Using pipelined control code generator as a reference system in FPGA’s selection as a hardware platform for fault-tolerance computing system

O Brekhov and M Ratnikov*
Department 304, Moscow Aviation Institute (National Research University), 4 Volokolamskoe Shosse, Moscow, 125993, Russian Federation

*E-mail: ratnikovmo@mai.ru

Abstract. One of the main features of hash functions is a significant change in the hash with any (even small) changes in the input data. Using their pipelined versions (which outputs the next hash on each clock cycle) allows to create a system that is guaranteed to respond to any failures in the process. Reference systems based on control code generators can be quickly developed and thanks to a wide range of functions, for calculating control codes, can be close to the characteristics of the developed system in terms of area and logical complexity. In this article, we propose using pipelined control code generators as reference systems in the development of fault tolerant computing systems. In this case, before or simultaneously with the development of a computing system, we need to perform a series of experiments with pipelined generators and control codes. Several versions of such reference system have been developed, each of which implements various fault tolerance techniques. The obtained protected reference systems are synthesized into the basis of available FPGAs by various software packages. Next, the results are compared and the most successful (with good results) sets are selected from FPGAs, software, and fault tolerance techniques.

1. Introduction
Currently, field-programmable gate arrays (commonly abbreviated as FPGA) are one of the most common platforms for the development and prototyping [1-2] of small-scale computing systems, including systems operating in conditions of negative external influence like as aerospace applications. Such systems are called protected and characterized by the use of additional fault tolerance techniques using hardware or temporary redundancy. Usually, when developing them, you first develop and debug a fully functional unsecured system. Then, the selected fault tolerance techniques are integrated into it.

FPGAs of different models (and vendors) usually have significantly differ characteristics (architecture, amount of available resources, operating frequency, power consumption, available IP blocks, etc.). This causes the characteristics of systems based on different FPGAs to be very different after the implementation fault tolerance techniques. Accordingly, the developer must solve a complex multi-criteria problem of choosing the optimal pair of FPGAs and a fault tolerance technique.

Another characteristic feature of FPGAs is that they are used as complete devices, i.e. the user can configure it to suit his needs, but cannot change the architecture of the chip, form factor, the number of available resources, etc. Therefore, with all the flexibility of FPGAs as a development platform, the development process itself and the software used in the process are set by the chip manufacturer. Therefore, the FPGA is, in fact, a software and hardware platform, since chips from any of the suppliers.
cannot be used in isolation from the supplied and/or available software (by the FPGA’s vendor or by 3rd party SW-developer). The software toolkit for working with FPGAs can be divided into the following main areas: source code development (compilation and simulation), firmware development (compilation, synthesis) and complete device debugging.

For the tasks discussed in this article, the software toolkit that compiles and synthesizes a netlist from RTL is of the greatest importance. Now, for all common FPGAs, the choice is limited to a very small number of options. These are software toolkits from the FPGA manufacturer, from large companies (Mentor Graphics, Cadence and Synopsys) and, sometimes, 1-2 options from small companies or from the open-source community.

When developing a fault-tolerant FPGA-based computing system the requirement for correct processing of the implemented fault tolerance technique is added to the set of requirements for the FPGA employed and the software toolkit. Considering that the selected fault tolerance technique is implemented in ready-made unprotected systems the troubles that may arise at this stage can lead to serious costs (replacement of FPGA and/or the software for it, replacement of the method of FPGA fault tolerance technique, associated processing of the developed system, its retesting, etc.).

Thus, before starting the development of a functional description, we can make conclusions about the possibility of using software, FPGAs and fault tolerance techniques. This will allow the developer to cut off options that definitely cannot be used or implemented. The final decision on the choice of FPGA and software toolkit should be made before start of RTL development and a fault tolerance technique can be selected after the development of a fully functional unprotected system is completed. It is based on the knowledge of the precise characteristics of the designed system, fault tolerance requirements, etc. The methodology for making such a decision is beyond the scope of this work.

We will consider the following terms:

1) We divide the problem of analysis (finding the optimal set of FPGA, software toolkit and fault tolerance technique) in several stages. In this paper, we will consider the first (preliminary) stage;
2) The list of possible solutions (available sets of FPGA, fault tolerance technique and software toolkit) should be determined as early as possible since many decisions depend on them;
3) The task at this stage of analysis is solved without real integration of the analyzed fault tolerance technique in the developed system;
4) We should minimize the time spent on the preliminary selection stage;
5) The permissible error during estimation should be no more than 35-40%.

Note relating to the error value - at this stage, our task is to evaluate the possibility of using this set, and not to obtain accurate values. If, as a result of the analysis, it becomes clear that the resources are obviously not sufficient or that the frequency of work is much less than that required or that the experiment failed (an error at the synthesis or compilation stage), then such a set should not be used. If at the stage of preliminary analysis, the resulting value is on the threshold of acceptable or better, then this method should be investigated at the main stage.

So, the task of FPGA, software toolkit and fault tolerance technique selection should be solved as early as possible, preferably simultaneously with the development of the RTL computing system, in order to minimize the costs of the last stages of computing system development. We propose to do this by performing all necessary checks on some reference system. That system should have well-known behavior, easily controlled and to work correctly with the existing FPGA, software toolkit and fault tolerance technique.

Currently, many works are devoted to the task of FPGAs fault tolerance technique ([3-5] for example), fault tolerance technique selection [6-8] and FPGA's failures modeling [6, 7, 9, 10]. In most of them, fully functional target systems or their parts are used as test systems (the systems specially designed for analysis) [5-7, 8, 10]. But such systems can't always guarantee the absence or detection of gray failures [9]. There are, additionally, a number of works devoted to testing FPGAs and fault tolerance techniques using special test systems [4, 5, 7].

The test systems used for testing FPGAs and their system environment, as well as for various experiments with the system architecture, are based on the following ideas:
• Implementation of a shift register. For example, a shift register with feedbacks from [11] or a shift register of [7]. These systems do not solve the problem of fault detecting. Also, if you need to increase the volume of the test system, there is a possibility that optimization during synthesis will remove part of the scheme.

• Implementation of the ALU sequence. For example, sequence of adders and multiplexers from [11] or system 3 of [7] with multipliers and shift registers. The disadvantages are similar to those mentioned above.

• Self-monitoring test system implementation. For example, in [12], but MBIST systems are relatively slow and are not suitable for analyzing the system architecture (since they are fixed to the blocks under test).

• Implementation of the control task (target system or its parts). For example, DSP from [11], or full functional target System-on-Chip from [10]. This approach provides the greatest accuracy of the analysis, but requires a ready-made target system. Also it will lead to difficulties when experimenting with fault tolerance techniques (this would involve significant work for every analyzed technique).

In some cases, systems are used that include features of the systems shift-registers and ALU based systems, for example, the CRC generator from [13]. The calculation of the CRC is well suited for solving the task of fault detecting. Also, synthesis’s optimization does not interfere with it, but there is a problem with scalability – the block cannot be increased if you need to create an enlarged test system. We can implement several such blocks, but then you need to solve the task individually for each block with input data generation and controlling the results. So we propose to use pipelined control or hash (not only CRC) code generator [14].

Pipelined control code generator is a computer system that calculates the value of a hash function or control code at each clock cycle. The input data are submitted for each of the stages of the pipeline. In such systems, each stage of the pipeline implements one step of the algorithm for calculating the control code. All data required for this step comes either from the previous step or from the inputs of the test system. In such systems, there are no elements that store data for more than one clock cycle, which allows to quickly fault detection that occurred during the calculation and respond to changes in the input data.

In this paper, we proposed to use a pipelined control codes generator to solve the problem of analytical evaluation of fault tolerance techniques at the preliminary stage.

2. Methods

Let us define the concepts of a target system and a test system. A target system is a fully functional system that we need to develop. A test system is a system specifically designed for experimentation and analysis of various fault tolerance techniques. We propose the following method of rapid analysis of changes in the characteristics of the system.

2.1. Step 1.

Select a test system based on a pipelined control code generator.

The system has the following requirements.

1) Easy to implement
   It is necessary to reduce the time spent on the implementation of the test system. At the same time, it is necessary to ensure the ability to quickly replace unprotected elements with protected ones.

2) Area
   Area of a test system must be greater than or equal to the area of the target system, because the smaller test system, the more errors in the estimation.
   - A secure test system should require no more resources than the chip has. Otherwise, there may be an error in the analysis due to the peculiarities of the synthesizer.

3) The bit width.
The registers width used of the test system should be as close as possible to the registers width of the target system.

To improve the accuracy of the analysis, it is necessary to maintain the ratio of registers of different bit sizes in the target and test system. For example, if the test system is expected to be dominated by single-digit registers, then it is desirable to use them in the test system.

If the pipelined algorithm used for calculating the control codes does not initially allow the use of registers of the desired bit depth (most control code generators rigidly determine the bit depth of the registers used in the calculation), it is proposed to divide the registers into two parts: protected and unprotected. In this case, the analyzed method of fault tolerance technique is applied only to the protected part, and the involvement of the unprotected part in the further analysis is minimized. If the target system uses elements with a bit depth greater than that specified by the algorithm of the test system, then we should either implement the fragment by combining several registers of each stage of the pipeline into one protected fragment, or take into account the possible error of the analysis.

4) Logical load.

The ratio of protected and unprotected fragments in the test and target systems should be comparable (the ratio of the area of all protected and all unprotected elements in the unprotected test and target systems should be as close as possible to 1).

\[
\frac{S_{prot\ part\ test}}{S_{unprot\ part\ test}} \approx \frac{S_{prot\ part\ target}}{S_{unprot\ part\ target}} \quad (1)
\]

5) Coding style.

The coding style must match that of the target system. In other words, the test system should, if possible, be developed by the same people who will develop the target system. This limits the use of third-party test systems, but increases the efficiency of the evaluation. Also, it should be noted that, as far as possible, the test system should use the same HDL capabilities and the same IP cores that will be used on the target system. In view of this requirement, the requirement of simplicity of the test system is of particular importance.

2.2. Step 2.

Implement the analyzed method of reliability assurance into the test system.

2.3. Step 3.

Determine the relative change in area as:

\[
k_{prot\ impl} = \frac{S_{test\ system\ prot}}{S_{test\ system\ base}} \quad (2)
\]

where \(k_{prot\ impl}\) is the relative change of area, \(S_{test\ system\ prot}\) is the area of the protected test system, \(S_{test\ system\ base}\) is the area of the unprotected test system.

2.4. Step 4.

Determine the area of the unprotected target system (\(S_{target}\)) analytically and experimentally.

The experimental method will allow a system’s designer to get the exact area of the target system in the corresponding hardware basis. To do this, we need to synthesize its functional description in the Hardware Description Language (HDL). In the case of a performed analysis assessment without a ready target system, its area is determined analytically. For example, this can be done by the method described in [15]. As an alternative way of estimating hardware costs for the implementation of the target system, an estimate based on the ratio of register area and combinational logic between them is proposed. Usually this ratio ranges from 70: 30 to 40:60 [16], often the optimal and most common ratio is called 66:33 or 2:1 [17]. Then, knowing the number of registers in the system, which can be estimated after determining the algorithm of the system or developing a model in a high-level language, of the area
occupied by these registers and the ratio of the area of registers and logic elements, it is possible to calculate the area of the entire system.

2.5. Step 5.
Calculate the estimated area of the protected test system. To do this, we transform formula (2) as:

$$S_{target\ system\ prot} = k_{prot\ impl} \times S_{target\ system\ base}$$

(3)

where $k_{prot\ impl}$ is the relative change of area, $S_{target\ system\ prot}$ is the area of the protected target system, $S_{target\ system\ base}$ is the area of the unprotected target system.

2.6. Step 6.
Evaluate the timing characteristics. When operating from a single clock signal, the maximum possible frequency of the system is determined by the frequency of the slowest of the elements. Thus the task of analyzing the change in time characteristics is reduced to finding the lowest value of the frequency of operation:

- protected test system;
- unprotected target system;
- protected elements used in a protected target system.

We will propose a method using two variants relating to an area of unprotected target system: experimental (obtained as a result of the synthesis of a complete functional description) and analytical (obtained by methodology based on the ratio of the number of resources used for registers and for combinational circuits). Next, we estimate the error of the obtained values in relation to the reference value – the area of the protected target system.

Similarly, we will evaluate the timing characteristics of the target system based on experimental data (obtained during the synthesis of the target design) and analytical data (based on the results of the synthesis of the test system and protected registers of different bit widths).

3. Calculations

3.1. Baseline

3.1.1. Hardware platform. The estimation will be made for three FPGAs of different manufacturers: Microsemi (Actel) APA 1000 CQFP352, Intel (Altera) Stratix EP1S25F672C, Xilinx Virtex XQR4VSX55CF1140.

3.1.2. Fault tolerance techniques. We will demonstrate the application of the proposed method by the example of the implementation of a fault-tolerance system based on the use of Hamming code that corrects one error and detects two errors. The protected element in this system are registers. The protected register includes encoder, decoder, register for storing protected data, multiplexer and the self-recovery circuit. If there is no data to write, the register overwrites the stored data at each clock cycle (each time the data word is encoded and decoded), which ensures the self-recovery of the recorded data.

3.1.3. Target system. Let's take the block of the input data streams formatted as a test device. In accordance with the specified mode, it collects data from two sources (data from one of them is buffered) and transmits to the output. It is design written in the language SystemVerilog 2005, has two FSM inside and registers of different bit sizes: from 1 bit to 64. The input data is received via an 8-bit interface, the output is transmitted via a two-bit interface. Commands to the control circuit are received via two data-strobe lines. The probable total number of bits in all registers of the block is 500-700 bits. A general view of target system is shown in figure 1.
List of registers width and number and methods for calculating these values, will be shown later in this article.

3.1.4. Software toolkit. In this study, we used two software packages that perform RTL compilation and synthesis: Mentor Graphics Precision 2013 and Mentor Graphics Precision 2015.

3.2. Application of the methodology
Selection of test system. For the above system, there are no requirements for a high degree of pipelining, minimizing hardware resources, etc. in this case, the most likely ratio between the volume of logical elements and registers is 2:1. Taking into account the above requirements, we propose to use a pipelined MD4 hash code generator for further analysis [18]. The main advantages of the MD4 pipelined generator are sufficient logical load; the ratio of the cost of logical elements and registers 2:1; simplicity of implementation; and possibility to use "mixed" registers (each stage has 4 registers, so developer can select protectable part of each register in stage).

The width of the registers protected part selection. The bits of the protected parts of the test system registers are selected so as to roughly preserve the ratio between the register widths in the test system. In accordance with the register width requirement, the following register types are used in the developed MD4 pipeline: 32 bit protected register; 16-bit protected register (the remaining 16 bits are not in the protected part); and 1 bit protected register (31 bits are outside the protected part).

3.2.1. Developing of test systems. The test system was developed using SystemVerilog 2005 Hardware Description Language. We implemented 2 versions of the test system: unprotected and protected. Test system’s source codes (for unprotected and protected versions) are available at [19].

3.2.2. Synthesis results of the test system. The results of synthesis of pipelined protected and unprotected pipelined MD4 hash code generator are given in tables 1 and 2.

Table 1. Test system synthesis results (Mentor Graphics Precision 2013).

| Value                        | Microsemi (Actel) | Intel (Altera) | Xilinx  |
|------------------------------|-------------------|----------------|---------|
| Unprotected system area      | 33291 Tiles       | 8662 LCs       | 2978 CLB|
| Protected system area        | 66297 Tiles       | 29130 LCs      | 10315 CLB|
| Ratio of protected and unprotected systems area | 1.991 | 3.362 | 3.463 |
Table 2. Test system synthesis results (Mentor Graphics Precision 2015).

|                     | Microsemi (Actel) | Intel (Altera) | Xilinx |
|---------------------|-------------------|----------------|--------|
| Unprotected system area | 33291 Tiles       | 8662 LCs       | 2978 CLB |
| Protected system area     | 66297 Tiles       | 27950 LCs      | 10315 CLB |
| Ratio of protected and unprotected systems area | 1.991             | 3.2267         | 3.6     |

A small comment about the results of the synthesis. As is seen, the results of the synthesis in this case are practically independent of the software version, with the exception of one case. But in the process of synthesis, a number of features of the software were revealed. For example, several times the synthesizer was unstable when working with a protected system. This was corrected by restarting the software and performing a re-synthesis. In some cases, the problem was solved by changing the compilation order of the project files. Also, the unstable operation of both software versions with macros (set via the define directive in the synthesizer command line) was detected. This problem was solved by using a header file that was attached along with other source code files. Subsequently, this information was taken into account when developing the target system.

3.2.3. Determination of target system area. To continue the evaluation, we need to obtain data on the area of the unprotected target system. This can be done experimentally (if the system development is complete) or analytically. Next, we defined analytically how much the characteristics of the target system change after the implementation of the method of fault tolerance technique. To do it we evaluated the registers of different bit sizes used in the system and their number. In our case, we used the exact values, but if there is no such data, it is possible to use an expert opinion or any other available assessment methodology (for example [20-22]). To calculate the area of such registers, it is necessary to conduct an experiment to obtain the area of registers of different widths. In our case, for registers with a bit width from 1 to 64. The results are presented in table 3. Also in these tables are shown widths and numbers of registers of each width in the block. The line "Total for the system" shows the area of all registers in the system.

Table 3. Target system register area.

| Width | Registers number | Mentor Graphics Precision 2013 | Mentor Graphics Precision 2015 |
|-------|------------------|--------------------------------|--------------------------------|
|       | Microsemi (Actel) | Intel (Altera) | Xilinx | Microsemi (Actel) | Intel (Altera) | Xilinx |
| 1     | 8                | 4                 | 2       | 1                 | 4                | 2       |
| 2     | 2                | 7                 | 3       | 1                 | 7                | 3       |
| 3     | 1                | 10                | 4       | 2                 | 9                | 4       |
| 4     | 1                | 13                | 5       | 2                 | 10               | 5       |
| 5     | 1                | 16                | 6       | 3                 | 13               | 6       |
| 6     | 1                | 19                | 7       | 3                 | 16               | 7       |
| 8     | 2                | 25                | 9       | 4                 | 25               | 9       |
| 12    | 3                | 41                | 13      | 6                 | 41               | 13      |
| 14    | 1                | 50                | 15      | 7                 | 50               | 15      |
| 64    | 8                | 212               | 65      | 32                | 212              | 65      |
| Total for system | 2023             | 636               | 309     | 2013              | 636             | 309     |

In the table 4, we wrote the values calculated by Mentor Graphics Precision 2013 and Mentor Graphics Precision 2015 toolkits. The values calculated above are recorded in the line 1 of table 4. The values in the line 2 are equal to the product of the number of resources spent on registers by a factor of 3 (based on the 2:1 ratio between the cost of logical elements and registers). Line 3 of the table 4 shows...
the values obtained from the synthesis of the target system in the basis of the corresponding FPGAs. The values in rows 5 and 6 are calculated by multiplying the coefficient from row 4 by the values from rows 1 and 2, respectively.

To determine the error of the experiment, the implementation of the analysed fault tolerance technique into the target system was carried out. After that, it was synthesized into the appropriate basis. The resulting values are shown at line 7 and 6 of the table 4. The estimation errors (relative error), calculated for the values, obtained during the analysis (lines 5 and 6) and the actual size of the system (line 7), are recorded in lines 8 and 9 of the table 4, respectively.

### Table 4. Results of estimation of target system area.

| Value                                             | Mentor Graphics Precision 2013 | Mentor Graphics Precision 2015 |
|---------------------------------------------------|-------------------------------|--------------------------------|
|                                                   | Microsemi (Actel) | Intel (Altera) | Xilinx | Microsemi (Actel) | Intel (Altera) | Xilinx |
| Registers area (estimated value)                  | 2023 Tiles | 636 LCs | 309 CLB | 2023 Tiles | 636 LCs | 309 CLB |
| Unprotected system area (estimated value)         | 6069 Tiles | 1908 LCs | 927 CLB | 6069 Tiles | 1908 LCs | 927 CLB |
| Unprotected target system area                    | 6017 Tiles | 2176 LCs | 849 CLB | 6017 Tiles | 2176 LCs | 849 CLB |
| Ratio of protected and unprotected system areas   | 1.991 | 3.362 | 3.463 | 1.991 | 3.362 | 3.463 |
| Area of protected target system (estimated value  | 12086 Tiles | 6416 LCs | 3210 CLB | 12086 Tiles | 6416 LCs | 3210 CLB |
| (based on estimated area of unprotected system)   | 11982 Tiles | 7317 LCs | 2940 CLB | 11982 Tiles | 7317 LCs | 2940 CLB |
| Area of protected target system (estimated value  | 14050 Tiles | 6496 LCs | 3203 CLB | 14050 Tiles | 6496 LCs | 3203 CLB |
| (based on exact area of unprotected system)       | 0.139 | 0.0122 | 0.002 | 0.139 | 0.0122 | 0.002 |
| Area of protected target system (based on exact   | 0.147 | 0.1265 | 0.081 | 0.147 | 0.1265 | 0.081 |
| unprotected system area)                          | 0.139 | 0.0122 | 0.002 | 0.139 | 0.0122 | 0.002 |

The error in the assessment did not exceed 20%. For the technique used at the preliminary stage, this is an acceptable result.

### 3.3. Evaluation of time characteristics

The frequency of protected fragments, target system and protected test system is shown in table 5.

As reader may notice, in this case in all experiments, precision showed the same frequency values regardless of the version. As well as in the area estimation the error does not exceed that specified at the beginning of the article.
Table 5. Frequency (MHz) for devices by vendors.

| Design                          | Mentor Graphics Precision 2013 | Mentor Graphics Precision 2015 |
|---------------------------------|--------------------------------|--------------------------------|
|                                 | Microsemi (Actel) | Intel (Altera) | Xilinx | Microsemi (Actel) | Intel (Altera) | Xilinx |
| 1 bit register (protected)      | 113.973           | 116.279        | 377.501 | 113.973           | 116.279        | 377.501 |
| 2 bits register (protected)     | 87.689            | 211.64         | 248.818 | 87.689            | 211.64         | 248.818 |
| 3 bits register (protected)     | 92.739            | 214.362        | 223.814 | 92.739            | 214.362        | 223.814 |
| 4 bits register (protected)     | 86.618            | 213.447        | 214.454 | 86.618            | 213.447        | 214.454 |
| 5 bits register (protected)     | 67.304            | 163.639        | 159.566 | 67.304            | 163.639        | 159.566 |
| 6 bits register (protected)     | 69.754            | 162.048        | 177.085 | 69.754            | 162.048        | 177.085 |
| 7 bits register (protected)     | 65.463            | 162.048        | 157.803 | 65.463            | 162.048        | 157.803 |
| 8 bits register (protected)     | 63.784            | 144.844        | 182.949 | 63.784            | 144.844        | 182.949 |
| 12 bits register (protected)    | 56.634            | 140.766        | 131.874 | 56.634            | 140.766        | 131.874 |
| 14 bits register (protected)    | 54.651            | 128.584        | 139.743 | 54.651            | 128.584        | 139.743 |
| 16 bits register (protected)    | 52.392            | 128.254        | 139.276 | 52.392            | 128.254        | 139.276 |
| 32 bits register (protected)    | 50.188            | 118.064        | 148.854 | 50.188            | 118.064        | 148.854 |
| 64 bits register (protected)    | 46.891            | 108.897        | 118.497 | 46.891            | 108.897        | 118.497 |
| Unprotected target system       | 56.04             | 176.11         | 181.85  | 56.04             | 176.11         | 181.85  |
| Protected test system           | 33.12             | 67.23          | 78.32   | 33.12             | 67.23          | 78.32   |
| Minimum frequency calculated during analysis (with target system synthesis results) | 33.12 | 67.23 | 78.32 | 33.12 | 67.23 | 78.32 |
| Minimum frequency calculated during analysis (without target system synthesis results) | 33.12 | 67.23 | 78.32 | 33.12 | 67.23 | 78.32 |
| Protected target system (experimental) | 29.95 | 80.37 | 87.29 | 29.95 | 80.37 | 87.29 |
| Estimation error                | 0.105             | 0.163          | 0.1027  | 0.105             | 0.163          | 0.1027  |

4. Conclusions

The paper proposes a method of using pipelined control code generators as reference systems in the development of fault tolerant computing systems. This method allows the checking work of the design path before the end of the development of the target full-featured system, evaluate the possible change in the characteristics of the computer system after the implementation of the fault tolerance method within it, and select the most promising sets of software, FPGAs and the fault tolerance technique for further analysis.

The analysis and evaluation of the results presented in this paper showed that the method allows the performance of a qualitative assessment of various sets of FPGAs, tool software and the method of fault tolerance technique.
Testing and approbation of the developed test system confirmed the fulfillment of the requirements imposed on it. It also avoids the disadvantages of conventional test systems and by design provides error detection and does not require a complete target system and provides the necessary flexibility.

References
[1] Brekhov O and Klimenko A 2016 Hardware-software simulation complex for fpga-prototyping of fault-tolerant computing systems. Comm. Com. Inf. Sc. 678 454 doi: 10.1007/978-3-319-51917-3_40
[2] Brekhov O and Klimenko A 2018 Fault tolerant ASIC/ULA-based computing systems testing via FPGA prototyping with fault injection. Adv. Intell. Syst. 582 60 doi:10.1007/978-3-319-59415-6_6
[3] Anjankar S, Pand A, Junghare R and Zalke J 2018 Real-time FPGA-based fault tolerant and recoverable technique for arithmetic design using functional triple modular redundancy (FRTMR). Adv. Intell. Syst. 712 491 doi: 10.1007/978-981-10-8228-3_45
[4] Ibe E, et al. 2019 Radiation-induced soft errors. VLSI Design and Test for Systems Dependability. ed Asai S (Tokyo: Springer) pp 57-127 doi: 10.1007/978-4-311-56594-9_3
[5] Garcia P, Gomes T, Salgado F, Cabral J, Cardoso P, Ekpanyapong M and Tavares A 2012 A fault tolerant design methodology for a FPGA-based softcore processor, IFAC Proceedings 45 (4) 145 doi: 10.3182/20120403-3-DE-3010.00005
[6] Wulf N, George A and Gordon-Ross A 2016 A framework for evaluating and optimizing FPGA-Based SoCs for aerospace computing. Acm. T. Reconfig. Techn. 10(1) 1 doi: https://doi.org/10.1145/2888400
[7] Zhongming W, Zhibin Y, Hongxia G and Min L 2011 A software solution to estimate the SEU-induced soft error rate for systems implemented on SRAM-based FPGAs. Journal of Semiconductors 32(5) 055008 31
[8] van Harten L, Mousavi M, Jordans R and Poursahebaghi H 2018 Determining the necessity of fault tolerance techniques in FPGA devices for space missions. Microprocess. Microsys. 63 1 https://doi.org/10.1016/j.micropro.2018.08.001 doi: 10.1088/1674-4926/32/5/055008
[9] Bernardeschi C and Domenici A 2019 Application of model checking to fault tolerance analysis. Software Engineering to Formal Methods and Tools, and Back. Lecture Notes in Computer Science ed ter Beek M, Fantechi A and Semini L (Cham: Springer) vol 11865 pp 531-547 https://doi.org/10.1007/978-3-030-30985-5_1
[10] Podivinsky J, Cekan O, Simkova M and Kotasek Z 2014 The Evaluation platform for testing fault-tolerance methodologies in electro-mechanical applications Microprocess. Microsys. 39 (8) 1215 https://doi.org/10.1016/j.micropro.2015.05.011
[11] Pratt B, Caffrey M, Graham P, Morgan K and Wirthlin M 2006 Improving FPGA design robustness with partial TMR. IEEE Int. Reliat Phy. Sym. Proceedings. 2006 226 doi:10.1109/RELPHY.2006.251221
[12] Stott E, Sedcole P and Cheung P 2010 Fault tolerance and reliability in field programmable gate arrays. Iet. Comput. Digit. Tec. 4(3) 196 doi: 10.1049/iet-cdt.2009.0011
[13] Bernardeschi C, Cassano L, and Domenici A 2011 Failure probability of SRAM-FPGA systems with stochastic activity networks. IEEE Int. Symp Design 2011 293 doi: 10.1109/DDECS.2011.5783098.
[14] Brekhov O, Ratnikov M, 2014 Pipelined error-detecting codes in FPGA testing. Adv. Electr. Comput. En. 14 57 doi: 10.4316/AECE.2014.02010
[15] Schumacher P and Jha P 2008 Fast and accurate resource estimation of RTL-based designs targeting FPGA Int. Con. Field Prog. 2008 59 doi: 10.1109/FPL.2008.4629908
[16] Maniwa T 1998 Matching the design size to an ASIC or FPGA. Integrated System Design, available at: http://www.pldworld.com/_hdl/1/RESOURCES/QUALIS/library/ASICColumn9806.html
[17] White Paper FPGA Architecture ver. 1.0 WP-01003-1.0 2006, Altera, available at https://www.intel.cn/content/dam/www/programmable/us/en/pdfs/literature/wp/wp-01003.pdf

[18] Rivest R 1992 The MD4 Message-Digest Algorithm MIT Laboratory for Computer Science and RSA Data Security, Inc., available at: https://tools.ietf.org/html/rfc1320

[19] Authors source code repository, At GitHub, available at: https://github.com/MaksimRatnikov/PipelinedHashCodeGenerator

[20] Nayak A, Haldar M, Choudhary A and Banerjee P 2002 Accurate area and delay estimators for FPGAs. Design, Automation and Test in Europe Conf. and Exhibition 2002 862 doi: 10.1109/DATE.2002.998400

[21] Brandolese C, Fornaciari W and Salice F 2004 An area estimation methodology for FPGA based designs at systemc-level. Des. Aut. Con. Proceedings (New York: Association for Computing Machinery) pp 129-132 doi: https://doi.org/10.1145/996566.996606

[22] Rollins N, Wirthlin M and Graham P 2004 Evaluation of power costs in applying TMR to FPGA designs, Proc. 7th Annual Military and Aerospace Programmable Logic Devices Int. Conf. (Washington: NASA Office of Logic Design) p 136