Verification methods for complex-functional blocks in CAD for chips deep submicron design standards

Vladimir Zolnikov[1*], *Konstantin* Zolnikov[1], *Nadezhda* Ilina[2], and *Kirill* Grabovy[3]

[1]Moscow Aviation Institute (National Research University), 4, Volokolamskoe shosse 125993, Moscow, Russia
[2]Voronezh State Technical University, 20-letiya Oktyabrya Street, 84, 394006, Voronezh, Russia
[3]Moscow State University of Civil Engineering, 129337, Yaroslavskoe shosse, 26, Moscow, Russia

Abstract. The article discusses the design stages of very large-scale integrated circuits (VLSI) and the features of the procedure for verifying complex-functional VLSI blocks. The main approaches to microcircuit verification procedures are analyzed to minimize the duration of verification cycles. In practice, a combination of several approaches to verification is usually used.

1 Introduction

The modern level of semiconductor technology (design standards of 0.13 μm, mastered in mass production, and 0.09 μm in pre-series prototypes) and further reduction in the size of components provide the possibility of placing on a chip all the main functional devices required by the customer. One of the main classes of modern and promising ultra-VLSIs are digital signal processing chips (DSP), which in turn are one of the main classes of "systems on a chip" that received the well-established abbreviation SoC (System-on-a-Chip) [1-3]. In addition, DSP can be implemented as a complex-functional block (SF-block) of a more complex SoC. Generally speaking, SoC devices should not be classified as a certain marketing segment or a special sector in terms of specific areas of application - in fact, this is a way to implement modern multifunctional VLSI. SoC devices can have very different structure, configuration or functionality, however, it can be said that in a typical case, DSP and SoC in general contain the following main fragments or components:

- processor or processor subsystem, incl. DSP, RISC core, etc.,
- storage devices of various types,
- processor bus
- peripheral bus
- a bridge between two tires,
- data conversion devices (including analog-to-digital converters, as implemented in DSP),
- controllers of peripheral devices.

* Corresponding author: wkz@rambler.ru

– interface circuits (ports).

In many respects, the verification of a DSP and the SoC as a whole is similar to the verification of specialized VLSIs of the ASIC class (Application-Specific Integrated Circuits) with a more homogeneous structure (Fig. 1): a large amount of modeling work is carried out, the compliance of the simulation result with the expectations fixed by technical specifications is checked, and finally , the simulation steps are repeated for various variants of initial conditions and initial data (scenarios). Therefore, the approach to verifying a DSP as an IP block is in many ways similar to verifying the entire, more complex "system-on-a-chip".

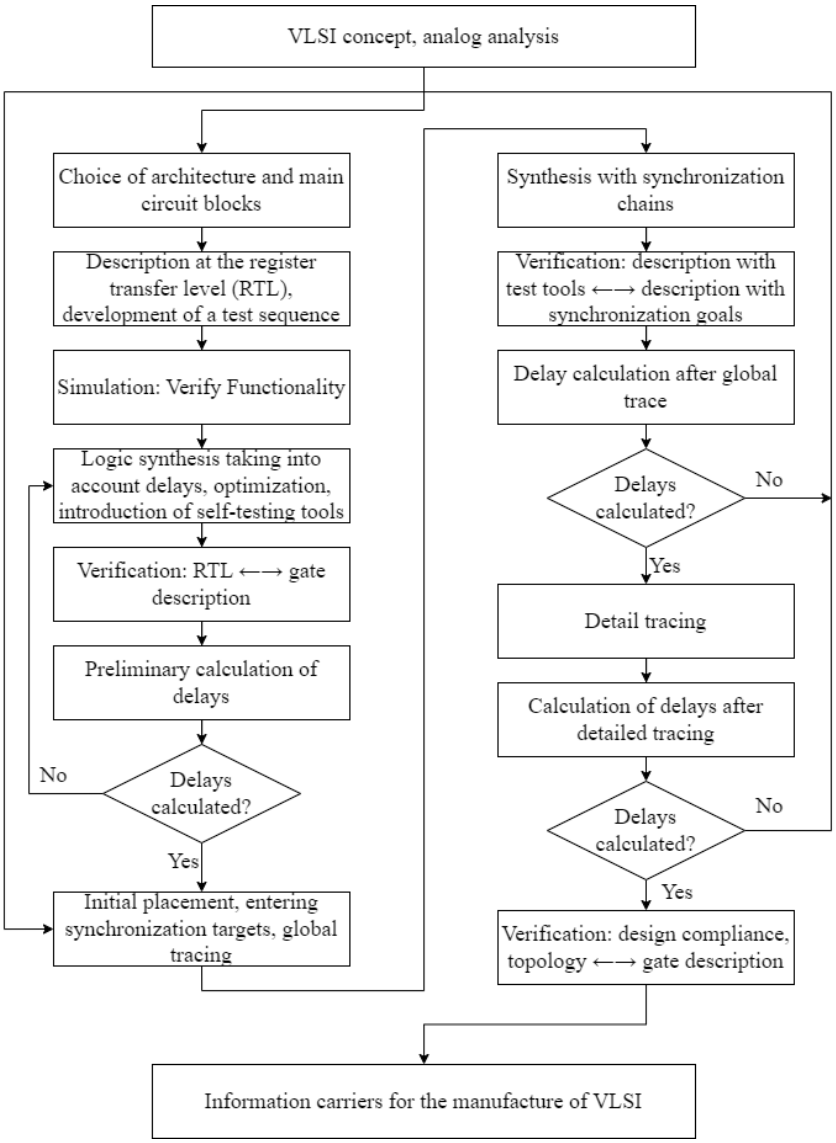

Fig. 1. VLSI design stages and verification procedures.

2 Model and method

Unlike very large-scale integrated circuits (VLSI) - digital or digital-analog - (in a generalized form, this process is shown in Fig. 2), SoC verification has certain features and sometimes is a rather complicated task [4-6]. These features include:

– *integration* - the main focus in the verification of SoC is to check the interconnection between the individual blocks of the system. It is assumed that each block (macrocomponent) is checked by itself. Sometimes such a check of the interconnection of blocks requires the development of specific methods;

– *complexity* – VLSI of the SoC type can have a limiting degree of integration for a given level of technology due to a large number of large constituent blocks themselves. Each of these blocks may require a special approach when organizing verification. As a result, it is necessary to develop combined testing methods, as well as to measure the degree of completeness of such testing procedures, taking into account the most unfavorable combinations of initial conditions;

– *the use of ready-made IP-blocks* - the use of borrowed IP-solutions in the form of certified libraries of macroelements, the parameters of which are determined based on the application of a particular technological process, also implies the existence of methods and methods for their verification. Sometimes even the possibility of verifying an IP block is considered as a greater value than the creation of this block “in silicon”. As a rule, the design of IP blocks at the subsystem level is carried out by various groups of developers, and therefore the presence of a certified verification technique in the library of IP blocks used is especially valuable;

– *joint verification of hardware and software* - general application or proprietary software installed on the processor can be verified in relation to this particular hardware solution. Moreover, existing verification methods imply the presence of an integrated hardware and software device that is subject to testing procedures (DUT - Device Under Test), and verification scenarios include various combinations of both hardware and software. Thus, it is possible to trace their mutual dependence in the generated tests and in the results of the measurements;

– *unique errors (bugs)* - there are typical areas where errors often occur when designing SoCs:

– methods of interaction of blocks, each of which was recognized as verified;

– conflicts when accessing shared resources;

a) arbitration problems and effects of "latching" devices;

b) priority conflicts;

c) off-design sequences of execution of commands by various devices.

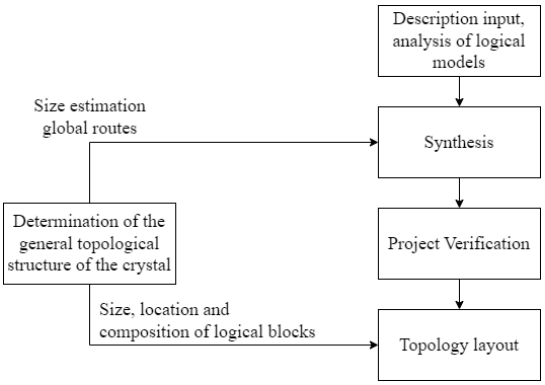

Fig. 2. Verification of the VLSI design at the initial stages of chip development.

All these features indicate the need for the most thorough verification of each SoC macrocomponent, on the one hand, and the need to develop a clear methodology and verification toolkit for the entire system as a whole, on the other. The requirement for completeness of testing leads to the need to implement the highest degree of automation while ensuring SoC verification - otherwise, the solution to the problem of conducting a full-fledged verification loses all meaning.

Currently, hundreds of companies in various countries are engaged in the development of "systems on a crystal". At the same time, there is no single, standard approach to SoC verification [7-9]. However, there are certain quirks that are common to most SoC development projects. Let's consider the main points.

Testing strategy: many developers use the same approaches to SoC verification as when testing specialized ASIC class VLSIs. Such approaches involve the development of a detailed test plan with hundreds of specialized functional tests, while describing all possible "scenarios" and operating conditions of the chip. While these types of test plans are certainly useful and important, their effectiveness is limited by two factors:

– the complexity of SoC is such that the implementation of some critical scenarios is almost impossible,

– as the complexity of systems grows, the preparation of highly specialized "directed" tests becomes more and more problematic, including from the point of view of the implementation of the tasks for which such tests are written.

Test generators: each highly specialized test is associated with only one specific scenario (initial data and external conditions). In fact, the task is to enumerate all possible scenarios in all admissible combinations. This, in turn, leads to the need to prepare tests with a wider scope, which, when launched, would cover entire areas of interest to developers. Sometimes random tests are used to achieve this goal, but they are usually used only at the end of the verification cycle. Although such tests based on random samples have fairly wide coverage areas, they usually have many errors. The problem here lies in the “blind” extension of such tests to all admissible areas, including those that are quite obvious from the point of view of the system’s functioning. In fact, such tests are of practical interest, which, on the one hand, would be quite general, on the other hand, they would provide a flexible opportunity for the developer to conduct intensive testing in those narrow areas that are the least studied and thus require the most intensive testing, and it is highly desirable to provide maximum flexibility in choosing these specific areas (Fig. 3).

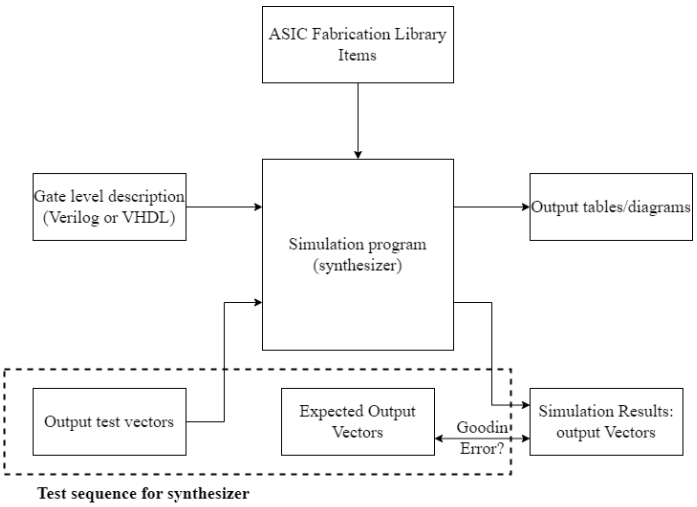

Fig. 3. The structure of post-synthesis logic modeling as a stage of project verification.

Integration control: many approaches to SoC verification do not have special controls for the correct integration of blocks into the system. Instead, the system is tested as if it were some kind of monolithic formation, and it is assumed that any inoperability in one or another block will inevitably be displayed in external measured parameters (for example, there will be a failure at the output of the switch through which data packet goes through). The main disadvantage of this approach is that tracing the source of the fault through the reverse chain can take too long. To avoid this, specialized tools (monitors) are needed that would track problems with the integration of the constituent SoC blocks immediately, starting from their source.

Joint verification of hardware and software: There are several commercial and in-house solutions that provide joint verification. By running real software on emulated hardware, both software and hardware can be debugged before the project goes into production. However, these types of tools fail to represent the combination of software and hardware as a single unit of test (DUT). These techniques may include the management of input test vectors, modification of program tables or variables, but they, as a rule, cannot provide for scenarios in which mutual dependencies of hardware and software are traced. For example, it is impossible to describe a scenario in which a certain signal is applied to the hardware input while the software is in one or another predefined state or in the hardware interrupt service mode. Therefore, an effective SoC verification system should include the ability to capture the critical interdependencies of the hardware and software parts of the design and include appropriate tests in the overall chip verification plan, including the development of control methods and assessment of the completeness of coverage for such procedures.

When is a project considered certified? Since the quality of verification is difficult to measure and express by some numerical parameter, there may not be an absolutely clear criterion for the complete certification of the project. However, there are such parameters as the level of code coverage, the intensity of software errors, the coverage of switching nodes and the completeness of tests - all of them are far from universal and often do not fit at all into those combined scenarios, implementation which is absolutely necessary for the full verification of the SoC. Thus, the task arises of developing a methodology that allows a more objective and complete assessment of the quality of project verification.

Summarizing the above, it can be noted that if when testing "ordinary" VLSI there is inevitably an element of uncertainty arising from a non-100% level of test coverage in absolutely all possible scenarios, then in cases with SoC, where a large number of different - native elements, this uncertainty is much higher. That is why new approaches and techniques are being developed that will be able to introduce the necessary level of determinism and minimize the time required for adequate verification of the SoC project.

When trying to build a SoC verification model, it is necessary to answer a number of questions: what does it mean to verify the system as a whole, and how does this differ from verification of individual blocks that make up the system? Should the hardware, software, or both be verified? How should re-existing block verification models be used in system testing?

In fact, the answers to these questions are the specifics of SoC verification. So, for example, you should take advantage of the fact that the SoC consists of IP (IP blocks) and pre-verified blocks in each IP part. It should also not be overlooked that in fact the SoC consists of two objects under test (DUT): the system crystal itself on the one hand and the “chip + software” complex on the other. Verification, as a rule, begins with formal documents: descriptions of the project and test plan. Therefore, the practical task is to create a full-fledged verification system on the basis of these documents, and to do this in the shortest possible time, spending the least resources. This implies the features of such verification systems [10-12]:

– all technical specifications and functional features of the SoC must be described in an executable format;

– all types of verification procedures should be automated;

– during verification, software and hardware tools that are certified in advance and suitable for subsequent use should be used.

Verification of the design, performed at the initial stages of development (before the detailed design of the topology), allows you to check (1) the correct functioning of the system circuits and (2) compliance with the specified conditions and restrictions on performance, testability, power consumption and, in part, topological connections and electrical modes.

Projects go through functional modeling before the logical synthesis procedure. However, despite the fact that modern synthesis tools are debugged and highly reliable programs, re-simulation after logical synthesis is a mandatory procedure. The traditional way is modeling at the level of the gate description of the circuit. In this case, verification consists in comparing the results of the repeated "post-synthesis" simulation with the results of the "pre-synthesis" calculation, while, naturally, the same test vectors are set at the input of the circuit. With a circuit complexity of more than 100 thousand equivalent vectors, the time required to model the synthesized circuit at the gate level becomes very significant. Projects with a complexity level of 1 million equivalent gates or more can require weeks of machine time to complete logic simulations.

Due to the inefficiency of using such direct methods for complex circuits, the so-called. formal verification, also known as Boolean equivalence verification.

3 Research and results

The purpose of verification is to ensure full compliance of the project with the functional and parametric requirements fixed in the terms of reference for the project. Due to the high level of complexity of modern SoCs, the verification procedures for SF blocks can take, according to foreign experts, from 40 to 70 percent of the total time needed for chip development. In this regard, questions that require practical answers arise: how much verification procedures should be recognized as sufficient, what approaches and techniques to apply, how to plan and minimize the duration of verification cycles?

Clearly structured and firmly established approaches to the verification of SF blocks and SoCs in general do not exist today. However, four different approaches can be distinguished:

– based on calculation models;

– static;

– formalized;

– physical-analytical.

The features of each of these approaches are described below. In practice, a combination of several approaches to verification is usually used.

3.1 Calculation Verification Procedures

The computational or simulation (based on modeling) approach includes event- or clock-cycle-defined simulators (simulation programs), procedural (transactional) verification, code coverage completeness analysis, mixed signal calculations, joint verification -fiction of the software and hardware of the project and various accelerators: emulators, rapidly created prototype systems, hardware models and hardware accelerators.

Event-based simulators (simulation programs) operate on one-time events (any changes in the input test vector are considered to be such), which propagate through all circuits of the device until a steady state is reached. At the same time, the models used include functional and temporal aspects. The state of the same project element can be characterized several times in one cycle due to different delays in signal propagation through different inputs, as

well as due to the possible influence of feedback signals coming from devices connected to the output of this circuit. In general, this approach provides a fairly accurate assessment of the state of the computing environment and makes it relatively easy to detect design errors. However, the speed of implementation of this type of verification depends on the size of the device under test and the intensity of input actions. For large projects, this approach is implemented quite slowly, because. complex algorithms are used for scheduling events and the evaluation of output states is performed repeatedly.

Cycle-based simulators do not take into account the time factor within a cycle. They evaluate the logical states at the terminals of the circuit elements in an "instantaneous" mode. Since each logical element is evaluated only once per cycle, the verification speed increases, although various failures associated with the temporal structure of the signals may occur. In this case, additional verification is required using the means of "static" temporal analysis. Loop simulators only work with synchronous logic. They provide verification speed 5-100 times faster than event simulators. For large circuits, the calculation speed can be up to 1000 cycles per second. This approach is best used for devices that require the use of large test vectors: microprocessors, specialized LSIs such as ASICs, SoCs.

Transaction-based verification allows emulation and debugging of the design at the level of internal procedures in addition to the level of signals on external pins. All possible types of interaction (transactions) between the blocks of the system are considered and periodically tested. Transactional verification does not require the development of detailed test sequences with large vectors. This type of verification uses the so-called bus function model (BFM - bus function model). BFM provides signal transmission to the interfaces of the tested device in accordance with the specified data exchange protocols. BFMs are fairly easy to define using hardware description languages or C++. This approach makes it possible to facilitate the implementation of device self-control procedures and the generation of directed pseudo-random tests.

Code coverage analysis provides the possibility of quantifying the functional coverage of a particular test that is assigned to the inputs of the device under consideration. It can be either a separate VLSI unit or an SoC chip as a whole. Such an analysis makes it possible to obtain an assessment of the completeness of tests for each parameter of interest to developers, as well as to reveal parametric areas that are not covered or insufficiently covered by tests. Code coverage completeness analysis is performed at the level of register transfers of the device, while various types of coverage are evaluated: states, switches, paths, signals, branches, etc.

In the course of joint hardware-software verification, software and hardware are integrated and verified simultaneously. The collaborative verification environment is a graphical user interface (GUI) that is compatible with hardware simulators and software emulators/debuggers used by developers and software, and the chip itself. This gives software developers the ability to run programs in a real hardware environment. In addition, the chip designers receive input test vectors directly, which simplifies the development of test sequences. This verification method provides high performance when generating correctness tests for interfaces, code segments, specialized drivers and utilities. A limitation of hardware-software verification is the often encountered low performance when launching user applications in real-time operating systems due to lack of system resources and insufficient computational speed.

Emulators are custom built hardware/software systems that typically contain reconfigurable logic, often in the form of user-programmable logic arrays (FPGAs). Some emulator systems common among SoC developers contain high-speed matrix processors. These systems are programmed in such a way as to reproduce (emulate) the behavior of the system being developed, while the functionality of the system is reproduced to such an extent that the emulator system can be inserted into a real working environment in which the

developed VLSI is expected to function. Since such systems are created on the basis of specialized hardware, their performance is orders of magnitude higher than that of purely software emulators, and in some cases is comparable to the performance of the devices being developed.

Rapid prototyping systems allow you to model the device being created based on the available hardware blocks and nodes. As an element base for such systems, as a rule, microprocessors, microcontrollers, digital signal processing chips (DSP), programmable logic arrays (FPGA) are used. Typically, these elements are assembled on a daughter board, which, in turn, is inserted into the mother backplane, which also contains programmable devices that reproduce the signal interfaces of the simulated device. As a rule, prototyping is used in the development of systems based on embedded processors, while additional components are used: memory chips, FPGA, additional processor cores ("cores"). This approach allows for effective debugging of software running on real hardware-analogue of the one being developed. This, in turn, facilitates the integration of the hardware and software of the system being developed at the stage when prototypes of the SoC crystal are obtained.

Hardware accelerators are specially designed devices that allow you to replace the software emulation of certain system components. In most cases, test sequences are launched in their original software form, while the project is verified using hardware accelerators. In some cases, the input vectors of test sequences are also "run" through the hardware accelerator.

Mixed signal modeling itself is more complex than pure digital signal paths or analog systems. It must be taken into account that in most cases analog-to-digital devices have digital sub-systems at the upper level of signal processing and analog blocks at the lower level. Due to the greater complexity of the analog signal, the analog modeling tools available in the industry are not as automated as they are in the digital signal domain. Therefore, the following sequence of actions is usually used: mixed analog-digital blocks are verified separately, the interface part of the mixed block is considered as a digital device, and the interface itself is verified after integration into the SoC.

3.2 Static Verification Technologies

Static verification technologies include software syntax checking and static timing analysis. A feature of such technologies is the absence of the need to form test sequences or test vectors.

Syntax checking (lint checking) provides a procedure for monitoring and detecting, for example, uninitiated variables, software configurations not supported by this type of software, mismatches in addresses of specified interface ports, etc. Syntax checking is carried out at the early stages of design. This reveals relatively simple errors in the program code, which, on the other hand, would require too much effort if using more powerful software tools and methods.

Static timing analysis allows you to check whether necessary timing constraints are met, such as trigger and storage element set-up and hold times, signal propagation delays, etc. Timing analysis is always critical for large systems, because each input of a functional cell is associated with a large number of circuit elements, in addition, the timing parameters significantly depend on the external conditions in which the microcircuit operates.

3.3 Use of formalized technologie

Formal (or formalized) technologies also do not require the compilation of large test sequences or input test vectors. They are very effective for detecting specific errors that result from certain sequences of events. These kinds of errors are very dangerous because of their

variability and, if they are not eliminated at the initial stages of the design process, they become dangerous as the project develops further, leading in some cases to the complete inoperability of the system. Formal verification technologies provide short procedure execution time and 100% code coverage. There are three types of formal verification technologies. The first one is a method based on the proof of certain theorems, which are formulated based on the behavioral model of the device. This method has not yet left the academic stage of research and is not yet used in practice.

The formal model verification technology uses formalized mathematical algorithms to verify the behavioral properties of the system being verified. Formal model verification software checks whether the behavioral response of the system being designed matches the set of logical conditions specified by the developer. These conditions follow directly from the functional algorithms of the developed system. Formal model checking is useful for controlling complex control devices such as bus arbiters, decoders, peripheral bridges, etc. The properties of the system to be checked are defined in the form of requests, and when the analyzer finds an error (mismatch), it generates a report that traces the entire "history" of the signal from the initial state to the state that does not meet the developer's expectations. This verification technique does not negate the need for calculation procedures, but rather complements it. Since the temporal parameters of the signals are also not taken into account, it is necessary to carry out an additional static temporal verification. Existing formal model-based verification tools are effective for checking relatively small SoCs due to the performance limitations of the computing tools used. In addition, if the behavioral properties of the system are not described quite correctly, existing errors may not be noticed [13-16].

The formal equivalence test method is used to confirm the equivalence of two approaches to the same logical description (project). Mathematical algorithms are used to verify the equivalence of the control and modified designs. Appropriate tools are used to confirm the equivalence of projects in the RTL-RTL bases, RTL-gate description and gate-gate description [17-19]. It is extremely important that the reference description used in the comparison is absolutely correct. This method does not require the development of test sequences and test vectors and also does not take into account the temporal aspects of signal propagation, which requires additional temporal analysis [20].

4 Conclusion

Physical verification and analysis become absolutely necessary when developing devices with submicron design standards, which include almost all DSP processors developed in the form of IP blocks. The physical effects in the "deep submicron" (DSM) region include, first of all, the influence of parasitic RC-parameters of interconnects, which become dominant in determining the propagation delays, at least, of interconnect signals. Issues requiring separate consideration are delays, signal stability, crosstalk, ohmic bus voltage drops, electromigration, local heating of heat-critical areas of the crystal, the so-called "antenna" effects, phase shifts in lithography and optical proximity correction (both the latter effects are of a diffractive nature and are very critical in the deep submicron region). There are a large number of techniques that allow you to evaluate the influence of all these factors, both before the development of the chip topology, and after the placement and routing of library elements.

References

1. S. Belokurov, V. Belokurov, V. Zolnikov, O. Cherkasov, Transportation Research Procedia **20**, 47–52 (2017)
2. V. Belokurov, S. Belokurov, V. Zolnikov, Transportation Research Procedia **36**, 44–49 (2018)
3. T.E. Smolentseva, V.I. Sumin, V.K. Zolnikov, V.V. Lavlinsky, Journal of Physics: Conference Series **973(1)**, 012045 (2018)
4. C. Zheng, D. Cao, C. Hu, Frontiers of Optoelectronics **15(1)** (2022) doi:10.1007/s12200-022-00004-9
5. W. Shi, Z. Huang, H. Huang, et al, Light: Science and Applications **11(1)** (2022) doi:10.1038/s41377-022-00809-5
6. M. Rusanovsky, O. Beeri, G. Oren, Scientific Reports **12(1)** (2022) doi:10.1038/s41598-022-08651-w
7. Y. Bityukov, Y. Deniskin, G. Deniskina, I. Pocebneva, E3S Web of Conferences **244** (2021) doi:10.1051/e3sconf/202124405004
8. R. Ojala, J. Vepsäläinen, K. Tammi, Journal of Big Data **9(1)** (2022) doi:10.1186/s40537-022-00581-8
9. M.A. Al-Malla, A. Jafar, N. Ghneim, Journal of Big Data **9(1)** (2022) doi:10.1186/s40537-022-00571-w
10. W. Yang, W. Liow, S. Chen et al, Eurasip Journal on Advances in Signal Processing **2022(1)** (2022) doi:10.1186/s13634-022-00839-6
11. S. Hao, X. Han, Y. Guo, M. Wang, Communications and Applications **18(4)** (2022) doi:10.1145/3498341
12. M. Lupión, A. Polo-Rodríguez, J. Medina-Quero et al, Expert Systems with Applications **203** (2022) doi:10. 1016/j.eswa.2022.117356
13. H. Zhang, S. Zhang, Y. Zhang et al, Robotics and Computer-Integrated Manufacturing **77** (2022) doi:10.1016/j.rcim.2022.102369
14. A.R. Deniskina, I.V. Pocebneva, A.V. Smolyaninov, Proceedings - 2021 International Russian Automation Conference, RusAutoCon 2021, 17-22 (2021) doi:10.1109/RusAutoCon52004.2021.9537333
15. A. Smolyaninov, I. Pocebneva, I. Fateeva, K. Singur, E3S Web of Conferences **244** (2021) doi:10.1051/e3sconf/202124411009
16. A. Amer Mohammed Salih, M. Al-Khannaq, K. Hasikin, N. Ashidi Mat Isa, Alexandria Engineering Journal **61(12)**, 11185-11195 (2022) doi:10.1016/j.aej.2022.04.023
17. A. Smolyaninov, I. Pocebneva, I. Fateeva, K. Singur, E3S Web of Conferences **244** (2021) doi:10.1051/e3sconf/202124411009
18. B. Li, L. Ye, J. Liang, Y. Wang, J. Han, Neurocomputing **500**, 13-25 (2022) doi:10.1016/j.neucom.2022.05.047
19. B. Li, L. Ye, J. Liang, Y. Wang, J. Han, Neurocomputing **500**, 13-25 (2022) doi:10.1016/j.neucom.2022.05.047
20. M. Romanovich, M. Kuzmenkova, V. Breskich, K. Kulakov. Transportation Research Procedia, Volume 54, 2021, Pages 819-826. https://doi.org/10.1016/j.trpro.2021.02.135
21. S. Cha, Y. Wang, Pattern Recognition Letters **158**, 87-93 (2022) doi:10.1016/j.patrec.2022.04.011