects, which is due to memory bits addressed only in emergency mode. The method of improving the checkability of FPGA projects based on the version redundancy of their program codes is proposed. The work of FPGA projects is organized with a periodic change of program code versions for addressing in normal mode to all used memory bits. The method is shown in the example of the FPGA project designed for the iterative array multiplier, where it determines all versions of the program code and selects their minimum number to maximize the checkability of the project.

Keywords: safety-related system, normal and emergency modes, FPGA project, LUT-oriented architecture, problem of hidden faults, checkability, memory bits of LUT unit, program code version.

1 Introduction and Related Works

FPGA-designing (Field Programmable Gate Array) is a promising direction in the development of digital components of computer systems. In turn, the development of computer systems receives the highest priority in the domain of safety-related applications. Therefore, we can observe the priority development of safety-related systems based on FPGA-designing [1, 2].

Safety-related systems play an important role in the sustainable development of mankind, allowing you to increase productive capacity and protect the environment from it, including the person himself. This process is based on the balance of quantitative and qualitative growth of high-risk objects, on the one hand, and the improvement of technologies implemented into Instrumentation and Control safety-related systems, on the other scale. High-risk objects are represented by power plants and power networks, high-speed land, and air transport with a powerful supporting infrastructure. Humanity cannot abandon a such development and increases the total capacity of these facilities, as well as the complexity that limits their observability and controllability. Safety-related systems are aimed at ensuring functional safety in the complex: both own and high-risk objects to prevent accidents and reduce losses if the accident cannot be prevented [3, 4].

The risk can be represented by the product of two factors: the cost of the accident consequences and the probability with which it can occur [5, 6]. The first factor is constantly growing along with the capacity of high-risk objects. Risk deterrence is possible only through a second factor, the reduction of which can be achieved only by improvements in safety-related systems.

Functional safety, for which failures belong to the main challenges, is based on the use of fault-tolerant solutions, including multi-version technologies. They are aimed at counteracting failures for a common cause, which follows from copying circuit and software solutions [7].

It should be noted that the attributes of observability and controllability are important not only for the high-risk object, but also for the functional safety of the control system, which can also be complicated in the process of its improvement. These attributes form the basis of testability and testable design of the digital components for computer systems [8-10].

Testability evaluates the digital circuit from the position of its suitability for testing, i.e. detection of faults in operation pauses. This feature characterizes testability as the simplest form of checkability, which is completely determined by the structure of the circuit, i.e., is structural checkability. In the operating mode, the checkability of the circuit additionally depends on the input data on which the circuit operates, and therefore becomes functional as well. Safety-related systems divide the operating mode into normal and emergency. For modern systems, this separation leads to different inputs in these modes and, as a result, to different functional checkability. This difference creates a problem of the hidden faults, which can accumulate in normal mode due to the lack of input data necessary for their manifestation. The problem starts in emergency mode on new input data, which detects accumulated faults in the amount exceeding the capabilities of fault-tolerant circuits [11, 12].

This problem creates a distrust of the fault tolerance of the components used in safety-related system. We can observe this in the use of dangerous imitation modes that recreate emergency conditions. Such an increase in checkability is carried out with emergency protection turned off, which has become one of the causes of the Chernobyl disaster. The danger of imitation modes has also been proved more than once by their unauthorized activation due to faults or human factor [13, 14].

A safe solution to the hidden fault problem can be obtained through a resource-based approach that examines the integration of models, methods and means that make up information computer resources into the natural world [15, 16].

According to this approach, resources are structuring according to the features of the natural world and two such features are most distinguished: parallelism and fuzziness. The objective structuring process can be traced to the development of floating-point formats, which transformed the codeword into its representation using two components: mantissa and exponent in the default number system [17, 18].

We can see this process also using the example of improving personal computers. They constantly increased the parallelization of circuit solutions in the processing of approximate data and made a jump from several floating-point pipelines in Pentium processors to several thousand such pipelines parallel working in the graphics processor [19].

The resource-based approach identifies replication and diversification levels in resource development. At the replication level, the integration of resources into the natural world occurs by copying them in conditions open to fill resource niches: market, technological, environmental and others. This process proceeds without restrictions from the natural world. The desire of resources for integration stimulates increased productivity in copying them. In the natural world, such cloning aims to survive through a higher birth rate than mortality.

The copy process ends when the resource niches are being closed. Typically, resource niches are closed at peak productivity. Clones perish. Their survival is possible only through the manifestation of features that raise them to the next level in resource development – diversification. Integration at this level takes place in close contact with the natural world, which structures resources according to its own features. Under these conditions, productivity is inferior to the priority of adequacy to the natural world, that is, trustworthiness.

The development of resources in today's computer world is more responsive to the level of replication and reflects the shortcomings of this lower level. Software products are copied to create new programs. Such copying clogs software products with redundant data and functions that the new software does not need. However, this replication process continues under conditions of open resource niches in productivity and memory capacity of the modern computer systems [20, 21], but is limited in mobile systems with the development of green technologies [22].

Hardware is also mainly at the replication level, which is represented by matrix structures. Arithmetic blocks are designed based on a pipeline organization of calculations, which belongs to the level of diversification. However, sections of modern pipelines contain matrix structures of parallel shifters and adders, iterative array multipliers and dividers, which perform arithmetic operations with data represented in parallel codes [23, 24].

These codes are parallel from the position of simultaneous availability of all their bits for data processing, which in matrix structures is performed both in parallel and in series. The iterative array multiplier of n-bit binary codes performs an operation in one clock cycle on a matrix of n^2 operational elements, each of which, due to successive connections, is used only for 1.6% and 0.8% of the time for $n = 32$ and 64, respectively [25, 26].

A further disadvantage of matrix structures is the large energy consumption. The static component of energy consumption is determined by the large dimensions of the matrices. The dynamic component depends on the number of signal transitions, the main part of which refers to parasitic [27, 28].

The main disadvantage of matrix structures is manifested in safety-related systems and is associated with the problem of the hidden faults. This problem is inherent only in these systems. Ordinary computers do not experience such a problem since the fault remains hidden throughout the operating mode.

The resource-based approach analyzes the problem of the hidden faults as a growth challenge. In safety-related applications, the systems rise to the level of diversification, in operating mode, input data and checkability. Components of the systems are lagging because they are designed using matrix structures at the replication level [29, 30].

Thus, the problem of the hidden faults must be solved as a challenge of growth by improving components to the level of their systems. The solution should be sought while maintaining matrix structures that have dominated in development of resources over several decades, including FPGA designing [31, 32].

The paper aims to develop a method to improve the checkability of FPGA projects with a LUT-oriented (Look-Up Table) architecture to eliminate hidden faults with the use of multi-version programming for the finished project. The basis of this method was discussed in [33, 34] for a single LUT unit. The proposed paper discloses the features of the method in its application to the FPGA project. Section 2 describes the main provisions of the method and the limitations arising from its application to the finished project. Section 3 demonstrates these features of the method application using the example of designing the iterative array multiplier.

2 Main Provisions of the Method

The method proposes to organize the operation of the FPGA project on several versions of program code, which change at a given periodicity, for example, every week. The purpose of the version change is to move the bits addressed in the LUT unit memory only in emergency mode to the place of the bits used during normal mode. Such an organization of the operation can be effective for components with slightly variable normal mode inputs and can serve as an alternative to manual regulation, which is used in practice for such components to improve their checkability. Manual regulation is performed for safety-related systems in power blocks of nuclear power plants no more than once in six months. The disadvantage of manual regulation is to change the input data only within the values allowed in the normal mode. Thus, faults occurring only in emergency mode are not detected, i.e. remain hidden.

The method is based on the features of FPGA projects with a LUT-oriented architecture that has version redundancy of program code, that is, the finished project can be programmed using different versions of program code. All versions completely retain the functionality of the project and do not make changes to its hardware component.

The LUT unit is the generator of the logical function of several variables that come to its inputs. For four variables, the LUT unit contains 4 inputs A, B, C and D. The function description is written to the LUT memory of the unit as program code in the FPGA programming process of the project [35, 36].

The version redundancy of the program code is inherent in each pair of LUT units for which the output of the first unit is the input of the second one. This pair provides two versions of program code without any influence on other LUT units of the FPGA project that are not connected to the output of the first LUT unit of the pair.

The versions differ in the initial or inverse value of the signal that propagates between the LUT units of the pair. The inverse signal value is provided by inverting all memory bits of the first LUT unit. The inversion thus obtained at the input of the second unit is compensated by changing the bit locations in the memory of the second LUT unit of the pair [37, 38].

The source data for this operation is the memory bit numbers of the LUT unit and the version numbers of its program code.

For four variables, the memory of the LUT unit contains program code composed of 16 bits, which can be numbered from 0 to 15. The whole set of versions of the second LUT unit of the pair is determined by the number of inputs used. For the four inputs connected to the outputs of the first LUT units, the second LUT unit of the pair can create 16 versions, which expediently to number with hexadecimal characters from 0_{HEX} to F_{HEX}. The binary code of these numbers contains 4 bits, which take the values "0" and "1" for the initial and inverse signal values at the inputs A, B, C and D of the LUT unit, respectively. The lower bit of the binary code corresponds to the A input.

The 0_{HEX} version is the source code of the LUT unit. Versions 1_{HEX}, 2_{HEX}, 4_{HEX} and 8_{HEX} are described by binary codes 0001_{BIN}, 0010_{BIN}, 0100_{BIN} and 1000_{BIN} which indicate inverting of one input of A, B, C or D, respectively. These versions divide the program code into 2^{4-X} of the same disjoint fragments 1, ..., 2^{4-X}, consisting of 2^X bits, where X accepts the values of 0, 1, 2 and 3 for the inputs A, B, C and D, respectively.

The bit location change is performed by the location change of each odd fragment with the next adjacent fragment.

Binary codes 0001_{BIN}, 0010_{BIN}, 0100_{BIN} and 1000_{BIN} are basic for obtaining any other nonzero code of the same size with the use of logical addition operation or addition on modulo two. The operation of the bit change can also be performed in a step-by-step sequence of such operations with the individual LUT inputs of the unit. The sequence of steps is irrelevant.

For example, the program code $ABBA_{HEX} = 1010\ 1011\ 1011\ 1010_{BIN}$ of the initial version 0_{HEX} can be converted to the version $B_{HEX} = 1101_{BIN}$ by performing operations with the sequential use of versions 1_{HEX}, 2_{HEX} and 8_{HEX} or 8_{HEX}, 2_{HEX} and 1_{HEX}.

In the first case, the program code will be divided into separate bits – fragments, and the initial version will be transformed into the 1_{HEX} version by places of neighboring fragments:

$$1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 0_{BIN};$$
$$0\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 1\ 1\ 0\ 1\ 0\ 1_{BIN}.$$

The 2_{HEX} version requires splitting the received program code into bit pairs and changing the locations of neighboring pairs:

$$01\ 01\ 01\ 11\ 01\ 11\ 01\ 01_{BIN};$$
$$01\ 01\ 11\ 01\ 11\ 01\ 01\ 01_{BIN}.$$

The 4_{HEX} version requires splitting the program code into fragments of eight bits and changing the places of neighboring fragments:

$$01011101\ 11010101_{BIN};$$
$$11010101\ 01011101_{BIN}.$$

In the second case, the program code goes through the following sequence of transformations:

$$10101011\ 10111010_{BIN};$$
$$10111010\ 10101011_{BIN};$$
$$10\ 11\ 10\ 10\ 10\ 10\ 10\ 11_{BIN};$$
$$11\ 10\ 10\ 10\ 10\ 10\ 11\ 10_{BIN};$$
$$1\ 1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 0_{BIN};$$
$$1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1_{BIN}.$$

In both cases, the result is the same:

$$1\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 0\ 1\ 1\ 1\ 0\ 1_{BIN} = D55D_{HEX}.$$

As a result of the permutation, bits 0, ..., 15 of the source program code swapped places with the following bits: 11, 10, 9, 8, 15, 14, 13, 12, 3, 2, 1, 0, 7, 6, 5, 4.

The initial data for the method is the data that arrives at the project FPGA inputs in normal and emergency mode, as well as a description of the digital circuit indicating the connections of the LUT units and their program codes.

The software implementation of the method performs simulation of the digital circuit operation on the input data of both modes and for each LUT unit determines the N_U set of memory bits used in normal mode and the E_A set of bits addressed only in emergency mode.

For each $Z \in E_A$ bit, the method determines the V_Z set of all possible $V \in V_Z$ versions of program code, which provide exchange of places with bits of the N_U set. From each non-empty V_Z set, one version is selected so that the number of different selected versions is minimal. The obtained set of versions ensures in normal mode the use of all bits of the LUT memory, addressed only in emergency mode, under the condition of non-empty V_Z sets. An empty V_Z set indicates that there is no version that allows you to swap the $Z \in E_A$ bit with the bit addressed in normal mode. Such LUT memory bits can accumulate hidden faults and pose a potential threat to functional safety of the FPGA project in emergency mode. The method determines all such bits and refers them to the $E_{N\text{-}R}$ set.

The proposed method is limited in the ability to increase the checkability of the FPGA project and reduce many hidden faults by the presence of empty V_Z sets. Such sets are inherent in LUT units that are not second units of any pair and therefore cannot create versions of their program code. The number of versions may also be insufficient for second LUT units if some inputs are connected to inputs of the FPGA project. The method determines all bits that do not have versions for permutation to the N_U set.

The second LUT units of the pair are fully provided with program code versions if all their used inputs are connected to the outputs of the first LUT units. In the case of unused inputs, the number of versions is halved for each such input. But at the same time, the used LUT memory of the unit is halved, which is provided by a full set of possible versions with complete elimination of hidden faults.

The possibilities of the method can be expanded if the inputs of the FPGA project are outputs of LUT units of other projects, that is, they can obtain initial and inverse signal values and thus create additional versions of program code.

3 Case Study of the Method

An experimental test of the method was carried out using the example of improving the checkability of the FPGA project implementing an iterative array multiplier of 4-bit binary codes. The iterative array multiplier was designed using the CAD Quartus Prime 18.1 Lite Edition on the Intel Max 10 FPGA 10M50DAF672I7G chip [39, 40]. The iterative array multiplier is implemented on 30 LUT units generating functions of four variables. LUT units use 111 inputs, 67 of which are connected to project FPGA inputs and reduce the number of possible program code versions. Only 2 LUT units create a complete set of possible versions.

The program implementation of the method, executed on Delphi 10 Seattle demo version [41], arranges the list of LUT units in the direction of their connection from the inputs to the outputs of the FPGA project and sequentially performs the functions of LUT units in this order on all values of input words composed of multiplier bits. The simulation performs 8 experiments for different values of threshold S, which divides the input words into those used in normal and emergency mode. Multipliers that do not exceed the threshold make up the input words processed in normal mode.

The plurality of N_U and E_A memory bits generated during the simulation process for each LUT unit serve to find versions that enhance the checkability of the FPGA project according to the proposed method. In addition, the method determines memory bits for which it is not possible to create versions that move them from the E_A set of the N_U one.

The results of the FPGA simulation of the project are shown in the main program panel (Fig. 1).

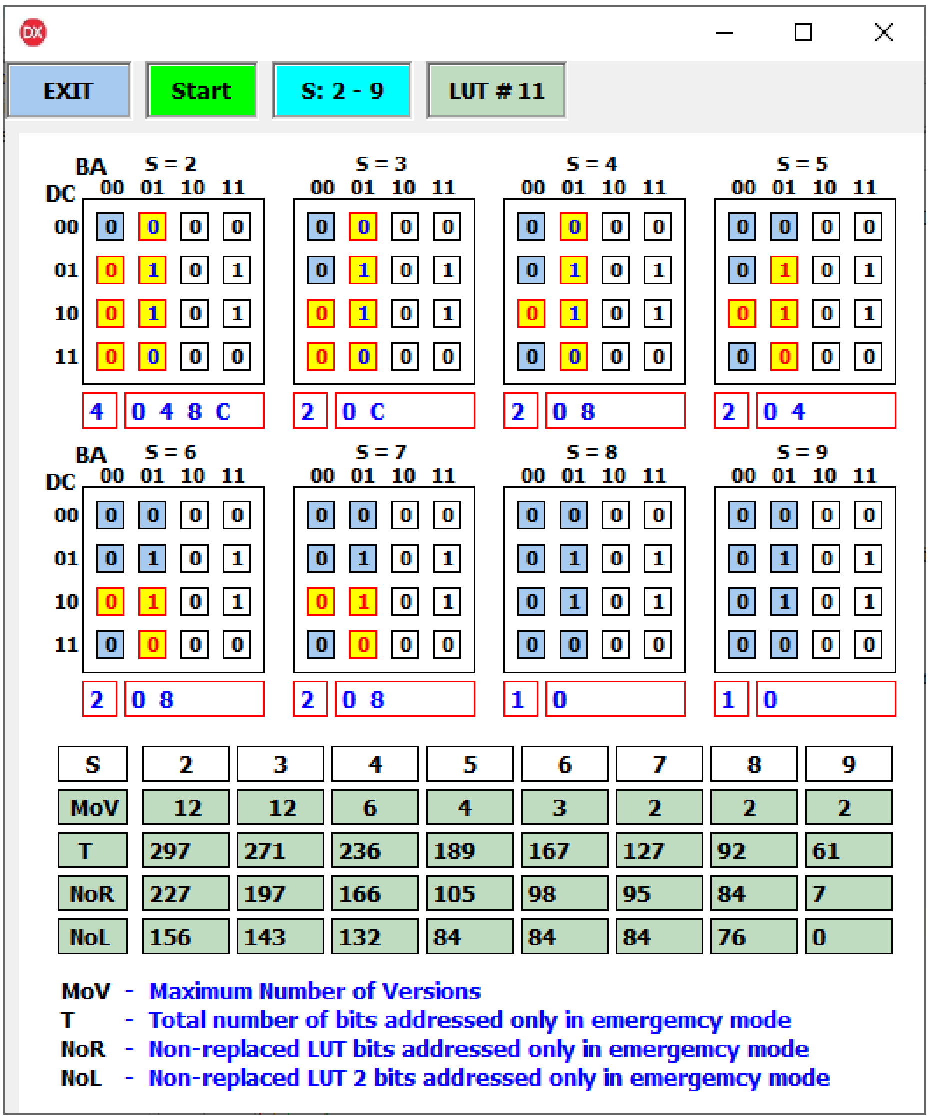
The main panel contains keys that show the threshold range S from 2 to 9 and the unit LUT number 11. Pressing these keys allows you to change the threshold values and number of the LUT unit in increments of 1 in a circle.

The panel below shows the memory bit matrices and their values in the selected 11 LUT unit for all threshold values. This LUT unit does not use input B and, accordingly, the right half of the memory. Input A is the input of the FPGA scheme of the project and reduces the number of possible versions by half. Inputs C and D are connected to the outputs of the previous LUT units and allow to create 4 versions of program code.

The bits of the sets N_U and E_A are colored blue and yellow, respectively. The values of the bits of the EA set are shown in red and blue for cases where there is or is no version for eliminating the hidden fault. The matrix below shows the number of versions used and their set, starting with the initial 0_{HEX} version.

In the case of threshold S = 2, the matrix contains only one bit 0 used in normal mode and 3 bits 4, 8, 12, which are addressed only in emergency mode. They can swap places with bit 0 using 4_{HEX}, 8_{HEX}, C_{HEX} versions for observation in normal mode. Bits 1, 5, 9, 13 of the next matrix column belong to the set E_{N-R}, since they are not provided with versions for their movement to the N_U set.

Increasing the threshold contributes to increasing the plurality of N_U bits and reducing the E_A set. For threshold S = 3 and S = 4, the number of additional versions is reduced to one: C_{HEX} or 8_{HEX}. Bits 1, 5, 9, 13 still do not have versions for observation in normal mode and belong to the E_{N-R} set.



S = 2

DC \ BA	00	01	10	11
00	0	0	0	0
01	0	1	0	1
10	0	1	0	1
11	0	0	0	0

4 | 0 4 8 C

S = 3

DC \ BA	00	01	10	11
00	0	0	0	0
01	0	1	0	1
10	0	1	0	1
11	0	0	0	0

2 | 0 C

S = 4

DC \ BA	00	01	10	11
00	0	0	0	0
01	0	1	0	1
10	0	1	0	1
11	0	0	0	0

2 | 0 8

S = 5

DC \ BA	00	01	10	11
00	0	0	0	0
01	0	1	0	1
10	0	1	0	1
11	0	0	0	0

2 | 0 4

S = 6

DC \ BA	00	01	10	11
00	0	0	0	0
01	0	1	0	1
10	0	1	0	1
11	0	0	0	0

2 | 0 8

S = 7

DC \ BA	00	01	10	11
00	0	0	0	0
01	0	1	0	1
10	0	1	0	1
11	0	0	0	0

2 | 0 8

S = 8

DC \ BA	00	01	10	11
00	0	0	0	0
01	0	1	0	1
10	0	1	0	1
11	0	0	0	0

1 | 0

S = 9

DC \ BA	00	01	10	11
00	0	0	0	0
01	0	1	0	1
10	0	1	0	1
11	0	0	0	0

1 | 0

S	2	3	4	5	6	7	8	9
MoV	12	12	6	4	3	2	2	2
T	297	271	236	189	167	127	92	61
NoR	227	197	166	105	98	95	84	7
NoL	156	143	132	84	84	84	76	0

MoV	-	Maximum Number of Versions
T	-	Total number of bits addressed only in emergemcy mode
NoR	-	Non-replaced LUT bits addressed only in emergemcy mode
NoL	-	Non-replaced LUT 2 bits addressed only in emergemcy mode

Fig. 1. Main panel of the program

The S threshold values from 5 to 7 allow all bits of the E_A set to be moved to the N_U set using one additional version: 4_{HEX} or 8_{HEX}.

Threshold S = 8 and S = 9 refers all memory bits to the N_U set and does not require the use of additional versions of program code.

Below the memory matrices, the main panel shows the overall results of the experiments conducted based on the analysis of all LUT units for each value of the S threshold.

The string "MoV" shows the maximum number of versions used in the LUT units of the entire project FPGA schema. The string "T" contains the total number of E_A bits that can accumulate hidden faults in the initial FPGA project. This number decreases from 297 to 61 with an increase in threshold S. The last two lines "NoR" and "NoL" show the total number of E_{N-R} bits for all LUT units and only for the second LUT units of the pair. These values decrease from 227 to 7 and from 156 to 0, respectively.

The improvement in project FPGA checkability can be estimated by the ratio of the number of bits $\Delta T = T - NoT$ for which the risk of hidden faults is eliminated to the total number T of bits addressed only in emergency mode. For example, in the case of S = 2, the amount $\Delta T = 297 - 227 = 70$, and the improvement of the checkability is equal to 70/297 = 23,6%.

For threshold values 2 – 7, the method increased the checkability of the FPGA project by an average of 27.5% from 13% (S = 3) to 44% (S = 5).

If it is possible to invert the signals at the project FPGA inputs, all memory bits of the LUT units are provided with versions to eliminate hidden faults and the checkability reaches 100%.

4 Conclusions

FPGA designing in the safety-related application domain encounters a problem of the hidden faults that can accumulate in memory bits of the LUT units during normal mode and reduce the fault tolerance of an FPGA project with a LUT-oriented architecture as well as functional safety of the safety-related systems in the most responsible emergency mode. This problem shows insufficient functional checkability of FPGA projects. The checkability deficiency is manifested in LUT units whose memory contains bits that are not observed during the normal mode, since they are used only in emergency mode. Fault tolerant FPGA components of safety-related systems become fail-safe only if they are checkable.

Treating the problem of the hidden faults as a growth challenge opens the way for it to be solved by increasing the level of FPGA components in their checkability to the level of diversification of the safety-related applications using multi-version technologies.

The proposed method uses the version redundancy of program code inherent in FPGA projects with a LUT-oriented architecture to increase their checkability by organizing work on several versions of the program code. All versions completely retain the functionality of the FPGA project and its hardware implementation. The method generates and selects versions that swap bits addressed only in emergency mode with bits observed during normal mode.

The method allows to eliminate non-checkable bits in the memory of all LUT units of the FPGA project when its inputs are connected to outputs of the LUT units of previous circuits.

The program implementation of the method was tested on the FPGA project of the iterative array multiplier. Simulation results showed an average of 27.5% increase in FPGA project checkability.

References

1. Jung, J., Ahmed, I.: Development of FPGA-based reactor trip functions using systems engineering approach, Nuclear Engineering and Technology, pp. 2-11 (2016)
2. Andina, J.: FPGAs: Fundamentals, Advanced Features, and Applications in Industrial Electronics. CRC Press, USA, Boca Raton (2017)
3. IEC 61508-1:2010. Functional Safety of Electrical / Electronic / Programmable Electronic Safety Related Systems – Part 1: General requirements. Geneva: IEC (2010).
4. Smith, D., Simpson, K.: The Safety Critical Systems Handbook, 5th Edition, Butterworth-Heinemann (2019)
5. Choe, S., Leite, F.: Assessing safety risk among different construction trades: Quantitative approach, Journal of Construction Engineering and Management 143(5), 04016133 (2017)
6. Ivanchenko, O., Kharchenko, V., Moroz, B. et. al.: Risk Assessment of Critical Energy Infrastructure Considering Physical and Cyber Assets: Methodology and Models. In: 10th IEEE International Conference IDAACS, Lviv, Ukraine, pp. 225-228 (2018)
7. Kotzanikolaou, P., Theoharidou, M., Gritzalis, D.: Cascading Effects of Common Cause Failures in Critical Infrastructures. In: International Conference on Critical Infrastructure Protection (ICCIP), Washington, DC, USA, pp.171-182 (2013)
8. IEEE Std1500-2005 Standard Testability Method for Embedded Core-based IC (2005) doi: 10.1109/IEEESTD.2005
9. Kondratenko, Y.P., Kozlov, O.V., Topalov, A.M., Gerasin, O.S. Computerized system for remote level control with discrete self-testing. In: CEUR Workshop Proceedings Open Access, vol. 1844, pp. 608-619 (2017) http://ceur-ws.org/Vol-1844/10000608.pdf
10. Romankevich, V.: Self-testing of multiprocessor systems with regular diagnostic connections. Automation and Remote Control, vol. 78, **2**, 289-299 (2017).
11. Drozd, A., Drozd, J., Antoshchuk, S. et. al.: Objects and Methods of On-Line Testing: Main Requirements and Perspectives of Development. In: IEEE East-West Design & Test Symposium, Yerevan, Armenia, pp. 72-76 (2016) doi: 10.1109/EWDTS.2016.7807750
12. Drozd, O., Antoniuk, V., Nikul, V., Drozd, M.: Hidden faults in FPGA-built digital components of safety-related systems. In: 14th IEEE International Conference TCSET, Lviv-Slavsko, Ukraine, pp. 805-809 (2018) doi: 10.1109/TCSET.2018.8336320
13. Gillis, D.: The Apocalypses that Might Have Been (2007) [Online]. Available: https://www.damninteresting.com/the-apocalypses-that-might-have-been/
14. Blakemore, E: The Chernobyl disaster: What happened, and the long-term impacts (2019) [Online]. Available: https://www.nationalgeographic.com/culture/topics/reference/chernobyl-disaster/
15. Drozd, J., Drozd, A., Al-dhabi, M.: A resource approach to on-line testing of computing circuits. In: IEEE East-West Design & Test Symposium, Batumi, Georgia, pp. 276-281 (2015) doi: 10.1109/EWDTS.2015.7493122
16. Drozd, A., Drozd, J., Antoshchuk, S. et. al.: Green Experiments with FPGA. In: Green IT Engineering: Components, Networks and Systems Implementation, SSDC, vol. 105, Springer International Publishing, Berlin, pp. 219-239 (2017) doi: 10.1007/978-3-319-55595-9_11
17. IEEE Std 754™-2008 (Revision of IEEE Std 754-1985) IEEE Standard for Floating-Point Arithmetic. IEEE 3 Park Avenue New York, NY 10016-5997, USA (2008)
18. Synopsys. DWFC Flexible Floating-Point Overview, no. August, pp. 1-6 (2016)
19. Ghorpade, J., Parande, J., Kulkarni, M., Bawaskar, A. GPGPU processing in CUDA architecture. Advanced Computing: An International Journal (ACIJ), vol. 3, **1**, 105-120 (2012)
20. Pomorova, O., Savenko, O., Lysenko, S., Kryshchuk, A., Bobrovnikova, K.: A technique for the botnet detection based on DNS-traffic analysis. In: CN 2015. CCIS, vol. 522, pp. 127-138. Springer, Heidelberg (2015)
21. Hovorushchenko, T, Pomorova, O. Ontological approach to the assessment of information sufficiency for software quality determination. SEUR-WS, vol. 1614, pp. 332-348 (2016)

22. Hahanov, V., Litvinova, E., Chumachenko, S.: Green Cyber-Physical Computing as Sustainable Development Model. In: Green IT Engineering: Components, Networks and Systems Implementation, SSDC, vol. 105, Springer International Publishing, Berlin, pp. 219-239 (2017) doi: 10.1007/978-3-319-55595-9_4
23. Palagin, A., Opanasenko, V.: The implementation of extended arithmetic's on FPGA-based structures. In: International Conference IDAACS, Bucharest, Romania, pp. 1014-1019 (2017)
24. Chernov, S., Titov, S., Chernova, L. et. al.: Algorithm for the simplification of solution to discrete optimization problems. Eastern-European Journal of Enterprise Technologies **3** (4), 1-12 (2018)
25. Neeraja, B., Sai Prasad Goud, R.: Design of an area efficient Braun multiplier using high speed parallel prefix adder in cadence. In: IEEE International Conference on Electrical, Computer and Communication Technologies, Coimbatore, India (2019)
26. Drozd, J., Drozd, A., Antoshchuk, S. et. al.: Effectiveness of Matrix and Pipeline FPGA-Based Arithmetic Components of Safety-Related Systems. In: 8th IEEE International Conference IDAACS. pp. 785-789. Warsaw, Poland (2015) doi: 10.1109/IDAACS.2015.7341410
27. Velegalati, R., Kaps, J.-P.: Glitch Detection in Hardware Implementations on FPGAs Using Delay Based Sampling Techniques. In: Euromicro Conference on Digital System, Design Los Alamitos, CA, USA (2013) doi: 10.1109/DSD.2013.107
28. Vasantha, K., Sharma M., Lal Kishore K.: A Technique to Reduce Glitch Power during Physical Design Stage for Low Power and Less IR Drop. In International Journal of Computer Applications (0975 – 8887), vol. 39, **18**, 62-67 (2012)
29. Tanasyuk, Y., Perepelitsyn, A., Ostapov, S.: Parameterized FPGA-based implementation of cryptographic hash functions using cellular automata. In 9th IEEE International Conference DESSERT, Kiev, Ukraine, pp. 238-241 (2018) doi: 10.1109/DESSERT.2018.8409133
30. Drozd, O., Kharchenko, V., Rucinski, A. et. al.: Development of Models in Resilient Computing. In: 10th IEEE International Conference on Dependable Systems, Services and Technologies, Leeds, UK, pp. 2-7 (2019) doi: 10.1109/DESSERT.2019.8770035
31. Amano, H.: Principles and Structures of FPGAs. Springer, USA, New-York (2018)
32. Vanderbauwhede, W., Benkrid, K.: High-performance computing using FPGAs. USA, New-York: Springer (2016)
33. Drozd, A., Antoshchuk, S., Drozd, J. et. al.: Checkable FPGA Design: Energy Consumption, Throughput and Trustworthiness. In: Green IT Engineering: Social, Business and Industrial Applications, SSDC, vol. 171, Springer, Berlin, pp. 73-94 (2019) doi: 10.1007/978-3-030-00253-4_4
34. Drozd, O., Kuznietsov, M., Martynyuk, O., Drozd, M.: A method of the hidden faults elimination in FPGA projects for the critical applications. In: 9th IEEE International Conference DESSERT, Kyiv, Ukraine, pp. 231-234 (2018) doi: 10.1109/DESSERT.2018.8409131
35. Cyclone Architecture. Cyclone Device Handbook, Volume 1. Altera Corporation (2008) https://www.intel.com/content/dam/www/programmable/us/en/pdfs/literature/hb/cyc/cyc_c51002.pdf
36. Intel FPGA Architecture (2019) https://www.intel.com/content/dam/www/ programmable/us/en /pdfs/literature/wp/wp-01003.pdf, last accessed 2019/07/30
37. Drozd, A., Drozd, M., Kuznietsov, M.: Use of Natural LUT Redundancy to Improve Trustworthiness of FPGA Design. In: CEUR Workshop Proceedings. 1614, 322-331 (2016)
38. Zashcholkin, K., Drozd, O. The Detection Method of Probable Areas of Hardware Trojans Location in FPGA-based Components of Safety-Critical Systems. In: IEEE International Conference DESSERT, Kyiv, pp. 212-217 (2018) doi: 10.1109/DESSERT.2018.8409130
39. Intel Quartus Prime Standard Edition User Guide. https://www.intel.com/content /dam/www/programmable/us/en/pdfs/ literature/ug/ug-qps-getting-started.pdf.
40. Max.10 FPGA Device Architecture (2017) https://www.intel.com/content/dam /www/programmable/us/en/pdfs/literature/hub/max-10 /m10_architecture.pdf.
41. Delphi 10 Seattle: Embarcadero https://www.embarcadero.com/docs/datasheet.pdf