

MINISTRY OF EDUCATION AND SCIENCE OF UKRAINE
NATIONAL TECHNICAL UNIVERSITY
«KHARKIV POLYTECHNIC INSTITUTE»

S. Leonov, H. Heiko

**COMPUTER-AIDED DESIGN TECHNOLOGIES
FOR COMPUTER SYSTEMS AND COMPONENTS**

Training manual for computer engineering students

Approved
by editorial and publishing
department of the NTU «KhPI»,
protocol № 1 dated 19.02.2026

Kharkiv
NTU «KhPI»
2026

Reviewers:

Andriy Kovalenko, Doctor of Technical Sciences, Professor, Head of the Department of Electronic Computers, Kharkiv National University of Radio Electronics

Maryna Miroshnyk, Doctor of Technical Sciences, Professor, Professor of the Department of Computer Systems and Robotics, V. N. Karazin Kharkiv National University

Leonov S.

L47 Computer-Aided Design technologies for computer systems and components: a training manual for computer engineering students / S. Leonov, H. Heiko. – Kharkiv : NTU «KhPI», 2026. – 243 p.

A training manual includes theoretical and practical material related to the provision of computer-aided design systems. The basic information on the composition and principles of operation of computer-aided design systems is provided. The full cycle of designing complex hierarchical devices is described – from functional modeling and verification to design and obtaining printed circuit boards.

For students of the specialty «Computer Engineering» and other technical specialties.

Fig. 273. Tables 32. Bibliography 9.

INTRODUCTION

Computer-aided design systems facilitate the creation, modification, analysis, and optimization of designs using computer systems. These systems are divided into computer-aided design (CAD), computer-aided manufacturing (CAM) and computer-aided engineering (CAE) systems.

The most important function of automated design is to determine the geometry of the structure and model functional and electrical diagrams. For this purpose, systems for developing working drawings, modeling and functional or topological design are usually used.

Computer-aided engineering is a technology that involves the use of computer systems to analyze CAD geometry, model, refine, and optimize designs.

OrCAD is a system that allows developers to model electronic circuits and design printed circuit boards. The software package consists of OrCAD Capture (a graphical editor for electrical schematic diagrams); OrCAD Capture CIS (a schematic editor with the ability to maintain component databases); OrCAD Layout Plus (a printed circuit board editor); OrCAD PSpice A/D (used for mixed modeling of analog and digital devices).

The textbook covers the methodological foundations of CAD and the analysis of the performance of designed devices. It examines the practical use of OrCAD in the development of complex hierarchical computing devices using various top-down and bottom-up design methods, analysis of their performance based on modeling, as well as topological design of these devices.

CHAPTER 1

BASICS OF CAD CONSTRUCTION

1.1 Methodology for building design systems

The design of a technical object is associated with the creation, transformation, and presentation of an image of this object in an accepted form. Design begins with the compilation of a design task, which is presented in the form of technical or other documents and is the initial description of the object. The result of design is a complete set of documentation containing sufficient information for the manufacture of the object. Such documentation is the final description of the object. The process of transforming the initial description into the final description of the object is a set of intermediate descriptions or design solutions.

Automated design is a process or set of measures aimed at implementing design solutions using computers. In this case, a rational distribution of functions between a person (designer) and computers must be provided. Non-automated design is design without the use of computers. Automatic design is a type of design in which the design is performed using computers without human involvement, the designer is completely "excluded" from the design process and is only involved in the decision-making process. The goal of design automation is: improving quality; reduction of material costs; reduction of design resources; reducing the number of designers; increasing the productivity of designers. CAD is a set of automated design tools interconnected with the departments of a design organization. Close interaction between humans and computers in the design process is one of the principles of CAD construction and operation. There are two types of design: external and internal. External design includes preliminary design studies, development, and approval of technical specifications. Internal design includes the stages of developing a technical proposal, preliminary design, technical design, working design, manufacturing, debugging, testing, and commissioning. Pre-project research involves conducting a survey of the project organization, preparing a technical report, and obtaining approval.

The survey includes the following activities: assessment of the possibility of creating a CAD system; data collection, description, and analysis of CAD; collection of proposals for the creation of CAD; composition of CAD

subsystems and components; formulation of technical requirements for CAD functions and structure; types of support and principles for CAD creation.

CAD systems are classified according to various criteria. In the field of application, CAD systems for mechanical engineering (MCAD), CAD systems for radio electronics (ECAD, EDA), and CAD systems in the field of architecture and construction (ArCAD) are widely used. Depending on their purpose, CAD systems are divided into those that implement different aspects of design. The need to design complex systems has led to the active use of a systematic approach in engineering, which is a set of methodological principles and provisions that allow the system to be viewed as a single whole. This approach involves studying each element of the system in its interaction with other elements and takes into account the following features: the design process must have a hierarchical structure; the design process must take into account the analysis of the relationships that exist in the object and in the control part of the system; also, when designing, one should provide for the continuous development of the system. In the design process, the tasks of structural and parametric synthesis are distinguished. Artificial intelligence methods are most often used in synthesis tasks. A feature of such tasks is the need to make multiple decisions during their implementation. The main types of synthesis are: optimization (the developer creates the initial version of the designed object, and then, if the characteristics of the object differ from the required ones, the parameters or structure of this object are changed); generation (consists in creating a model of the object).

There are many classifications of synthesis tasks. For example, stages, levels, phases, characteristics of mathematical models, methods of solution, can be selected as classification criteria. In Fig. 1.1 shows one such classification.

To synthesize an object, it is first necessary to determine its structure, element parameters, and design. The following stages are distinguished in the design process:

- at the research stage, the need for new products with a specified purpose is studied, the principles of product construction and the possibilities for implementing these principles are investigated, and the values of the characteristics and parameters of objects are predicted (the result is the formulation of technical specifications for the development of the object);

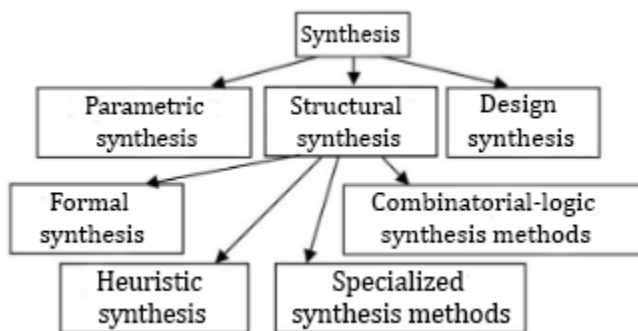


Fig. 1.1 – Classification of synthesis tasks

- at the stage of experimental design work, a preliminary design of the product is created (during its development, the principles and provisions established at the stage of research work are checked, specified, and corrected);

- at the technical design stage, more detailed graphic and text documentation is developed, which gives a complete picture of the device (the technical design includes all the necessary calculations);

- at the working design stage, a complete set of design and technological documentation is created, sufficient for the manufacture of the object.

- at the testing stage, results are obtained that allow the identification of possible errors and shortcomings in the design, and measures are taken to eliminate them.

At any stage of the design process, errors or previously accepted suboptimal decisions may be identified. In particular, it may be necessary to adjust the technical specifications.

From a scientific and technical point of view, the problem of creating CAD boils down to solving the following tasks:

- developing a methodology for designing electronic products;
- development of applied mathematical software;
- creation of CAD technical means;
- creation of organizational and technical support.

The methodology for creating CAD can be broadly formulated as follows: training personnel; organizing the CAD development process; implementing a

distributed control system; using artificial intelligence. Most modern CAD systems are based on exhaustive internal logic, which requires the user to explicitly describe all the actions that determine the decision-making process. This organization of the design process has disadvantages in that CAD systems are not evolutionary and their scope of application is limited to solving typical problems. At the same time, the introduction of artificial intelligence (AI) methods allows these disadvantages to be eliminated.

Conventionally, the following levels of CAD intellectualization can be distinguished:

- 1) a level at which AI tools are used only as components of individual CAD subsystems;
- 2) a level that involves the presence of CAD subsystems that are fully organized in accordance with AI methodology;
- 3) a level achieved when CAD is built entirely on the organizational principles of AI systems.

The main object of study in AI is knowledge. This concept is not formally defined and has the following characteristics: interpretability, structure, connectivity, and activity. Knowledge about objects and phenomena is divided into syntactic (describing the structure of objects and phenomena), semantic (information related to their meaning and sense), and pragmatic (describing objects and phenomena from the point of view of the task being solved).

In the process of intellectualized CAD, the following tasks are solved:

- technical specifications are drawn up and analyzed;
- a design and layout scheme is selected and the cost of the project is analyzed;
- structural synthesis and optimization are performed;
- search design is carried out and the project is planned;
- redesign and refinement of the design;
- the efficiency and quality of engineering analysis is improved, and simulation modeling is performed;
- compliance with industry standards is verified;
- documentation and working drawings are prepared.

Experience in designing large integrated circuits (ICs) allows us to formulate requirements for CAD systems for such circuits:

- structure invariance to changes in ICs design tools, methods, and

manufacturing technology;

- compatibility of manual, automated, and automatic design;
- variety of modes of interaction between the designer and CAD;
- productivity;
- adaptability;
- simultaneous design of several products;
- independence of commissioning and operation of individual

subsystems;

- quick system configuration for the selected design technology route;
- information consistency of subsystems and application software packages for different stages of large ICs design;
- variety of forms of presentation of source documentation;
- high reliability and reasonable cost.

These requirements will be met if the following principles are used when creating CAD for ICs:

- the principle of comprehensiveness (the ability to solve problems throughout the entire product design cycle);
- compatibility of automatic, automated, and traditional design;
- availability of a common database;
- modularity (this will allow the implementation of individual subsystems in the form of functionally independent modules with their subsequent replacement by other modules);
- multi-accessibility (CAD large ICs should be a system for collective use);
- openness (adding new subsystems or individual programs does not require changing the methods of organizing management, description, and processing of information).

When designing a CAD system, the principle of decomposition is used first and foremost, which allows a large-scale task to be reduced to several smaller-scale tasks. The decomposition of the designed object leads to a hierarchical design principle, and the more complex the designed object, the more levels of hierarchy there are. During design, it is necessary to ensure that the correctness of design decisions is verified at the design stage. The essence of the iterative design principle is a sequential approach to meeting the specified requirements based on the results of modeling and optimization at

each stage. It should be noted that the controllability of each stage is an important design principle. When placing elements and tracing inside circuit connections on an ICs chip or printed circuit board, control is applied in conjunction with the design process. Control of the correctness of design work at various stages of ICs is called verification.

When manufacturing ICs, the possible appearance of defects caused by design errors should also be taken into account, which is why all errors should be eliminated before the prototype manufacturing stage using a topology verification system. It should also be noted that when designing large ICs, when the topology design process is not complete (the tracing of some interconnections is not finished), the topology can be corrected using manual methods, during which errors are also detected.

The types of checks required at the final stage of design are divided into three groups:

1) verification of geometric parameters, detection of deviations from design rules;

2) checking interconnections, identifying defects and errors in the manufacture of circuit elements;

3) verification of electrical characteristics.

The result of CAD large ICs work should be the release of project documentation, which includes:

- text and text-graphic documents (description of the functioning algorithm, functional-logical diagram, timing diagrams, schematic diagrams of functional blocks, electrical characteristics of functional blocks and components, tables of component parameters);

- graphic documents (drawings of the topology of ICs functional blocks);

- documents on machine media (control tests for functional control installations, information on machine media for software-controlled large ICs technological equipment).

CAD software is divided into system-wide and application-specific software. System-wide software is a set of system programs that organizes the execution of application programs on CAD hardware. Application-specific software is characterized by a set of application program packages (APP) for all stages of ICs design and instructions for their use. CAD large ICs application software should consist of a set of APP, which should be specific

for each design stage, and within each of these sets, uniform rules for describing the initial data should be used. APP must include programs that perform: structural synthesis of mathematical model compilation; mathematical model solutions; optimization; statistical analysis. Methodological support includes a set of instructions and other documents regulating the use of automated design tools.

Organizational support is a list of documents that regulate the interaction and functions of users involved in the automated design process, as well as the material and technical support of CAD large ICs.

Technical support is a set of computers and peripheral devices for entering, storing, processing information, and transferring programs and data. These technical means must ensure interactive operation, the execution of all necessary design procedures, and interaction between all developers working on a joint project. The ICs design route is understood as a sequence of design procedures that must be performed when developing a specific circuit. Currently, due to the variety of ICs manufacturing processes and design methods, there are many design routes. It should be noted that nowadays, the preparation of product manufacturing focuses on the design of technological processes, which contain information about labor and material standards, without which it is impossible to plan and manage production resources. If there are means of automating the preparation of technical documentation and means of information support for design, it is possible to achieve great success in saving the labor of technologists, which is the main goal of creating CAD for technological processes.

1.2 Block-hierarchical approach to design

Practice shows that the vast majority of complex systems, both in nature and in technology, have a hierarchical internal structure. Each subsystem can be divided into subsystems down to the lowest "elementary" level. At the same time, the choice of the level whose components should be considered elementary remains with the researcher. At the elementary level, the system usually consists of a few types of subsystems, combined and organized in different ways.

The process of breaking down a complex object into relatively independent parts is called decomposition. When decomposing, it is important

to note that the connections between individual parts should be weaker than the connections between elements within the parts. In addition, in order to be able to assemble an object from the resulting parts, it is necessary to determine all types of connections between the parts during the decomposition process.

When creating very complex objects, the decomposition process is performed repeatedly: each block, in turn, is decomposed into parts until blocks that are relatively easy to develop are obtained. This development method is called step-by-step detailing.

The result of decomposition is usually presented in the form of a hierarchy diagram, at the bottom level of which there are relatively simple blocks, and at the top level there is the object to be developed. At each hierarchical level, the description of the blocks is performed with a certain degree of detail, abstracting from insignificant details.

Each level uses its own forms of documentation and models that reflect the essence of the processes performed by each block. For the object as a whole, it is usually possible to formulate only the most general requirements, and the lower-level blocks must be specified so that they can be assembled into a working object. In other words, the larger the block, the more abstract its description should be. By adhering to this principle, the developer retains the ability to comprehend the project and, therefore, can make the most correct decisions at each stage, which is called local optimization. It should be remembered that the concept of a complex object changes as technology improves, and what was complex yesterday will not necessarily remain complex tomorrow.

Thus, the basis of the block-hierarchical approach is decomposition and hierarchical ordering.

The following principles play an important role:

- consistency (control of the consistency of elements with each other);
- completeness (control for the presence of redundant elements);
- formalization (strictness of the methodological approach);
- repeatability (the need to identify identical blocks to reduce costs and speed up development);
- local optimization (optimization within the hierarchy level).

The set of model languages, task settings, and description methods of a certain hierarchical level is commonly referred to as the design level. Each

object in the design process usually has to be considered from several angles. Different views of the design object are commonly referred to as design aspects. In addition to enabling the creation of complex systems, the use of a block-hierarchical approach also simplifies the verification of the performance of both the system as a whole and individual blocks; it provides the possibility of modernizing systems, for example, replacing unreliable blocks while preserving their interfaces. It should be noted that the use of a block-hierarchical approach to software systems became possible only after the general provisions of the approach were specified and some changes were made to the design process.

When using a block-hierarchical approach to design, the designed system is divided into hierarchical levels. At the top level, the least detailed representation is used, reflecting only the most general features and characteristics of the designed system. At the next levels, the level of detail of the description increases. The task of designing matrix ICs is to transition from a given logic circuit to its physical implementation based on a basic matrix crystal (BMC), where the input data is a description of the logic circuit at the level of library logic elements. In this case, it is necessary to obtain design documentation for the manufacture of a working matrix ICs.

The base crystal has the form of a rectangular multilayer plate of a fixed size. In Fig. 1.2, *a* shows the peripheral and internal areas of the BMC. The peripheral area contains external contact pads (ECP) for external connection and peripheral areas (PA) for the implementation of buffer circuits (Fig. 1.2, *b*).

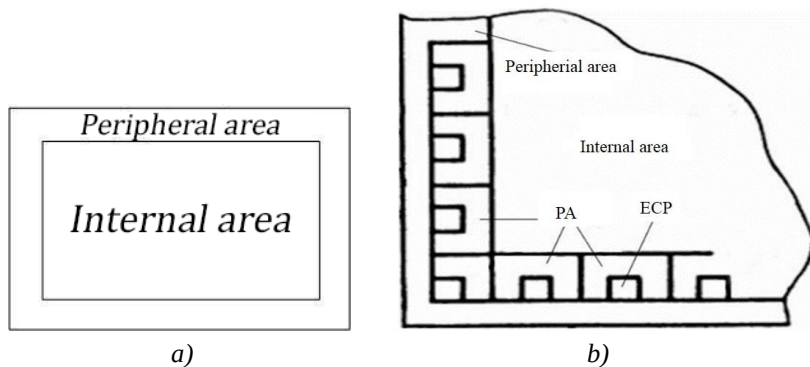


Fig. 1.2 – Main areas of the BMC (a) and structure of the peripheral region (b)

In the internal area of the crystal, macrocells (MC) are arranged in a matrix manner to implement the elements of the projected circuits (Fig. 1.3). The gaps between the macrocells are used for electrical connections. With this arrangement of macrocells, the tracing area is divided into horizontal and vertical channels, and cells for implementing logic elements are located within the macrocell.

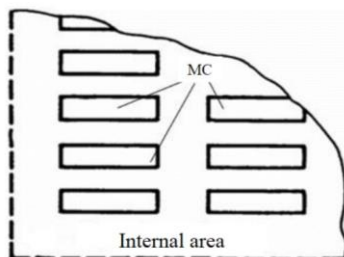


Fig. 1.3 – Internal area structure

In a channel matrix, the cells are arranged in arrays, with free areas between the columns called channels. In the absence of channels, the cells are arranged in a single large array (Fig. 1.4).

By implementing the appropriate interconnections between the cells, simple logic gates can be obtained, and by combining several basic cells, more complex circuits can be obtained.

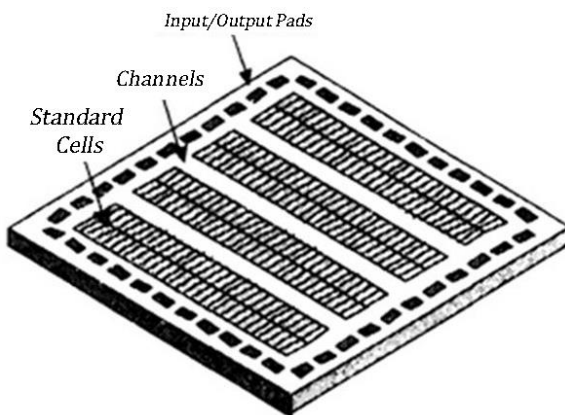


Fig. 1.4 – BMC structure

The process of designing matrix large ICs can be represented in the following stages: modeling the functioning of the object; topology development; checking the design results and making adjustments; release of design documentation. It should be noted that even at the design stage, it is necessary to ensure the correct functioning of the matrix IC, since it is not a configurable and repairable object. There are two ways to do this. One is to create a layout of matrix ICs based on discrete elements, and the other is mathematical modeling. The first method is time-consuming and costly, so the model is used when it is not specifically developed but already exists. The second method requires the creation of an effective circuit modeling system. At the control stage, the adequacy of the resulting design is checked against the initial data. The final stage of design is the release of design documentation. The general approach to solving large ICs design is to break down the initial design task into subtasks. An example is the stage of breaking down the designed circuit, when the number of interconnections is minimized. Then the design object is divided into fragments, and each fragment is designed independently. Thus, we can talk about a hierarchical approach to designing large ICs. The process of solving the given task is replaced by the repeated solution of similar tasks of smaller dimensions.

In block-hierarchical design of large ICs, the following levels of object description are used:

- algorithmic (the main focus is on the behavioral description of the large ICs);
- structural (the large ICs is represented as a set of basic blocks and connections between them);
- command system level (commands and rules for their execution by hardware are defined);
- inter-register transfer level (registers serve as the main elements of the system);
- logic gate level (the large ICs is represented as a set of gates and triggers, as well as connections between them);
- circuit level (the elements of the large ICs are transistors, diodes, resistors, and the description contains information about their connections);
- photomask level (schematic level elements are converted into a detailed topology).

1.3 Technological route of ICs design

With regard to ICs design using CAD, the concept of an ICs design technological route can be introduced, which is understood as any coordinated set of programs with software for different stages, which ensures the design of a given specific circuit throughout the entire cycle. Depending on the specifics of designing specific ICs, the concept of a technological route can also be introduced for a set of programs within a single software package. With software packages for solving tasks at different stages of design, it is possible to form technological routes that ensure high-quality design within acceptable time frames.

For example, the technological route for calculating component parameters is formed on the basis of the following provisions:

- the composition of the software package in a specific technological route is determined by the developer depending on the requirements for the component parameters and the availability of the technological process for its manufacture;
- the complete technological route must include programs to ensure the solution of tasks of modeling, analysis, optimization, and statistical analysis of a given ICs component;
- the total machine time spent on calculating a component largely depends on the scheme of combining programs and the order of their functioning, as well as on the correct choice of the mathematical model of the component and the efficiency of the numerical methods used in the programs.

The variety of technical requirements for ICs, manufacturing processes, and design methods has led to a wide variety of design routes. To implement the project, it is necessary to develop an individual system of transistor interconnections, to implement the large ICs on the base crystal, it is sufficient to design and manufacture only photomasks for variable switching layers. A semi-custom large ICs manufactured in this way is called a matrix large ICs. The increasing complexity of ICs designs precludes the use of manual methods for designing semi-custom ICs. The technological route for designing a matrix large ICs consists of the following stages: development of a basic electrical diagram of the matrix large ICs; development of the topology of variable layers and control of the matrix large ICs design; development of technical documentation (programs for testing the performance of the matrix large ICs

and control programs for manufacturing photomasks). The development of the schematic diagram is carried out using a functional-logical modeling subsystem. The engineer develops a sketch of the schematic diagram, then describes it in a special language. This description is one of the sections of the task for modeling the large ICs on a computer. The text of the task undergoes syntactic and semantic control with the issuance of diagnostic error messages. After correcting the errors in the initial description, the process of modeling the matrix large ICs begins.

The functional-logical modeling stage is completed when, for all input influences, the necessary signals (responses to actions) are obtained at the outputs of the circuit. Thus, the modeled text of the circuit description becomes the reference, and the system of influences and responses becomes the initial information for topological design programs.

The stage of developing variable layers of topology can be implemented in three ways: by manual design followed by machine processing, automated design followed by manual refinement, and automatic synthesis. In manual design, the designer develops a sketch of the matrix large ICs topology in accordance with the circuit drawing. The sketch is then encoded, the information is processed, and transferred to the control stage.

In automated design, the source information is the reference text describing the basic electrical circuit. If, during such design, a number of connections are found to be unresolved, it is necessary to manually change the placement of the elements and repeat the tracing procedure. The technological route of matrix large ICs design is shown in Fig. 1.5. With automatic synthesis, the project topology is fully implemented in automatic mode, eliminating the need for control. Since this process can be implemented with the same information support as logical modeling, there is a wide opportunity to study the performance of the matrix large ICs project, taking into account its topological implementation. To do this, a special extractor program is used to restore the matrix large ICs electrical diagram from the topology, taking into account the interconnections. These structures are converted into delay elements, which determine the initial electrical diagram. Then, under the same influences, the diagram is modeled again. Thus, it is possible to determine the performance of the matrix large ICs project before the manufacturing stage, which is impossible with other methods of topology design.

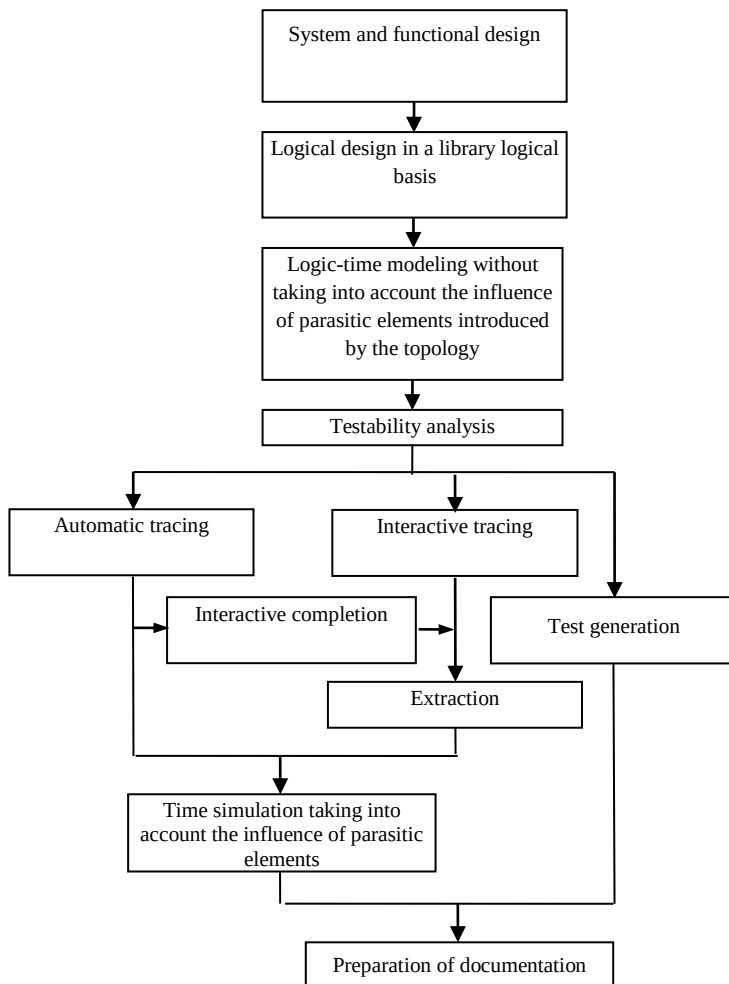


Fig. 1.5 – Technological route of ICs design on BMC

The choice of the technological route for topological design is determined by the number of elements and the number of their outputs. The most common criterion for evaluating the quality of the resulting topology is the criterion of maximum filling density of the base crystal, which is equal to the ratio of the number of matrix large ICs elements to the number of cells (seats) of the crystal. With a filling density of 90...100 %, manual design is used, with a

density of 80...90 % – automated design, and with a density of less than 80 % – automatic synthesis is used. The class of fully custom ICs includes circuits that implement custom designs. Such designs can be single-chip computers, microprocessor ICs, special coprocessors, processors. Such designs contain arbitrary logical structures.

The methodology for designing custom ICs differs from the methodology for designing semi-custom ICs. Semi-custom ICs are most often designed from the top down (top-down design), from a functional description to implementation at the level of library logic elements. Custom ICs are designed from the bottom up (bottom-up design), from the design of logical elements to functional implementation. With this methodology, optimal parameters are achieved at each hierarchical level of the project representation.

The technical indicators of custom ICs are clock frequency, the area of the crystal on which the project is implemented, power dissipation per unit area of the crystal, and the degree of integration. In order for the parameters to be optimal, the design methodology may vary. If there are no strict requirements for clock frequency and chip area, automatic design systems can be used that implement a top-down design methodology, as in the design of semi-custom ICs.

Such design is performed using integrated CAD systems (INCAD). The technical means of such CAD systems are built according to a two- or three-level hierarchical principle: at the first level, supercomputers are used (high performance and large operational and long-term memory); at the second level, medium or mini computers are used, which serve individual terminals, devices, and subscriber consoles; at the third level, mini or micro computers are used, which are the individual workstations of CAD users.

INCAD are the most common CAD systems. INCAD application software contains problem-oriented software modules (POSM), control programs, and databases. Each module has an input block, a solution block and an output block.

Different POSM can be created by different organizations and integrated into the system, while retaining individual input and output protocols. In this case, the user calls the corresponding POSM from the database via the interface and solves the large ICs design problem. In Fig. 1.6 shows a typical route for a custom large ICs in INCAD.

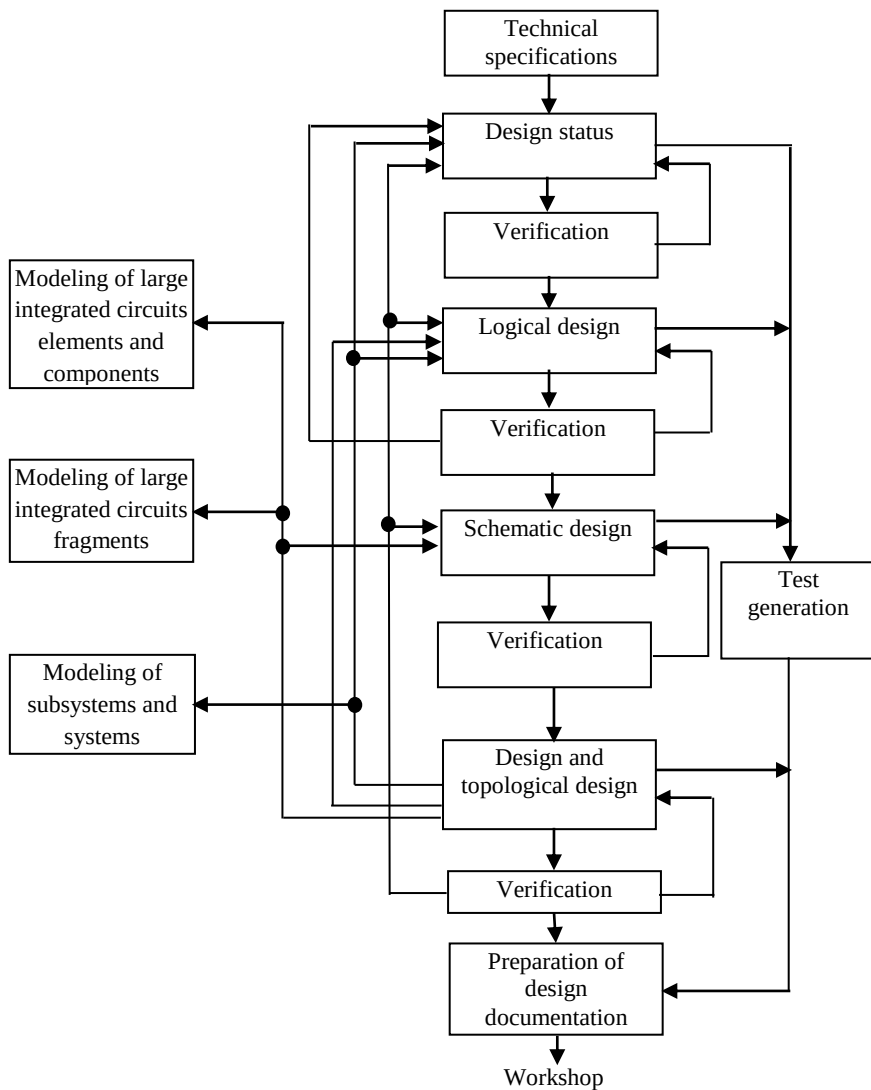


Fig. 1.6 – Typical route of a large ICs project

Individual CAD systems can be combined into a special-purpose CAD system with their own input languages within a given package. Such a

combination plays a huge positive role in the creation of cost-effective and efficient CAD systems for various purposes and structures. In addition, integration proceeds both in terms of the number of problems solved and the number and capabilities of POSM connections. INCAD is necessary for the creation and research of new circuit and technological solutions at enterprises manufacturing ICs for various purposes. They integrate the experience of POSM development specialists over many years. Naturally, such systems require highly developed computing resources.

Specialized microcircuits may contain various blocks, such as custom circuits, standard cell circuits, gate array circuits, and programmable interconnect circuits. The main difference in the design of these blocks is the degree of freedom in choosing the topology. For example, for custom circuits, the freedom of choice is limited only by the rules of topology design. In standard cell blocks (which are pre-designed typical blocks), the freedom of choice is limited by their height and the location of the connections. In gate array blocks, the topology of the elements is strictly defined. In circuits with programmable connections, the topology is completely specified, and during the configuration process, the designer creates the necessary connections. Let's consider the sequence of work in the functional-logical design of large ICs on BMC according to the bottom-up design strategy.

The first stage consists of decomposing the initial logic diagram into separate functional blocks. Elements that can be used in the designed large ICs are selected from the data bank library, and a set of functional blocks and nodes to be developed is selected from them. The second stage is the logical design of functional nodes on large ICs cells. The third stage is the functional design of individual blocks. The fourth stage consists of developing a logical large ICs design. As mentioned earlier, there are two types of design strategies: top-down and bottom-up. The basic principle of design is the one-time development of the topology of each type of component. The process of using the top-down design strategy begins with the consideration of the senior circuit and ends with the obtaining of the topologies of the lowest-level subcircuits. The peculiarity of this design strategy is that it allows for the targeted distribution of crystal design resources and takes into account the location of other subcircuits when designing a subcircuit. However, this strategy can lead to unacceptably rough estimates of the subcircuit topology dimensions.

According to the bottom-up design strategy, the topologies of the lowest-level subcircuits are developed first, which are then used to develop the topologies of higher-level subcircuits in the hierarchy. An important advantage of this strategy is that it allows the dimensions of the topologies of all necessary components to be known at each stage. The advantages of both strategies include the structural continuity of functional nodes, which creates the conditions for using libraries of typical subcircuit topologies and reducing the labor intensity of design, allowing more complex methods of designing individual components to be applied.

The main disadvantage of the strategies considered is the relatively low efficiency of crystal area utilization, since the design does not sufficiently take into account the specifics of the crystal structure and the topology of the subcircuit. Currently, combined methods are used, based on a combination of the principles of both strategies with the inclusion of stages of iterative improvement of solutions. Another disadvantage of these strategies is that their application is justified only in certain cases, since large ICs on BMC are characterized by high element density.

To effectively fill the crystal area, it is advisable to allow subcircuits of the same type to have different topology shapes, and this can be achieved by placing each subcircuit on the crystal in parts. Therefore, even before placement, each subcircuit must be broken down into parts (blocks). The minimum block is a logic element. When developing more complex ICs in conditions of complete uncertainty, when there is no logic diagram, top-down design should be used.

Let's look at the order of solving problems when developing a logic design of an ICs according to the top-down design technology. As a preparatory stage, the architectural design of a digital device is carried out. At this stage, the composition and technical requirements for the ICs included in the device are established.

At the first stage, a generalized circuit diagram of the device is developed, the device is decomposed into individual ICs, and the initial technical requirements for each ICs are established a priori based on the technical characteristics of the device as a whole.

At the second stage, the functioning algorithm and technical requirements for each ICs are refined, each algorithmic model of the ICs is debugged using

appropriate functional tests, and then the debugged models are assembled into a structural diagram of the device. Next, functional tests are developed to verify the correct operation of the device, and joint modeling of the device is performed on ICs models. Based on the modeling results, the connections between the ICs are refined and the technical requirements for each ICs are adjusted. The modeling results are stored in the CAD database. It should be noted that the development and manufacture of various ICs included in the device may occur unevenly – some ICs may be designed and manufactured, while others may be in the design stage. Therefore, it is always possible to return to the architectural design stage and perform joint modeling of the device on ICs models of varying accuracy.

The second stage is the architectural design of the ICs. At this stage, the development and debugging of a simplified ICs diagram is carried out:

- a generalized diagram consisting of functional blocks is compiled;
- algorithmic models and functional tests are compiled to check each of these blocks;
- the algorithmic models of the blocks are debugged, then they are assembled to form a structural model of the ICs.

After the above steps, the results obtained are compared with the results of the algorithmic modeling of the ICs at the previous stage. If these results match, it means that the algorithmic and structural models of the ICs are fully debugged; otherwise, either the algorithmic models of the ICs or the algorithmic models of the functional blocks will have to be refined.

The third stage is the functional design of the large ICs. At the first stage, the functional diagrams of the blocks are developed and debugged (functional diagrams of the blocks are compiled; algorithmic models of the nodes and functional tests are compiled and debugged; a structural model of the block diagram is compiled and verified; the results obtained are compared with the results of algorithmic modeling performed at the previous stage, and if the results do not match, either the structural model of the node or the functional tests are refined). At the second stage, the functional diagram of the large ICs is developed and debugged. The final stage of design is the logical design of the large ICs.

Thus, in the process of developing a logical ICs design using the bottom-up and top-down design methods, the following are created:

- a set of algorithmic models of nodes, blocks, and the ICs as a whole (in the case of top-down design);
- a logical diagram of the large ICs, expanded to the level of BMC cells and presented in the form of a hierarchical fragment-by-fragment description of mutually nested multipoles;
- a hierarchy of functional tests for nodes, blocks, and ICs.

Based on the design developed during the logical and topological design stages, real ICs samples are created. Then, at the functional control stage, the correct operation of the sample is checked. Therefore, it is very important to consider testing on a mathematical model or on a real sample at the early stages of design, in which ICs malfunctions are detected by analyzing the states of its outputs on certain sets of input signals. Successful completion of this task determines its most important characteristics (design defect-free, reliability, stability of operation).

There are two types of ICs testing:

- 1) functional – performed at all stages of logic circuit development (the purpose of functional testing is to verify the correct functioning of the designed circuit and its compliance with the technical specifications);
- 2) functional control of the correct operation of ICs samples after their manufacture.

The peculiarity of functional testing is that the circuit design process remains largely approximate and heavily dependent on the developer's experience, which can lead to the unpredictability of possible errors. To successfully solve this problem, it is necessary to create a hierarchy of functional tests in accordance with the hierarchy of fragments that make up the ICs. Each individual functional unit (accumulator, counter, shift register) is a relatively simple device for which it is possible to construct a complete test to verify its functionality. Blocks with significantly more complex logical functions are developed on the basis of the node. At the same time, there is no need to perform a complete functional test of each node included in the block, since it has already been performed earlier during its development.

When testing, special attention should be paid to checking the correctness of the connections between functional nodes within the block, as well as checking the internal states of the block itself. Similarly, when developing functional tests for ICs, special attention is paid to checking the correctness of

the connections between ICs blocks. The purpose of functional control is to establish the presence or absence of malfunctions in a specific ICs sample that interfere with its operation. That is, it is necessary to establish the conformity of the manufactured sample to the ICs design. A large ICs malfunction is a condition caused by the failure of one or more of its elements. Functional control contains one very serious contradiction between the quality of the test and the time required to test the large ICs, which determines the cost of testing.

For functional control, it is necessary to develop a set of control tests designed to check all states of the ICs. Possible ways to solve the problem of such control are: the development of methods for designing the testability of ICs, aimed at increasing the testability of ICs; the development of methods for built-in self-control of ICs; the development of methods for automatic synthesis and minimization of control tests; the development of methods for analyzing the quality of tests.

To evaluate the quality of tests, methods of modeling faults on a computer are used, the essence of which is to compare the results of modeling a functional circuit and a set of faulty circuits obtained by sequentially introducing specified faults into the functional circuit. If there is at least one test in the test sequence that reveals a discrepancy between the logical states of the outputs of the functional and faulty circuits, then such a fault is considered covered. Currently, methods that take into account probable factors are being actively developed.

1.4 Simulation modeling

Due to the fact that real systems are subject to random influences, there is a need to study the behavior of such systems. For this purpose, simulation modeling is used, which is currently the most effective means of studying complex systems. The simulation modeling block is included in typical CAD architectures. In simulation modeling, elementary events that occur in the system are simulated while preserving the logic of their interaction, the mathematical model reproduces the process of functioning over time. Simulation modeling has the following advantages: it allows you to study the features of the functioning of a real system; it allows you to solve complex problems; it significantly reduces the cost and duration of testing; it is a practical method for studying complex systems.

Simulation modeling allows solving problems of structure selection and assessment of the impact of various parameters, which is the basis of CAD. The main value of simulation modeling is that it is based on the methodology of system analysis. It allows you to study the system according to an operational research scheme, which includes the following stages: meaningful formulation of the problem; development of a conceptual model; development and software implementation of a simulation model; verification of the adequacy of the model and evaluation of the accuracy of the simulation results; planning of experiments; decision making. This allows simulation modeling to be used as a universal approach for decision-making in conditions of uncertainty, as well as to apply the basic principles of the systems approach in practice to solve practical problems (taking into account factors that are difficult to formalize in models). However, simulation modeling should only be used in cases where the development of other types of models does not give satisfactory results. This is because the implementation of a simulation model and conducting experiments on it is a labor-intensive task. It is important to note some ambiguity in modeling terminology, namely mathematical, simulation, and statistical modeling. The broadest class of models includes mathematical modeling.

A mathematical model is an abstract model represented in the language of mathematical relations, which can be specified in the form of analytical dependencies or algorithms. This class of models includes differential and integral equations.

Simulation modeling is a method of constructing a model of a system and setting up experiments on the model. As a rule, the simulation model is taken as its software implementation on a computer, and simulation modeling boils down to conducting experiments with this model by running the program on a certain set of data. The simulation model works in the same way as the system. Simulation modeling is performed to determine the structure and parameters of the system that provide the best performance indicators. From a mathematical point of view, a simulation model can be considered as consisting of equations that are solved by tracking the evolution of their solutions over a certain period of time.

Statistical modeling is a type of simulation modeling of probabilistic systems based on the method of statistical testing. The development of a

simulation model usually begins with a simple initial model, which is then refined and adapted to new conditions as the initial data and system characteristics are clarified. At the same time, the simulation model must remain sufficiently clear, and its level of detail is selected taking into account the modeling objectives, constraints, and initial data. Thus, the simulation model must be clear, multi-level, adaptive, developed iteratively, and refined and adjusted during its creation. It is advisable to program and debug the model in stages, with subsequent additions of software modules.

The methodology of system analysis in relation to simulation modeling includes several stages.

1. Formulation of the problem and setting of the task. At this stage, simulation modeling is used in the development of subsystems of various automated control systems (ACS) and is associated with methods of analysis and synthesis of production processes, systems for collecting, processing, and displaying information. The following tasks are examples of situations in which simulation modeling can be used: selection and justification of the technological route for processing parts; selection of the organizational structure for workshop management; analysis of the decision-making system for managing the enterprise workshop; analysis of the operation of the information collection, transmission, and processing system; analysis of user access time to the information base; selection of the composition of the complex of technical means for information processing in the ACS. This list of tasks shows that the simulation model is designed to be multi-purpose, allowing several different tasks to be solved. It should be noted that when formulating the problem for which the model is being developed, the modeling objectives are first determined, and then a targeted survey of the modeling object is conducted. As a result, at the content level, a description of the main characteristics of the system, input and output parameters, their interrelationship, external actions on the system are obtained, and the main criteria and limitations are determined.

2. Development of a conceptual model, an abstract model that reveals the cause-and-effect relationships inherent in the object under study. Its development begins with the compilation of a substantive description of the object's functioning process, which is carried out in accordance with the principles of system analysis: the criteria of the system or target functions are

formulated, the structure of the system, its input and output variables, external influences, and internal parameters of the system are determined. It should be noted that an incorrect concept underlying the model, incorrect assumptions about the relationship between variables and parameters will lead to the fact that the implementation of all subsequent stages of building a simulation model will be meaningless and will lead to unjustified costs.

It can be said that when developing a conceptual model, the task of structural and parametric identification in the broad sense of the word is actually solved. An important point is the choice of the necessary level of detail in the description of the system's functioning processes, determining the degree of accuracy of the description of the real process in order to obtain reliable information through modeling. However, it should be noted that although a more detailed model will be more accurate and have a wider scope of application, it will also be more expensive. Therefore, the level of detail of the system when building a model should be determined based on the principles of expediency. At the model development stage, the requirements for input data are determined. Some of the data can be obtained from the system's technical documentation, official reports, and reference books. At the same time, for newly created systems, the initial data may not exist – in this case, expert assessments obtained by surveying a group of specialists in the field are used to obtain them. It is important to remember that obtaining initial data is a critical stage, and using incorrect or inaccurate data will lead to incorrect modeling results. It should be noted that the process of building a conceptual model is iterative: first, a generalized model is built, then the model is detailed, and subsequently, the conceptual model is submitted for discussion with the user. This development process allows for the identification of inaccuracies and adjustments. At the end of this stage, a logical and mathematical description of the modeled system is provided, taking into account the dynamics of its functioning. Those parts of the model that can be described mathematically in analytical form are presented as analytical dependencies, while other parts of the model are presented as a detailed verbal description or algorithm. After the conceptual model is built, its adequacy to the object being modeled is checked.

3. Development and software implementation of the simulation model. At this stage, the conceptual model is first formalized. Such formalization

involves the construction of the simulation model structure. The model structure can be constructed in terms of the simulation modeling language or an application software package.

4. Verification of the adequacy of the simulation model.
5. Planning experiments on the model.
6. Evaluation of the accuracy of the modeling results.
7. Interpretation of modeling results and decision making.

The listed modeling stages are interrelated, and the procedure for building a simulation model is iterative. This is due to the fact that after each stage is completed, the model is checked for correctness, and if the model and the object do not match, the process returns to the previous stages for correction.

1.5 "System on a chip"

According to Moore's Law, the number of transistors on an ICs chip increases by approximately 60 % each year. At a certain point in time, it became possible to place equipment that used to be located on a single printed circuit board on a single chip. This is beneficial due to the reduction in total cost, the number of microcircuits required, power consumption, and increased reliability.

Thus, a single chip can accommodate not only a specific functional device, such as a central microprocessor, but also others, such as ADC, RAM, ROM, digital signal processing units, interface nodes, which complement it to form a complete system of units.

A "system on a chip" (SoC) is an integrated circuit that integrates various functional blocks on a chip to form a complete product. Since in most cases an SoC is a digital large ICs that may also contain a number of analog blocks, the same methods and tools are used for its design as for large ICs. The structure of a typical SoC is shown in Fig. 1.7. An SoC can include digital and analog blocks. The main block is usually a processor, and there is also a specialized digital signal processing block that performs specific functions. ADCs can act as peripheral devices; memory modules can be part of the SoC or can be connected as external blocks. Modern SoCs differ from microcontrollers only in the presence of specialized data processing units. Complex functional units such as processors, timers. are used in the design. These units are called IP (Intellectual Property) modules and are represented either as topological

fragments that can be directly implemented in the physical structure of the chip, or as models in the hardware description language.

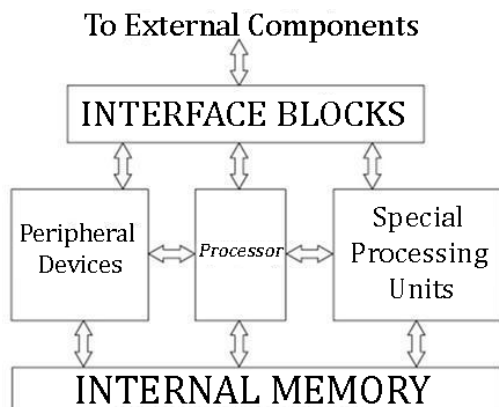


Fig. 1.7 – Structure of a "system on a chip"

Chip design began to be used at the stage of mastering technological processes for the production of 350 – 250 nm integrated circuits, when it became possible to integrate all the main digital components of the final product on a single silicon crystal. Once this level of technology made it possible to integrate the entire system on a single crystal, and subsequent technological processes made it possible to integrate even more complex systems, the question arose of how to completely fill this crystal. The main method proposed for increasing design productivity was the reuse of intellectual property, starting with small blocks (bus interfaces) and moving on to larger blocks (embedded processor cores). Then, moving on to sets of blocks that had been previously integrated and verified – to the so-called SoC "platforms".

Currently, top-down design is being replaced by a spiral design methodology. In this model, before moving on to the next stage, all design tasks at the current stage must be completed. At the same time, design is carried out simultaneously in four main areas: software development, code development, logical synthesis, and physical synthesis. In the course of their work, the development team constantly exchanges design results.

During the development of the SoC large ICs design, problems arose related to the software, hardware, and methodological support of the design process. The fact is that with the improvement of design standards for microelectronic products, the cost of production preparation increases, and, consequently, the price of the risk of technical and ideological errors. Nowadays, most large ICs development companies do not have their own production facilities, and crystals are manufactured at specialized silicon factories that provide developers with libraries for logical synthesis. Specialists perform the final refinement of photomasks and crystal manufacturing. In this case, the use of complex functional blocks, which are fully developed elements used in SoC, becomes relevant. This leads to an increase in the complexity of the design process. For the successful implementation of any complex project, it is necessary to organize its hierarchical decomposition. For example, at the register transfer level, the project is a set of arithmetic and logic nodes. The logic level describes the project at the level of logic gates and flip-flops. A set of templates of topological elements of the crystal corresponds to the geometric level. In this case, the behavior of the circuit is described by a system of logical equations. Typically, a top-down design model is used when designing large ICs. Such design implies minimal interaction between developers at different stages of the project.

At the first stage of design, the technical requirements must be developed and analyzed, and preliminary modeling must be performed. At the end of this stage, a complete functional verification of the technical requirements must be performed. It should be noted that in this case, the modeling task is complicated by the need to develop a model of test influences.

At the second stage, the project is described using the VHDL or Verilog language at the Register Transfer Level register transfer level. Based on the description at the register transfer level, a list of circuits is formed, which takes into account delays on library elements. Then, the crystal layout is developed and optimized by placing these elements and interconnections in an optimal way. After developing the topology, you can re-run the delay file generation and further time simulation, which will take into account the influence of interconnections. Then the crystal can be transferred to production and further testing of samples can be carried out. The disadvantage of this design methodology is the increased risk of errors as the complexity of the project

increases. In addition, it will only become clear at the very end of the design process to what extent the ICs under development meets the requirements proposed for it.

Modern microelectronic technology provides options for implementing SoC based on high-integration programmable logic integrated circuits (PLD – Programmable Logic Device), as well as in the form of custom large ICs. Currently, it is possible to place virtually any project, regardless of complexity, on a crystal. However, project verification technologies lag significantly behind the technologies and computing capabilities of design systems (as is well known, the degree of integration of modern large ICs is growing exponentially). According to the traditional, generally accepted design algorithm, which consists of each block being designed and verified separately, with joint verification taking place after combining into larger blocks, there is a high probability of an error being detected at one of the final stages of integration. This leads to additional time and resource costs for making changes. The solution to this problem lies in improving testing tools and changing verification methods.

1.6 Designing digital devices on PLD

Reducing the number of circuit components and increasing the degree of integration is a general trend in the development of modern microelectronics. At the same time, it is well known that reducing the linear dimensions of active elements on a crystal leads to an improvement in their functional characteristics. This trend has led to the creation of a technology for manufacturing large ICs with a minimum design norm of 45 nm. This is almost half the conventional upper nanotechnology limit of 100 nm, above which objects are considered to belong to the field of nanoengineering. The number of components in modern systems exceeds hundreds of millions, so the methods of designing nanosystems differ significantly from the methods previously used to design electronic systems, and this is only possible with the use of powerful computing complexes and design automation systems.

One of the combinatorial problems is the problem of topological design of nanoengineering objects. This task belongs to the class of complex tasks, because the time required to obtain an optimal solution with the growth of the degree of integration of large ICs as a nanoengineering object increases

exponentially. To solve it, heuristic algorithms are used, the use of which leads to a solution close to the optimal one. The cost of large ICs crystals consists of the cost of their design (which can be reduced by standardizing their topology) and the cost of production (which can be reduced by increasing the degree of integration), which increase almost proportionally to the number of wafers put into production. These two factors are mutually exclusive. In particular, for crystals manufactured in small series with a large production range, it is important to reduce the role of the first of these factors. Currently, almost all microprocessors are designed interactively, in a way in which the results of CAD work are evaluated and controlled by the developer at each iteration. It should be noted that if only CAD tools are used, this will lead to an increase in the size of the crystal. However, with "manual" design, which is quite time-consuming, this type of design allows for the creation of smaller large ICs crystals. Block-hierarchical topology design involves dividing the logical structure of the large ICs according to the principle of top-down design, the logical diagram is presented in the form of separate blocks of acceptable sizes. The blocks are then placed on the crystal and the interconnections between them are traced from the bottom to the top level. In Fig. 1.8 shows the topology of large ICs – first, blocks 1-5 are placed on the chip, and then blocks A, B, C, are placed in block 5.

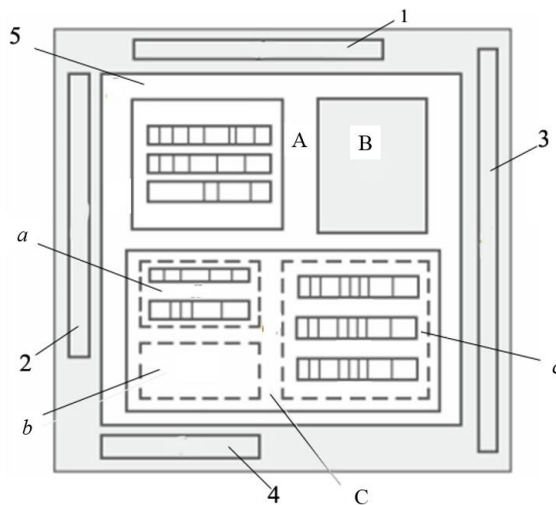


Fig. 1.8 – Topology of an NAND gate

The stage of topological design of nanoengineering objects is preceded by logical design, which is currently implemented using circuit descriptions in the VHDL language. Previously, Boolean equations were used in the development of circuits (for example, triggers). However, when a project contains thousands of flip-flops, the large number of Boolean equations makes their use impractical. Therefore, developers now prefer graphical representation of projects. This expands the possibilities for using logical equations by using additional circuits in addition to triggers, allowing more components to be placed with less effort than would be required with the Boolean equation method.

One of the most interesting areas of modern digital electronics is FPGA, which, as they become more complex and larger, are in high demand. FPGA are digital integrated circuits consisting of programmable logic blocks and programmable connections between these blocks.

The ability to configure these devices allows developers to solve a variety of different tasks. The main advantage of PLD over other specialized circuits is the short manufacturing time for the required custom circuit variants. There is no longer any need to contact ICs manufacturers to apply a metal mask and install the crystal in the case.

Many modern PLD contain built-in microprocessor cores and high-speed input/output interfaces and are used in communication devices, radar, and image processing. The cost of PLD is much lower than that of custom integrated circuits, and making changes to the device when using PLD is not particularly difficult and significantly reduces the time it takes for such devices to reach the market.

A PLD is essentially a custom microchip. Programmers and various debugging environments are used for programming, with which you can set the desired structure of a digital device either in the form of a schematic diagram or in the form of a program in hardware description languages.

The design of digital devices on PLD can be divided into the following stages:

- 1) system stage – the entire project is broken down into parts, their purpose and interrelationships are determined, and decisions are made on how to implement these parts;
- 2) structural-algorithmic and functional-logical – these stages are based

on the iterative introduction and verification of descriptions of parallel processes, each of which implements a given algorithm.

In this case, the device can be described in the following ways:

- using description languages (VHDL, Verilog);
- schematically (using a visual design program, after which the diagram is converted into a language description);
- graphical representation in a specialized editor that converts the resulting graphical representation into a language description;
- description of combinational logic using state tables and Karnaugh maps;

3) design and technology stage, which is divided into the following subtasks:

a) synthesis subtask – conversion of the initial circuit description of the device into a description that can be optimally implemented on the selected FPGA. At this stage, various methods of description optimization are used, aimed at achieving the best results in terms of the minimum required crystal resources and power consumption;

b) global placement – creating the best conditions for local placement and tracing;

c) local placement – detailed assignment of logical resources of macrocells to parts of the circuit;

d) tracing – determining the connections between logical blocks.

After each subtask is completed, the resulting description is verified using various modeling and analysis tools. After tracing and verification of the results, a file containing information about the switching and functionality of all crystal resources can be automatically generated. At the final stage of the design route, FPGA programming and device verification are performed.

The design of FPGA-based devices is performed using specialized CAD systems, which involves the sequential use of the provided software tools (this process is called the design flow). In Fig. 1.9 shows a generalized design flow used by Xilinx CAD systems. Design begins with the creation of a project, followed by the selection of one of the available synthesis programs, a simulation program, and a description language. It should be noted that the implementation route for a CPLD project differs from the FPGA implementation route in that it includes an assembly stage. After the tracing

stage, a PLD configuration file is generated, after which the developer specifies the sequence of the initial configuration upload to the PLD and programs all the necessary devices.

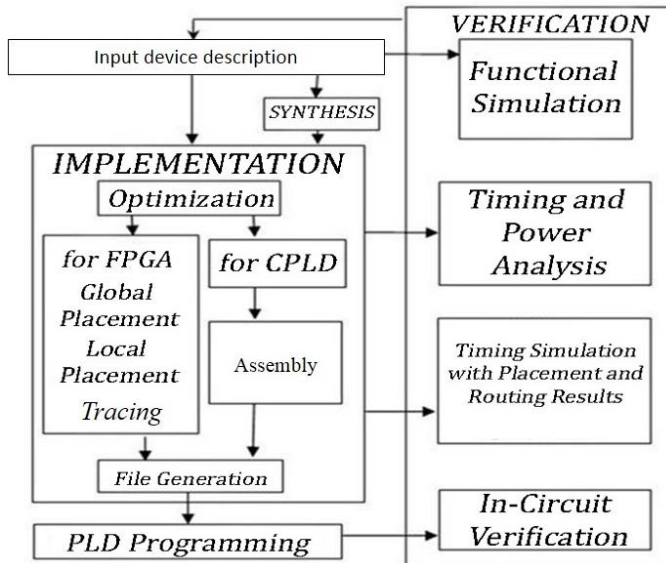


Fig. 1.9 – Generalized design flow

PLD are divided into two main groups:

1) CPLD (Complex Programmed Logic Device) – low cost and few resources, consisting of logic gate blocks connected by a programmable switching matrix;

2) FPGA (Field Programmed Gate Array) – contain more resources and more complex blocks compared to CPLD. The main difference between FPGA and CPLD is the need to load the configuration when the power is turned on. This means that FPGA require an external permanent storage device, while CPLD retain their logical structure even after power is turned off. However, a number of recent FPGA models also have modifications with non-volatile configuration memory.

Altera and Xilinx are the leaders in FPGA manufacturing. Xilinx software has long been known as a means of ensuring high performance of developed devices and is a simple and inexpensive product.

Altera is one of the largest developers of FPGA and focuses on the development of circuits and modules based on VHDL, Verilog, and AHDL languages. Its main products are programmable microcircuits, as well as services for converting projects to FPGA for mass production. The company also produces software for developing embedded software for PLD and compilers for its own processor core. Currently, the software products are Max+Plus and Quartus.

The Max+Plus II software product is an environment for developing digital devices based on PLD, which provides the following stages: project entry; data preparation for PLD programming; project verification; PLD programming.

The Quartus II software product provides a complete cycle for creating FPGA and also offers developers extensive options when compiling projects. Hardware description languages (HDL) are used to program FPGA. Among them, the most popular are Verilog and VHDL (Very High Speed Integration Circuits Hardware Description Language). The main task of these languages is to describe electrical digital circuits in a manner similar to graphical diagrams.

Traditional programming languages are not suitable for describing hardware because they were created to describe algorithms, which is not the same as a digital circuit. Programming languages are characterized by syntax and semantics. Syntax defines the grammatical rules for writing programs and using language constructs, while semantics defines the meaning of these constructs.

Programs written in traditional programming languages simulate a sequential process. Since the result of executing individual operators often affects subsequent operators, it is not possible to change their order. VHDL and Verilog are similar in many ways, but they have different syntax and differ in some details.

VHDL provides the possibility of hierarchical design. This language was developed for the purpose of formally describing logic circuits for all stages of electronic system development.

The VHDL language is used for the following purposes:

- automating the conversion of the initial circuit description into a lower-level description;

- describing the behavior of digital devices over time and when input influences change;
- describing the structure of digital devices with varying degrees of detail;
- describing test inputs when modeling devices;
- modeling digital devices.

VHDL allows for design at different levels of abstraction, and its syntax is largely similar to that of universal programming languages. However, specific language elements (signals, ports, configurations) are used to describe complex objects.

Verilog is a hardware description language used to describe and model electronic systems. It is a fairly simple language, similar to the C programming language.

VHDL and Verilog are considered standard languages for developing digital devices, while SystemVerilog is an extended version of Verilog (it looks like a hybrid extension of Verilog, containing HDL hardware description and an object-oriented approach to programming). In Verilog and VHDL, developers can represent the required functionality in the form of program text, and then simulate and debug the program.

VHDL is a strictly typed and deterministic language, and more detailed than Verilog. Due to its structure, VHDL allows you to track more errors than Verilog in the early stages of the development process. Currently, engineers effectively use VHDL, Verilog, and SystemVerilog for their ASIC designs.

1.7 Theoretical information about electromagnetic compatibility

The electromagnetic compatibility (EMC) of an electronic device is its ability to function with the required quality in conjunction with other devices under conditions of possible interference from unforeseen sources, without causing unacceptable interference to other devices. Ensuring the EMC of technical devices is an independent scientific and technical field that covers numerous aspects of radio engineering, electronics, and electrical engineering. The issue of EMC is one of the most important in the electric power industry, and its economic nature indicates the enormous losses that arise from non-compliance with its requirements.

The technical characteristics of an electronic device can be divided into functional characteristics, which determine the properties of the device, and

those that affect EMC. To ensure the EMC of devices, it is necessary to rationally select radio channel frequencies for specific radio services and perform calculations taking into account functional characteristics and EMC characteristics. Technical measures are divided into design and technological measures and circuitry measures.

The general concept of the methodology for creating and operating devices with EMC in mind is based on a systematic approach to solving the problem of ensuring EMC, identifying all factors that affect it during the joint operation of the system or device being developed, as well as determining the causes of these factors and establishing the relationships and interactions between them. The systematic approach leads to a multifaceted solution to the problem of ensuring EMC at all levels and comprehensive solutions aimed at improving interference immunity and reducing interference energy at its sources and in the propagation environment.

Electromagnetic interference is an electromagnetic, electrical, or magnetic process that causes unwanted changes in a useful signal during its transmission, reception, or conversion to a specified form. Electromagnetic interference always exists in power supply systems: all electrical equipment elements generate electromagnetic interference, they are sources of electromagnetic interference emissions. In the field of EMC, the source refers only to the source of interference that arises unintentionally or due to insufficient technical and organizational measures.

Electromagnetic interference can be classified as stationary (interference from the antenna of a radio transmitting device), industrial (interference from electronic, electrical, and radio devices used in everyday life, industry, transport), contact, and natural.

In order to plan the EMC of a system, the electromagnetic conditions, communication mechanisms, and sensitivity of the interference receiver between two electrical circuits must be known. The intensity of nearby magnetic and electric fields in free space is inversely proportional to the square of the distance from the element that excites the field. When analyzing the impact of electromagnetic interference, special attention is paid to the cause of its occurrence, the ways it spreads, the specifics of its impact on various objects, as well as measures, methods, and means of protection and suppression of this interference.

By the type of propagation, spatial and conductive interference are distinguished. The former is characterized by the influence of an electromagnetic field that is emitted and propagates in space, while the latter penetrates the equipment through conductive communication and power supply channels. Conductive interference refers to interference that occurs when communicating through common resistance, for example, through grounding buses or power sources. In this case, currents from different circuits flow through common resistance, and the voltage drop across it from each current will interfere with other circuits. Depending on the location of the interference source relative to the electronic device under study, interference is classified as external (caused by processes in other devices), internal (arising as electromagnetic phenomena and connections not provided for by the circuit and design of the device), and intrinsic (noise from components associated with the operation of the device itself). Depending on the type of signal, interference is classified as random or deterministic. Interference can be pulsed, broadband, or narrowband. This classification allows you to assess the overall interference situation, analyze it, and develop an approach to eliminate the undesirable effects of a particular type of interference on the device.

Technical literature uses special terms to characterize the interference environment:

- 1) The receiver (receptor) of interference is any system or its constituent parts, down to the elements and individual components affected by the interference.

- 2) susceptibility is the measure of the receiver's response to interference, characterizing its ability to reduce the quality of functioning under the influence of interference;

- 3) interference immunity is the ability of a receiver to maintain its performance under the influence of interference, to resist it by means of circuit design measures;

- 4) interference protection is the ability of a receiver to maintain its performance and counteract interference through circuitry, design, technological, and other protective measures that do not alter the principles of operation and construction of the receiver.

Solving the EMC problem of a system usually begins with studying the electromagnetic environment, the combination of electromagnetic, electric,

and magnetic fields, as well as interference currents and voltages that can affect the functioning of this system.

One type of unintentional electromagnetic interference is crosstalk, an undesirable phenomenon in which, when conductors are located close to each other in parallel, a signal on one of them (the source of interference) causes a change in the signal on the other (the receiver of interference). This can cause the electronic device to malfunction. The effect of crosstalk manifests itself in the input path of an electronic device as a change in the amplitudes or phases that make up the spectrum of the useful signal. Crosstalk can occur inside microcircuits, as well as on printed circuit boards, connectors, microcircuit housings, and cables. If there are two or more signal conductive tracks on printed circuit boards in long sections that are parallel and close to each other, interference may occur due to mutual influence at the inputs of "passive" circuits.

Cross-interference is caused by parasitic capacitive, inductive, or conductive connections in an electrical circuit. Capacitive coupling causes a sudden increase in current in the lines, which leads to reflection on the transmission lines. There are also inductive couplings created by magnetic fields. They mainly cause power supply noise. Cross-interference decreases proportionally to the square of the distance, and the degree of influence is proportional to the surface area of the board, the distance between the tracks. There is a wide range of possible actions to reduce cross-interference related to changing the distance between conductors, rearranging signals in conductors, and shielding them.

A digital circuit must be designed so that no voltages exceeding the specified ranges appear at its output. However, under certain circumstances, this can happen. For example, if the output of a digital microcircuit is heavily loaded, it will not be able to generate voltage at certain levels due to the limited efficiency of the signal source. Synchronous buses are used for parallel transmission. They are characterized by very good resistance to crosstalk, which can only occur at the moment of state change, and within a narrow time window.

Cross-interference does not affect signal quality as long as the receiver waits long enough for it to stabilize before sampling the bus. If cross-interference occurs during a signal state change, its only effect is jitter (phase

noise). Asynchronous and ungrouped signals are always sensitive to cross-interference.

When designing a high-speed board, it is necessary to consider the issue of signal integrity, guide and verify the design of this board. The issue of signal integrity causes an increase in signal frequency, a decrease in the size of the printed circuit board, an increase in wiring density, and a decrease in the thickness of the intermediate layer caused by an increase in the number of board layers. Design at the board and system levels is a complex process and includes the following steps: circuit design, microchip selection, schematic design, printed circuit board layout, and tracing. During the design process, it is necessary to identify crosstalk at various stages and take measures to suppress it. Calculating crosstalk is very complex. The amplitude of the crosstalk signal is affected by the degree of coupling and the distance between the conductors.

The design of high-speed printed circuit boards requires both a detailed understanding of theoretical concepts and the continuous accumulation of experience and improvement of theory. At the same time, the skillful use of appropriate software can also shorten the design cycle, which will increase competitiveness and play an important role in the successful completion of the design.

Depending on the number of layers of the printed circuit board, boards are divided into single-sided, double-sided, and multi-sided. Single- and double-sided boards are cheaper and easier to manufacture, but their electrical characteristics cannot be considered satisfactory, so to increase the speed of digital equipment, preference should be given to multilayer boards, the use of which in most cases is the only way to implement micro-miniature equipment.

Conductor systems that form a communication line with electromagnetic field localization can be divided into the following types: single-layer without shielding, double-layer with a shielding plane, and triple-layer with two shielding planes. With an increase in the number of shielding planes in these systems, the electromagnetic field is more concentrated in the area between the conductor and the shield, which leads to a decrease in the level of induction interference.

The design of multilayer printed circuit boards with EMC in mind can be reduced to the following steps:

- selection of the basic structure of the board with a specified number of signal and potential layers;
- calculation of the geometric dimensions of the links that make up the basic structure;
- if there are several base structures that correspond to the specified number of signal and potential layers, selecting the structure with the smallest thickness;
- verification of the board implementation by comparing its thickness with the technologically possible one.

The advantages of multilayer printed circuit boards from the point of view of internal EMC include the stability of the electrical parameters of communication lines, the ability to ensure the required wave resistance of the line by design methods, and a sufficient degree of shielding of signal conductors located in the inner layers of the board. Shielding is a design means of ensuring the attenuation of electromagnetic interference fields within a certain space. Its purpose is to increase interference immunity and ensure the EMC of electronic devices. Structures that implement these requirements are called shields.

Electromagnetic shields work as follows: the electromagnetic field penetrates the shield wall and excites charges or induction currents in it, whose own fields overlap the initial field, partially or completely compensating for it. Screens can be used for individual functional units, equipment blocks, and for the electronic device as a whole. Materials with high conductivity for active field flows are used to manufacture screens, which are capable of creating a counteracting magnetic field due to induction. The main means of attenuating conductive interference generated in the power and switching circuits of direct and alternating current equipment is filtration. Filters reduce this interference from both external and internal sources.

The main requirements for filter operation are:

- ensuring the specified filtration efficiency in the required frequency range;
- limiting the permissible drop in DC or AC voltage across the filter at maximum current;
- limiting the permissible value of the reactive component of the current at the fundamental frequency in accordance with safety requirements;

- ensuring acceptable nonlinear distortions of the supply voltage, which determine the requirements for filter linearity;
- filter elements must be selected taking into account the rated currents and voltages of the electrical circuit.

Digital integrated circuits can generate and receive external interference. To reduce interference in digital logic circuits, it is necessary to work with signals that have longer rise and fall times and lower amplitudes, limit the number of simultaneously switching signals, and use effective shunting and grounding methods. To increase the resistance of circuits to external interference, slow synchronization circuits should be used. If the system has long conductors, it is advisable to use differential transmitting and receiving devices connected by symmetrical communication lines to reduce the level of interference.

Shunt capacitors are sources of impulse current consumed by digital circuits during switching, reduce voltage drops in power and ground circuits, and help filter interference generated by power and ground sources. In the 30 MHz to 1 GHz frequency range, clock signals and their harmonics are the main cause of radiated interference. Even harmonics can be reduced by using clock signals with a 50 % duty cycle.

Try to reduce the number of integrated circuits controlled by each clock signal. If synchronous signals must be supplied to several boards, it is advisable to use Schmitt trigger input logic elements as a buffer and to limit the voltage swing and rise time of the main synchronizing signals. The interference problem can be solved by carefully synchronizing the system. To reduce transient currents arising in power supplies and grounding devices, a small group of microcircuits should be controlled using spaced clock signals.

CHAPTER 2

USING ORCAD FOR DEVICE MODELING

Practical work 1

ASCENDING AND DESCENDING DESIGN

The purpose of the work: Perform ascending and descending design for hierarchical devices and evaluate the advantages and disadvantages of each type of computer-aided design.

Creating a project in OrCAD Capture

When creating a new schema project in *Capture*, it is recommended to avoid long path names and file names and to use special characters to denote a chain, node, project or library: ?, @, ~, #, &, %, «, !, (), <, =, >, [], *. To create a new project, run *Orcad Capture*. In the *File* menu, select *New Project*. In the dialog box (Fig. 2.1.1) define the project name as *lab1*. Select *Analog* or *Mixed A/D*, determine the directory in which the project files will be created.

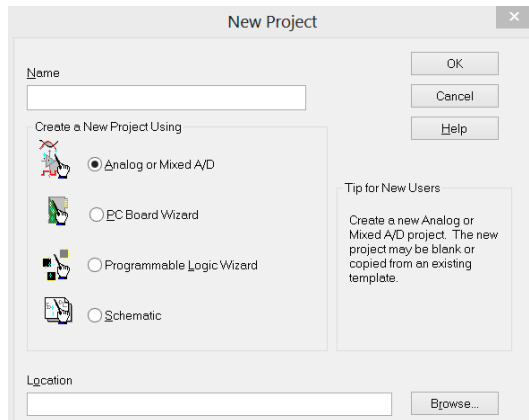


Fig. 2.1.1 – *New Project* dialog box

In the *Project Manager* window, right-click on SCHEMATIC1 and select *Rename* in the menu (Fig. 2.1.2). Let's change the name of the schema directory and the schema page to CODER.

Before you start creating a project, you need to determine the

characteristics of the project. In the *Options* menu, select *Design Template*. Fonts for various objects are set on the *Fonts* tab, corner stamp inscriptions are set on the *Title Block* tab, and the Page Size is set on the *Page Size* tab (Fig. 2.1.3).

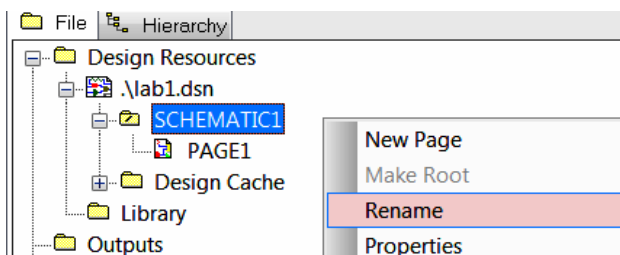


Fig. 2.1.2 – Rename the schema directory

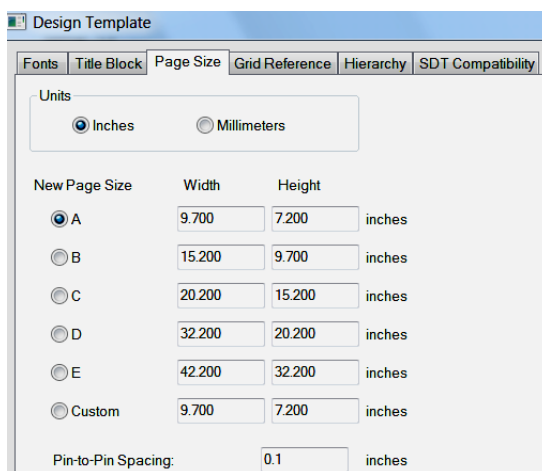


Fig. 2.1.3 – *Page Size* tab of the *Design Template* dialog box

The *Pin-to-Pin* Spacing field specifies the minimum distance between the pins in graphical symbol (GS) elements, and this parameter determines the step size of the grid.

The parameters of the frame around the diagram sheet are set on the *Grid Reference* tab (Fig. 2.1.4):

- *Alphabetic* – numbering of fields in alphabetical order;
- *Numeric* – numbering of fields in numerical order;
- *Ascending* – setting the field numbers of the frame (in ascending order);
- *Count* – the number of fields on the frame horizontally and vertically;
- *Width* – frame width horizontally and vertically;
- *Descending* – setting the field numbers of the frame (in descending order);
- *Title Block Visible* – display of the main inscription on the display (*Displayed*) and when printing (*Printed*);
- *Border Visible* – display of page borders on the display (*Displayed*) and when printing (*Printed*);
- *Grid Reference Visible* – display of the frame of the diagram sheet on the display (*Displayed*) and when printed (*Printed*);
- *ANSI grid references* – image of the frame of the diagram sheet according to the ANSI standard.

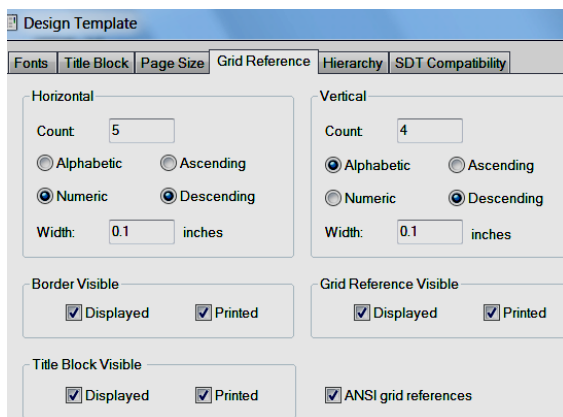


Fig. 2.1.4 –Grid Reference tab of the *Design Template* dialog box

Parameters that are accepted by default when creating new hierarchical blocks (*Hierarchical Blocks*) and elements (*Parts*) are specified on the *Hierarchy* tab (Fig. 2.1.5): *Primitive* – primitive components; *Nonprimitive* – components with a hierarchical structure.

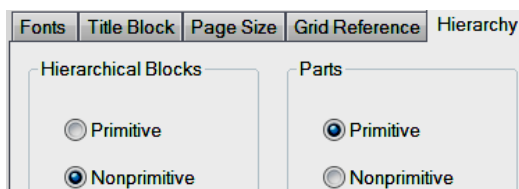


Fig. 2.1.5 – *Hierarchy* tab of the *Design Template* dialog box

Creating a flat (single-level) project

Let's create a simple flat *Coder* project. By clicking the *Place Part* button  and in the dialog box, select the library (*Add Library* button) from which the item should be added, and then place it on the page.

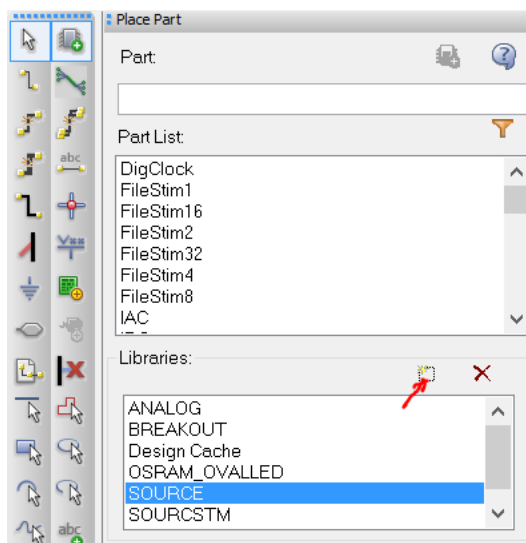


Fig. 2.1.6 – *Place Part* dialog box

Using the *Browse File* window in the */tools/capture/library/pspice* directory, select the 7400.0LB library and click *Open*. The 7400 library will appear in the *Libraries* list. In the *Part List*, select 7404 and click *OK*. Similarly, place the other elements. Right-click and select *EndMode*. In the

Place menu, select *Wire* and connect the elements (Fig. 2.1.7).

If it is necessary to connect elements on the scheme by means of buses, the following operations are carried out:

1. In the *Place* menu, select *Bus*. Then in the *Place* menu, select *BusEntry* to place the pins of individual circuits.
2. To assign names (aliases) of tires and chains that are part of them, in the *Place* menu, select *Net Alias*. When assigning chain aliases, their numbers are automatically incremented by one (eg A1, A2, AZ). The name of the tire is written as A [1...3].

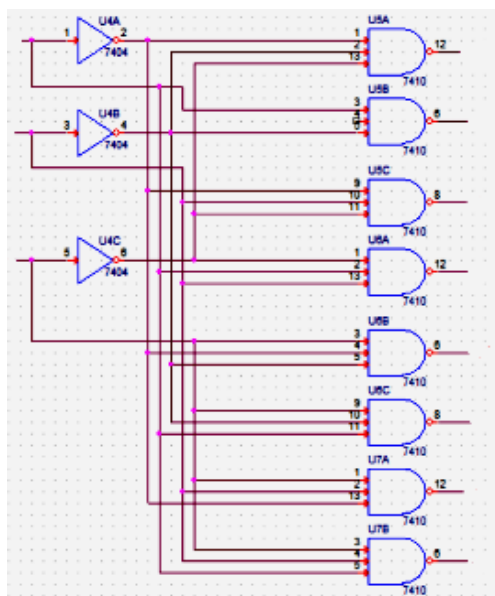


Fig. 2.1.7 – Connected elements

To add input and output ports to the project, perform the following sequence of operations:

1. In the *Place* menu, select *Hierarchical Port* (you can click the *Place Port* button on the toolbar). The *Place Hierarchical Port* dialog box opens.
2. In the Libraries tab, select *CAPSYM*.
3. Add the PORTRIGHT-R input ports (Fig. 2.1.8).

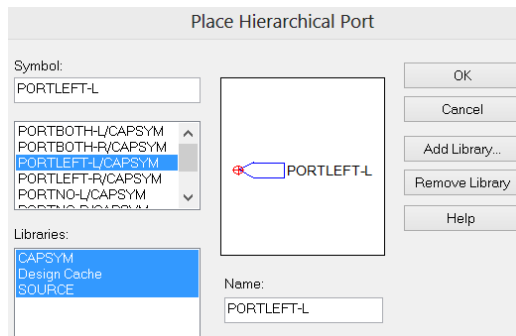


Fig. 2.1.8 – Choice of ports

4. Add eight output ports. To do this, select *PORTLEFT-L* in the *CAPSYM* library.

5. To rename the ports, double-click on the port name and in the *Display Properties* dialog box, change the value of the *Name* properties of the input ports to *X1..X3* and the output ports to *Z0 ... Z7* (Fig. 2.1.9). Save the project by clicking *Save*.

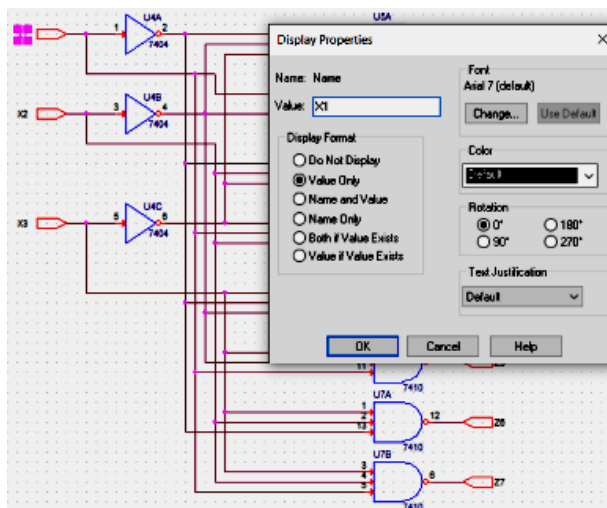


Fig. 2.1.9 – *Display Properties* dialog box

In *Capture*, you can create hierarchical projects using one of the following methods: *Bottom-up method* or *Top-down method*.

Ascending method of creating a hierarchical project

When creating a hierarchical project using the ascending methodology, the following operations must be performed:

- create a project of the lowest level;
- create higher level projects (in the form of hierarchical blocks).

Consider the creation of a *DC* project in ascending methodology. To do this, perform the following operations:

1. Let's create the lowest level project (in our case it is already created the *Coder* project).
2. In the *Project Manager* window, right-click on *lab0.dsn* and select *New Schematic*, then name it *DC*.
3. To make *Coder* the root scheme of a high-level project, right-click on *DC* and select *Make Root* from the menu. The *DC* directory will move up and a slash will appear on it.
4. Right-click on *DC* and select *New Page*.
5. In the *New Page in Schematic* dialog box, define the page name as *DC* and click OK. A new *DC* page (Fig. 2.1.10) is added below the *DC* circuit directory.

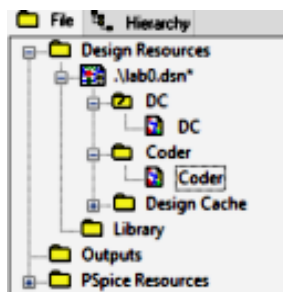


Fig. 2.1.10 – Creating a new page

6. To open a page for editing, double-click on its name with the left mouse button.

7. In the *Place* menu, select *Hierarchical Block*, in the dialog box which will define the reference position *Reference* as *DC*, *Implementation Type* as *Schematic View*, *Implementation name* as *CODER* and click *OK*.

8. Let's draw a rectangle on the page of the scheme. A hierarchical block with input and output ports will appear on the page (Fig. 2.1.11).

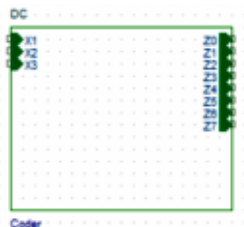


Fig. 2.1.11 – Hierarchical block

9. If necessary, change the size of the block (you can rearrange the input and output ports). To check the correctness of the hierarchical block, click on it with right-click and select *Descend Hierarchy*. The previously created *Coder* project will open.

10. Add a signal source. In the *Place Part* dialog box, click the *Add Library* button to add the SOURCESTM.OLB library (located in the */tools/capture/library/pspice* directory).

11. In the *Part List*, select *DigStim1* and click *OK* (Fig. 2.1.12). The GS of the element "sticks" to the cursor.

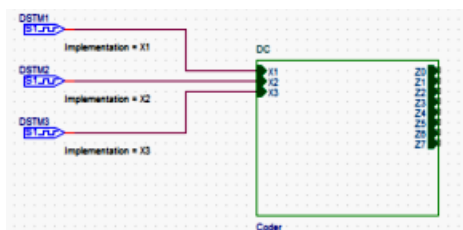


Fig. 2.1.12 – Installation of input sources

12. Place the signal source next to the three input ports.

13. Right-click on the diagram and select *End Mode*.
14. Define the value of *Implementation property* as *X1, X2, X3*.
15. Input wires can be integrated into the bus. Add the source ports and change their names to *Z0 ... Z7* (Fig. 2.1.13).
16. Let's save the project.

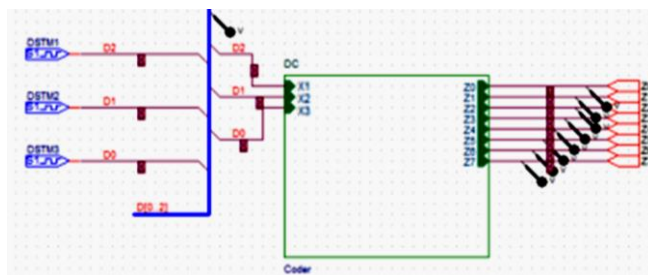


Fig. 2.1.13 – Decoder circuit

The simulation results are shown in Fig. 2.1.14.

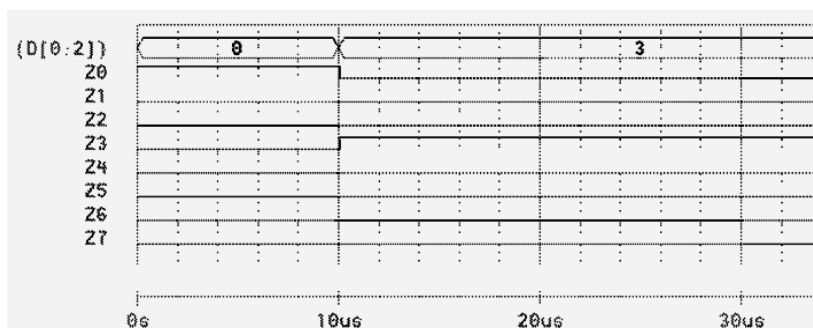


Fig. 2.1.14 – Simulation results

Descending method of creating a hierarchical project

When creating a descending hierarchical project, use the following sequence of operations:

- create a top-level project using functional blocks, the inputs and outputs of which are known;

– create a draft scheme for the functional block, which is used in the project of the upper level.

Consider the operations that must be performed to create a *Coder* project using a descending methodology.

Let's create the project of the top level for what we will make the following:

1. In the *Place* menu, select *Hierarchical Block* or click the *Place hierarchical block* toolbar button.

2. In the *Place Hierarchical Block* dialog box, define the *Reference* designation as *DC*, the *Implementation Type* as *Schematic View*, and the *Implementation name* as *CODER*. Unlike a hierarchical block created by ascending methodology, it has no attached ports.

3. Select the hierarchical block and select *Hierarchical Pin* in the *Place* menu.

4. In the *Place Hierarchical Pin* dialog box, define the output name as *X1*, *Type* as *Input*, *Width* as *Scalar* and click *OK*.

5. Place the input output on the hierarchical block.

6. Add input pins *X2*, *X3* and eight output pins *Z0* ... *Z7* as shown in Fig. 2.1.15.

7. Let's finish creation of the scheme, having added for this purpose ports, conductors and signal sources. Let's save the project.

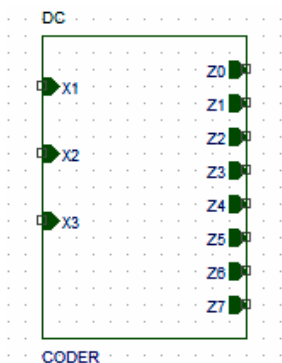


Fig. 2.1.15 – Hierarchical block with ports

Let's draw the project of lower level (in our case such project is the *Coder* scheme). To do this, perform:

1. Right-click on the hierarchical block and select *Descend Hierarchy* in the menu.

2. The *New Page in Schematic* dialog box opens. Define the page name as *DC*. A new schema page will appear with three input ports *X1*, *X2*, *X3* and output ports *Z0* ... *Z7* (Fig. 2.1.16).

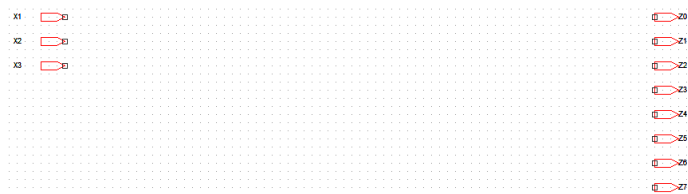


Fig. 2.1.16 – Scheme page with input and output ports

On this page, you can draw a *Coder* diagram using the operations described earlier when creating a one-tier project.

Adding to the hierarchical block sources of input signals and output ports, we obtain a scheme similar to that shown in Fig. 2.1.13.

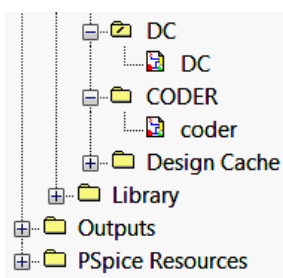


Fig. 2.1.17 – Dialog box with added *CODER* directory

Navigation in a hierarchical project

To move to a lower level of the hierarchy, right-click on the hierarchical block and select *Descend Hierarchy*. Similarly, to move up the hierarchy, right-click and select *Ascend Hierarchy*. The *Ascend Hierarchy* and *Descend*

Hierarchy menu options are also available in the *View* menu.

When working with hierarchical projects, you can make changes in hierarchical blocks, as well as in schemes at the lowest level. You can save hierarchical levels with changes using the *Synchronize* options available in the *View* menu. If changes have been made at the lowest level of the project and you want them to be reflected at a higher level, select *Synchronize Up*. If changes have been made to the hierarchical block and you want them to be reflected in all sample blocks, select *Synchronize Across*. If changes have been made in the hierarchy and you want them to be reflected in the lowest level project, select *Synchronize Down*.

Creating a graphic symbol for the scheme

Instead of creating a hierarchical block for a *Coder* project, you can generate a GS for it and then reuse it in any other project. In this section, we generate a GS for the *Coder* schema that was created earlier. To do this, perform the following operations:

1. In the *Project Manager* window, select the *CODER* directory with the cursor, select the *Generate Part* in the *Tools* menu and determine the location of the project file that contains the schema for which the GS is to be created.

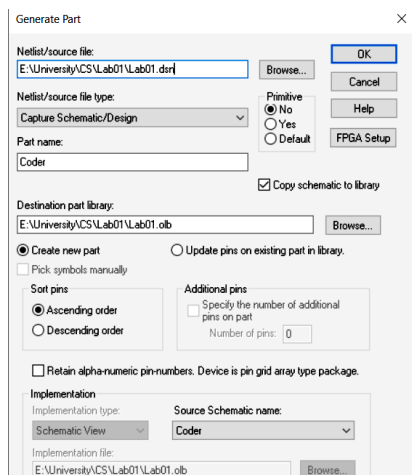


Fig. 2.1.18 – *Generate Part* dialog box

2. In the *Netlist / source file type* list, define the project type as *Capture Schematic*; in the *Part Name* text box, define the GS name as *CODER*.

3. Determine the name and location of the library that will contain the created GS (for example, lab01.olb).

4. If you want the source schema to be saved next to the new GS, select the *Copy schematic to library* check box. Make sure the *Create new part* option is selected. To determine the circuit directory, select *CODER* from the *Source Schematic name* list.

By clicking *OK*, we will generate a GS *Coder*. The new lab01.olb library will be created and displayed under the *Outputs* directory in the *Project Manager* window (Fig. 2.1.19). It will also be added in the *Place Part* dialog box. You can now use the *Place Part* dialog box to add a *Coder* to any project.

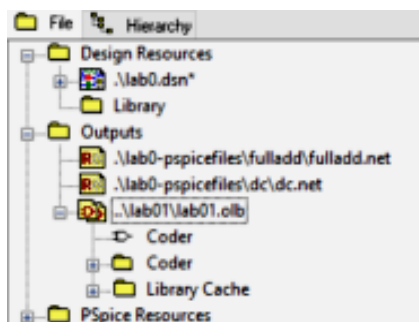


Fig. 2.1.19 – A library has been created

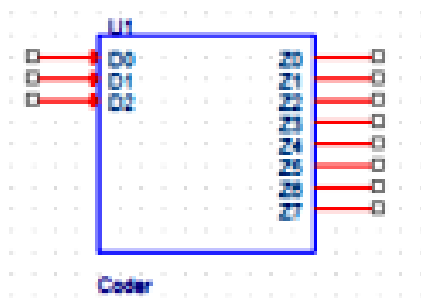


Fig. 2.1.20 – GS of a decoder

Add item placemarks

To transfer the project scheme to ORCAD *Layout* (PCB breeding program), all components in the project must be uniquely identified by the positional designation of the elements. In *Capture*, you can assign a placemark either manually or using the *Annotate* command.

In our project, an annotation is not required at this stage, because by default the positional designations of the element are attached to all components. Placement of placemarks is done automatically – by default *Capture* adds the placemark of the element to all components that are placed on the page of the scheme. If necessary, you can disable this feature by performing the following operations:

1. In the *Options* menu, select *Preferences*.
2. In the *Preferences* dialog box, select the *Miscellaneous* tab.
3. In the *Auto Reference* field, remove the "check mark" in front of the *Automatically reference placed parts* command.
4. To save these settings, click *OK*.

If the components in the project do not have positional symbols, you must run the *Annotate* command.

To assign an item to the components in the created project using the *Annotate* command, perform the following operations:

1. In the *Project Manager* window, select the file lab0.dsn.
2. In the *Tools* menu, select *Annotate* or click the *Annotate* button on the toolbar. The *Annotate* dialog box is shown on Fig. 2.1.21, it has the following fields:

Scope:

- *Update entire design* – update position designations and packaging information of the entire project;
- *Update selection* – update the position designations and other information of the selected part of the project.

Action:

- *Incremental reference update* – update the position designations and packaging information of components in which the question mark "?" is placed instead of the number, while the component numbers are increased by one;
- *Unconditional reference update* – updating of position designations and packing information of all components in the chosen branch;

- *Reset part reference to "?"* – replacement of component numbers with "?";
- *Add Intersheet Reference* – adding links to other pages;
- *Delete Intersheet Reference* – delete links to other pages.

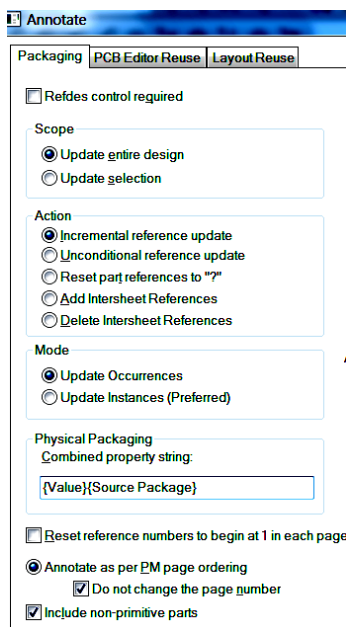


Fig. 2.1.21 – *Annotate* command dialog box

Mode:

- *Update Occurrences* – updating the parameters of all individual samples of the component;
- *Update Instances* – update component parameters and all links to it;
- *Physical Packaging* (automatic packaging of components according to the specified properties);
- *Combined property string* – property string;
- *Reset reference numbers to begin at 1 in each page* – start from the unit numbering of positional symbols of the same type of components on each page;

– *Do not change the page number* – do not change the page number.

3. In the *Packaging* position of the *Annotate* dialog box, define the need to change the complete project or only the project element. Select the *Update entire design* option.

4. In the *Actions* field, select the *Incremental reference update* option.

5. For the created project, select the *Update Occurrence* option.

Note: When you select the *Update Occurrence* option, you may receive a warning. This message should be ignored because *Occurrence* mode is the most acceptable for all complex hierarchical projects.

6. For other options, accept the default value and click *OK* to save the selected settings. The *Undo Warning message* window opens.

7. Click *Yes*. A report window will open stating that an annotation will be created.

8. Click *OK*. The project will bear the positional designations of the components and it will be saved. You can view the updated placemarks on the chart page.

Creating a Cross reference report

Using *Capture*, you can create a *Cross reference* report for all elements in the project. The *Cross reference* report contains information such as the name of the items, their placemarks, and the libraries from which the item was selected.

To generate a *Cross reference* report:

1. In the *Tools* menu, select *Cross References* or click the *Cross Reference Parts* button on the toolbar.

2. In the *Cross Reference Parts* dialog box, select the *Cross reference entire design* option. If you need to generate a *Cross reference* report for a specific directory, select the schema directory before opening the *Cross Reference Parts* dialog box.

3. In the *Mode* field, select the *Use Occurrences* option.

4. Define the report you want to create.

5. If you want the report to be automatically displayed on the screen, check the *View Output* field.

6. Click *OK* to generate the report.

Tasks for practical work

Perform ascending and descending design of a given device according to Table 2.1.

Table 2.1

№	Device
1	4-bit adder
2	16-bit adder
3	12-bit adder
4	encoder 8×3
5	3-bit register
6	multiplexer 5×1
7	8-bit adder
8	multiplexer 6×1
9	3-bit adder
10	binary counter with $K = 5$
11	binary counter with $K = 3$
12	binary counter with $K = 4$
13	4-bit register
14	6-output decoder
15	10-output decoder

Practical work 2

CREATING YOUR OWN ELEMENT BASED ON BASIC COMPONENTS

The purpose of the work: Acquire practical skills to create your own element in the OrCAD environment.

The OrCAD suite uses a library design method, which involves the device being "assembled" from individual components that are combined into libraries. The user can create a new description of components and include them in existing libraries, as well as create personal libraries. In the schematic diagram, the component is represented as GS, it is called a symbol (*Part, Symbol*). On the PCB, the same component looks different. The main thing here is to give its physical size and shape to find out how much space will be needed to place it, this description is called design or physical, it is stored in a library with the extension *.lib. To model a project you need to have a description of the function of the component, the algorithm of its operation. Symbols are divided into main and auxiliary.

Auxiliary characters are a small group that includes page connectors, hierarchical block ports, corner stamps, and power supply symbols. The main symbols are divided into simple and hierarchical. Simple symbols are called structural primitives and are not subject to detail during design. Primitives are objects with known behavior, they always have a functional description.

Hierarchical symbols in OrCAD are implemented in the form of hierarchical blocks (*Hierarchical Block*) or primitive symbols. Hierarchical symbols have two types of description – external and internal. When the character description level decreases, it is transformed into a structure, showing the "hidden" subcircuit or VHDL code. The internal description in most cases is a subordinate scheme, which is called a substitution scheme or equivalent scheme. Simple symbols include symbols for real and abstract components. The main difference between abstract components is that they do not specify a specific physical implementation. For example, you can depict a two-input multiplexer, describe its function and then use it in a project, although in reality there is no such chip. Due to the lack of data on physical implementation, abstract components do not allow to bring the project to

reality. However, they are widely used in the early stages of functional design and in the construction of hierarchical blocks.

Consider the process of creating your own component and all its parts on the example of a multiplexer. To begin with, let's create our own library. Launch the graphic editor OrCAD *Capture* and select *File / New / Library*. Let's try to copy to your library some components from the system library, for example, from *t11.olb*. Using the *File / Open / Library* command, load both libraries into the OrCAD Capture editor and place their windows side by side (Fig. 2.2.1). Left-click select one element, for example, 7404, then select *Copy* in the menu and paste from the buffer into the window of your own library (Fig. 2.2.2).

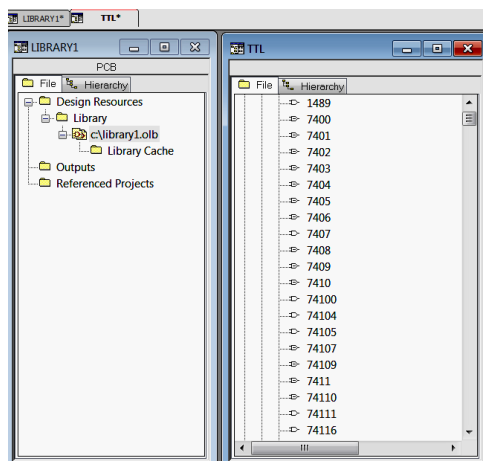


Fig. 2.2.1 – View of the *t11* library

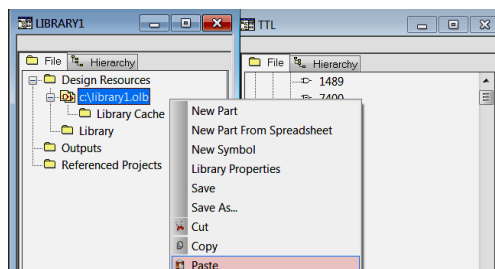


Fig. 2.2.2 – The process of copying items from the *t11* library

Using the method described above, copy components 7408 and 7432 to the personal library. Now rename your own library to NEW_LIBRARY.olb. To do this, right-click on the line with the name of the library, and when the menu opens, execute the command "Save as". Enter the desired name and save the library (Fig. 2.2.3).

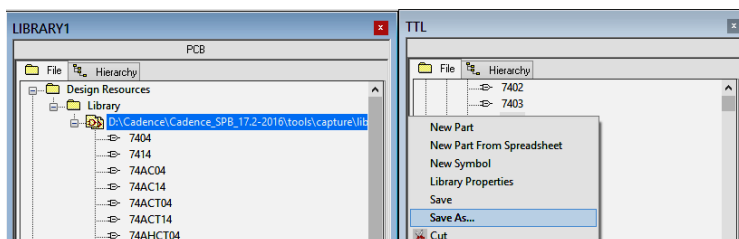


Fig. 2.2.3 – Renaming and saving the library

The design of hierarchical symbols can be easily performed in compliance with all the requirements of the standards, which is an important advantage of the OrCAD package. As an example, let's draw a scheme of substitution of a hierarchical symbol, having chosen for a multiplexer on four inputs. Let's create a new project *File / New / Project*. Connect your own library *new_library.olb* (Fig. 2.2.4) to the project and create a multiplexer circuit in it (Fig. 2.2.5).

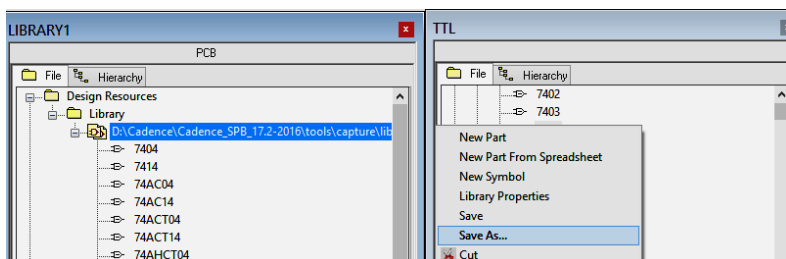


Fig. 2.2.4 – Connecting your own library

Let's call the default *schematic1* directory by another name, such as *multip*. Save the project and close the working window with the replacement scheme. Now open the library *new_library.olb* and copy the directory *multip*

(Fig. 2.2.6), with the desired directory in the file structure of the library, and in the library cache (Library Cache) – the corresponding components included in the substitution scheme (Fig. 2.2.7).

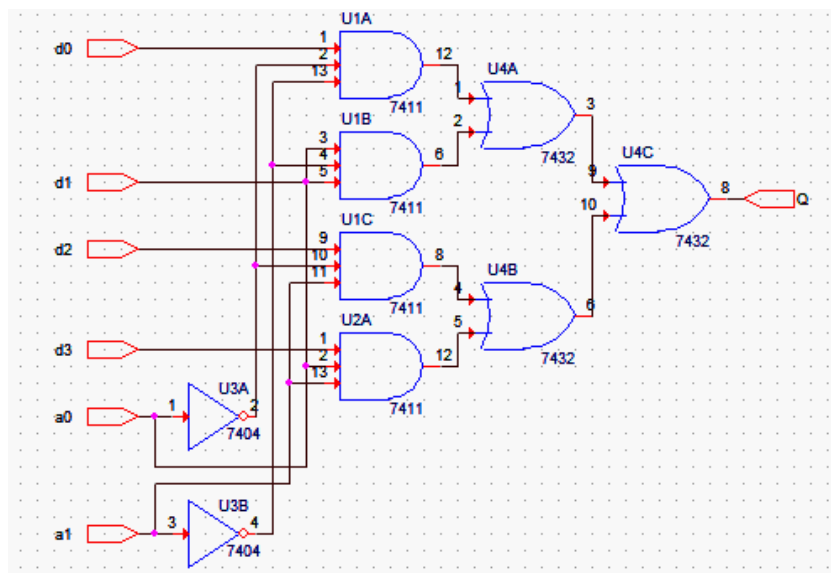


Fig. 2.2.5 – Multiplexer scheme

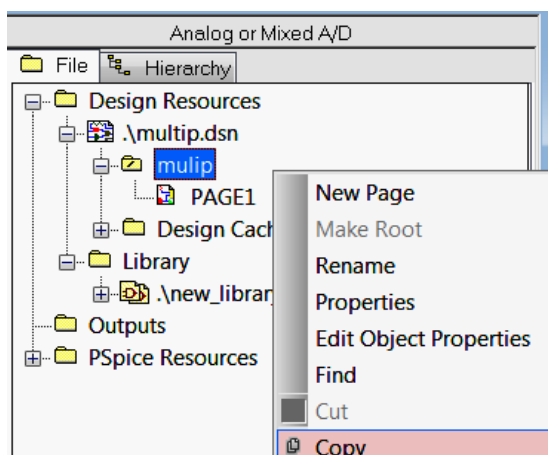


Fig. 2.2.6 – The process of copying the multiplexer circuit to your own library

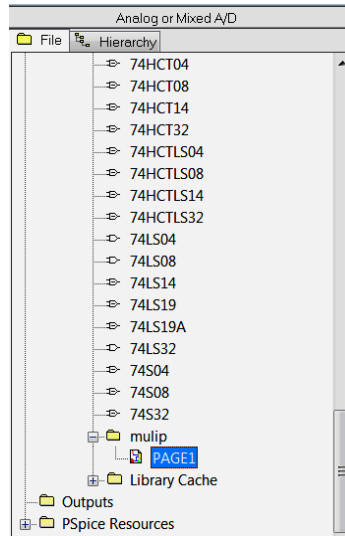


Fig. 2.2.7 – View of your own library

Next you need to create an external description of the hierarchical symbol. Right-click on the catalog with the name of library and execute the command *New Part* (Fig. 2.2.8).

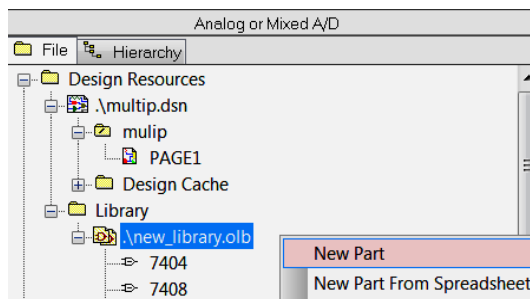


Fig. 2.2.8 – Creating a *New Part*

The *New Part Properties* panel opens. Enter the name of the MULT symbol in the *Name* field and make a reference to the hierarchical symbol substitution scheme (Fig. 2.2.9).

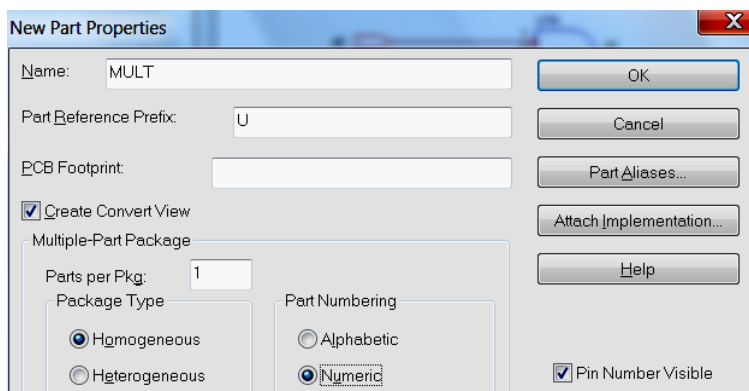


Fig. 2.2.9 – *New Part* properties window

The *PCB Footprint* field specifies the type of case in which the real component is placed. For an abstract symbol, such information does not make sense, so leave it blank. Select the *Create Convert View* check box to show that there is a logical equivalent notation for the projected element. Click the *Attach Implementation* button. In our case, this is a substitution scheme stored in the multip directory.

A dialog box with the same name will open. In the upper field *Implementation Type* enter the type of internal description: *Schematic View* – substitution scheme, in the next field *Implementation* – the name of the directory where it is stored, the lower field is left blank, because the substitution scheme is in the same library as and the symbol (Fig. 2.2.10).

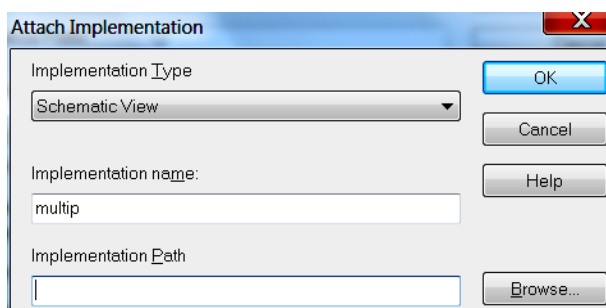


Fig. 2.2.10 – Implementation connection

In the next window *Parts per Pkg* you need to set the number of elements that are placed in the chip. Let's leave here the number "1", given by default. Note that when you create a component in the *Package Type* section, you can change its type and select homogeneous or inhomogeneous characters. In the first case, various parts can be placed in the case, but this is quite rare and is not even supported by some CAD. In the *Part Numbering* section (character numbering method – alphanumeric or numeric) select *Numeric*.

Click *OK*, the edit window appears. The dotted line indicates the boundaries of the designation (not including inputs and outputs). To draw GS lines, we need to select *Place / Line /* on the top toolbar.

In this menu it is also possible to draw right angles, ellipses, arcs, elliptical arcs, curved lines, squares and rectangles, closed curved lines. For example, let's draw a rhombus. It should be noted that when creating any form of GS, its selection area will always be a quadrilateral (dotted line). Further addition of inputs and outputs occurs to this area, they cannot be visually attached to a rhombus (in our example). But the inscription on the pins can be moved anywhere (Fig. 2.2.11).

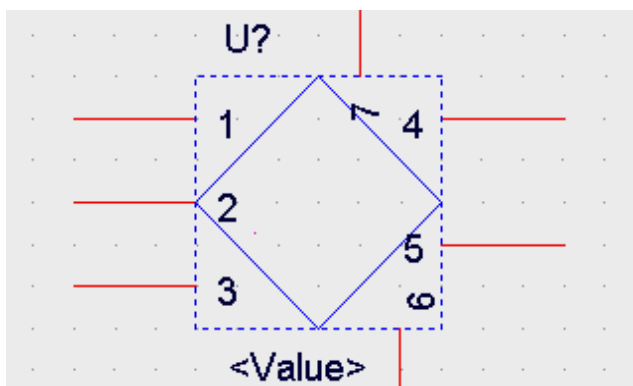


Fig. 2.2.11 – Example of connected pins

Draw a multiplexer in the form of a rectangle, and using the menu command *Place / Text /* get GS, as shown in Fig. 2.2.12. Now add contacts (inputs/outputs). Unlike lines and text, which are only graphical

representations in a diagram, contacts are used to connect components and display GS. To do this, select the *Place Pin* tool (Fig. 2.2.13).

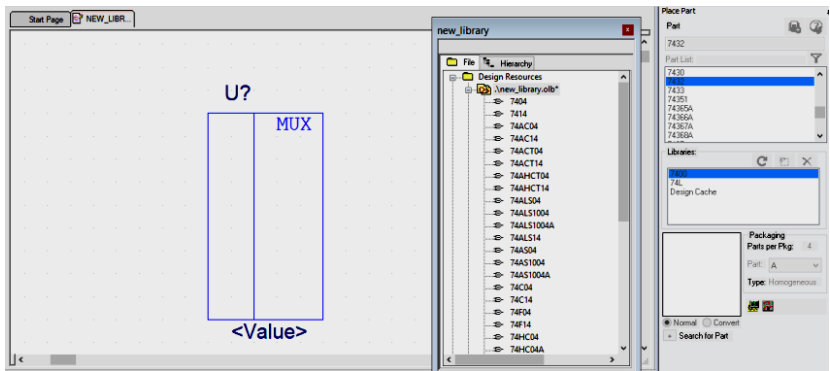


Fig. 2.2.12 – GS of multiplexer

It specifies the name of the contact, the contact number in the case (only relevant for real components), the contact form and its type. The *User Properties* button is used to set and edit parameters (it is enough to specify the name and type).

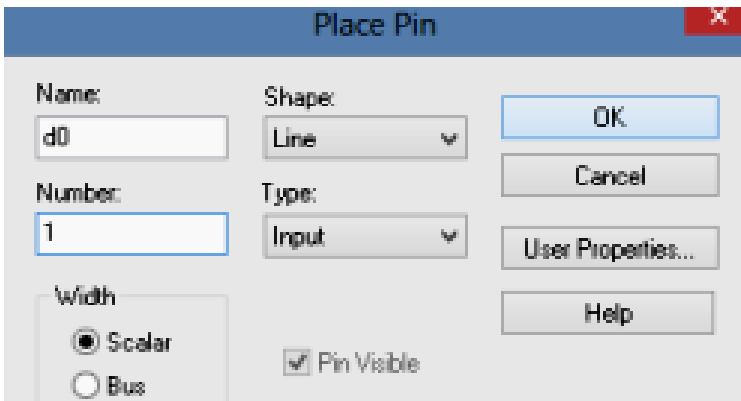


Fig. 2.2.13 – Window *Place Pin*

After placing all the contacts we get GS, as shown in the Fig. 2.2.14. Let's save the library and close it. Now let's create a new project *mult.opj*, connect to it a personal library *new_library.olb* as shown in Fig. 2.2.15.

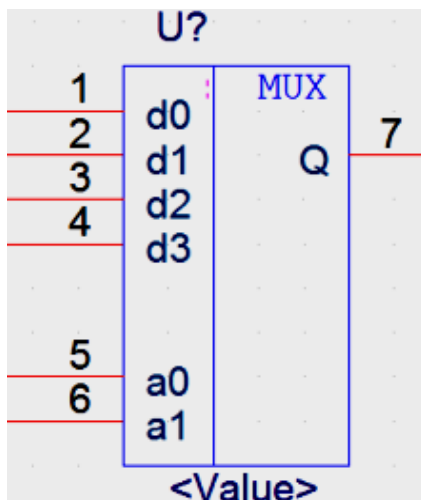


Fig. 2.2.14 – GS of multiplexer

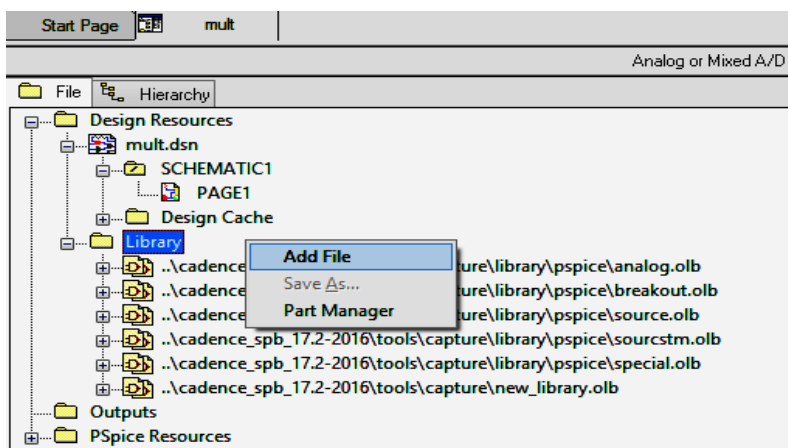


Fig. 2.2.15 – Connecting your own library

Place one copy of the hierarchical *MULT* symbol (Fig. 2.2.16) in the graphics editor window and tell the OrCAD *Capture* editor that the *MULT* symbol is not a primitive but a hierarchical symbol. To do this, double-click on it left-click and when the *Property Editor* window opens, change the value of the *Primitive* property from *default* (by default the character is considered a primitive) to *NO* (Fig. 2.2.17).

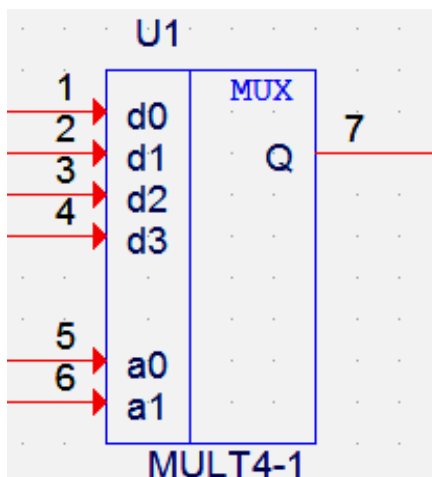


Fig. 2.2.16 – Placing the multiplexer on the worksheet

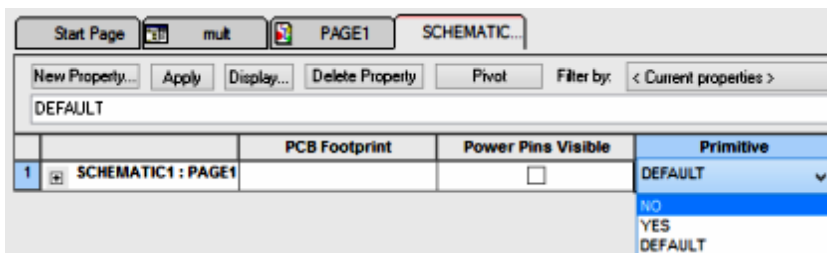


Fig. 2.2.17 – Specify the properties of the multiplexer

Make sure that the symbol now has a subordinate scheme. To do this, simply execute the command *View / Descend Hierarchy*, while the editor must reduce the hierarchy of the description and show the substitution scheme of the

symbol *MULT* (Fig. 2.2.18).

It remains to simulate a scheme that contains only one hierarchical symbol. To do this, connect signal generators to the input of the multiplexer (Fig. 2.2.19). The results of modeling the scheme indicate the correct creation of the component and the correctness of its operation. At this stage, the so-called abstract component is obtained, which can be used in the development and modeling of schemes.

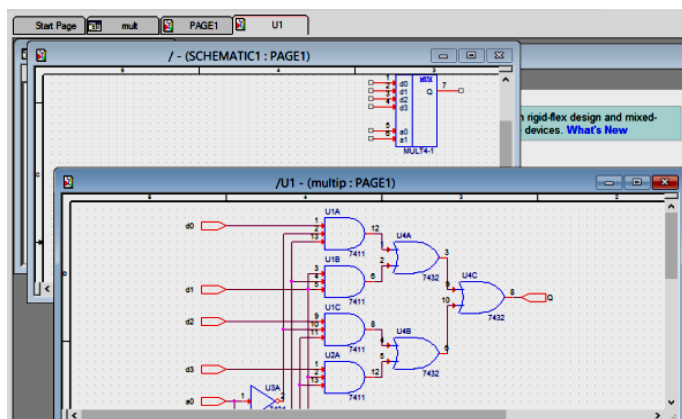


Fig. 2.2.18 – GS of multiplexer and its slave circuit

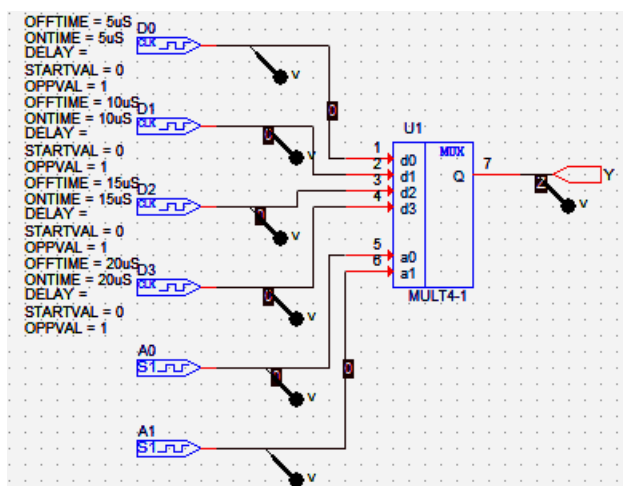


Fig. 2.2.19 – Multiplexer circuit with connected generators input signals

Tasks for practical work

Create your own element (Table 2.2) in the OrCAD.

Table 2.2

№	The form of a logical element	A function performed by an element
1	rectangle	$F = A + \bar{B} + C + D$
2	square	$F = A + B + A \cdot B \cdot C \cdot D$
3	parallelogram	$F = A + B + A \cdot \bar{C} \cdot D$
4	triangle	$F = \bar{A} \cdot \bar{B} \cdot \bar{C} \cdot D$
5	circle	$F = (A + B) \cdot \bar{C} \cdot \bar{D}$
6	star	$F = A \cdot B + \bar{C}$
7	rhomb	$F = (A + B) \cdot \bar{C}$
8	rectangle	$F = \overline{(A \cdot B \cdot C)} \cdot A$
9	square	$F = A \cdot \bar{B} \cdot \bar{C}$
10	parallelogram	$F = A + B + \bar{C} \cdot D$
11	triangle	$F = A \cdot B \cdot \bar{C} \cdot D$
12	square	$F = A \cdot \bar{B} \cdot \bar{C} + A$
13	star	$F = A \cdot B + C + D$
14	rhomb	$F = A + B \cdot C + A \cdot B \cdot C$
15	parallelogram	$F = A + B \cdot C \cdot D + A \cdot \bar{C}$

Practical work 3

PROGRAMMING A MACRO ELEMENT

The purpose of the work: Perform macro element programming in OrCAD.

For *PSPice*, the simulation task is entered in text files. Knowledge of their formats when graphically entering the scheme is desirable, because it allows you to create templates of new symbols of components, as well as allows you to compose text descriptions of macromodels and facilitates the search for errors when setting up the scheme. Graphical input of schemas creates three task files with the same name: *.NET (connection table), *.ALS (list of circuit connections to component pins) and *.CIR (list of modeling directives). When modeling, the *.CIR file is loaded directly into *PSPice*, which contains links to other files. To assemble it manually, the schematic diagram of the simulated device is first drawn and the names of all its nodes are assigned. The standard designation of nodes is given in Table 2.3.

Table 2.3 – Designation of nodes

Node name	High-voltage/ level	Description
0	0 B	Analog "ground"
\$ G_CD4000_VDD	5 B	Power supply CMOS integrated circuit
\$ G_CD4000_VSS	0 B	Power supply CMOS integrated circuit
\$ G_DPWR	5 B	Power supply TTL integrated circuit
\$ G_DGND	0 B	Power supply TTL integrated circuit
\$ D_HI	"1"	Logical "1"
\$ D_LO	"0"	Logical "0"
\$ D_X	"X"	Uncertain logical state
\$ D_NC	–	The output of the digital component is not connected to the circuit

The \$ D_NC node is used to indicate unused pins (NC means Not Connected). When referring to digital node names, parentheses are used, for

example V (6) is the potential of node 6. Node names in the form of alphanumeric characters with references to them are taken in square brackets [] to distinguish them from component names. For example, the potential of the node IN is denoted as V ([IN]), and V (I2) is the voltage at the current source I2. There is an agreement in the program that all nodes whose names begin with the characters \$ G_ are *global*, such as the node \$ G_POS. Global nodes are used in circuits that have macromodels. After naming the nodes, a simulation task is created, which is entered into a file with the extension *.CIR, in which *the first line of the file* is the header line (displayed as a header in the source file), *comment lines* contain the symbol "*" in the first position (the end of the line after the sign ";" is perceived as a comment), *the last line of the file* has a .END command, and the *continuation line* begins with the "+" character in the first position (maximum line length 132 characters). Spaces, commas, or equals are equivalent (*PSpice* does not distinguish between uppercase and lowercase letters).

It makes sense to design individual fragments of the component substitution scheme in the form of a macromodel, the description of which begins with the .SUBCKT directive and ends with the .ENDS directive. Between them are descriptions of the components:

```
.SUBCKT <macromodel name> <list of nodes>
+ [OPTIONAL: << interface node > = <default value>> *]
+ [PARAMS: <parameter name> = <value> *]
+ [TEXT: <<the name of the text variable> – < text >> *]
{Description of components}
.ENDS [macromodel name]
```

The OPTIONAL keyword is used to specify one or more optional macromodel nodes – the node name and its default value are specified. If these nodes are not specified when the macromodel is called, their default values are used. The PARAMS keyword is followed by a list of parameters whose values are passed from the main circle to the macromodel. After the keyword TEXT – a text variable that is passed from the description of the main chain to the description of the macromodel, which is used only in the simulation of digital devices. You can place a description of other macromodels and other directives between the .SUBCKT and .ENDS directives.

High-level simulation of the adder

The task of high-level modeling is not to reveal the structure of the object and simulate its operation, such a tool should simulate not only the function but also the time relationships in the scheme. The main idea of this method is the following – to present a model of a digital device in the form of two parts, the first of which (logic block) should simulate the logic of operation with zero delays, and the second (delay block) should simulate real delays. A typical macromodel is a list of connections, a description of a component at a lower level, where it is represented as a structure (functional or schematic diagram). The logic-time model does not have a list of connections, but contains two or three (LOGICEXP, PINDLY and CONSTRAINT) primitives.

```
Uxxx LOGICEXP (<number of inputs>, <number of outputs>)
+ <power> <earth>
+ <input_node_1> ... <input_node_m>
+ <source_node_1> ... <source_node_n>
+ <model_name> < model_name_input/output>
+ [IO_LEVEL=<interface_model_level>]
+ [MNTYMXDLY=<delay_selection_selection>]
+ LOGIC:
+ <logical_value>
```

The logical assignment may include logical expressions for intermediate variables and source nodes in the following form:

```
<output_node_name> = {<logical expression>}
<intermediate_variable> = {<logical expression>}
```

Each mentioned source node must have only one logical expression, and intermediate variables, which are assigned once, can be used in subsequent expressions repeatedly. Consider what this primitive looks like for the ADDER. Functional model of the ADDER with zero delays:

```
.SUBCKT Adder_log C0 A1 B1 A2 B2 SUM1 SUM2 C2
; macromodel start directive
+ OPTIONAL: DPWR = $G_DPWR DGND = $G_DGND
; power and ground contacts
```

```

+ PARAMS: MNTYMXDLY=0 IO_LEVEL=0
; initialization of parameters
U82LOG LOGICEXP(5,8) DPWR DGND
; the primitive name must start with the letter "U"
+ C0 A1 B1 A2 B2 SUM1 SUM2 C2
; inputs and outputs of the logic unit
+ D0_GATE IO_STD IO_LEVEL={IO_LEVEL}
; reference to the standard input / output model
+ LOGIC:
; the beginning of the logical section
+ C0A1B1 = { ~( (C0 & A1) | (C0 & B1) | (A1 & B1) ) }
; logical expression for the source node of the macromodel
.ENDS Adder_log
; directive to complete the macromodel

```

Add to this macromodel the PINDLY primitive, which reproduces delays. With its addition it is necessary to make changes in the LOGICEXP logical block. In this case, the outputs C0_I A1_I B1_I A2_I B2_I is a direct translation of the input signals C0 A1 B1 A2 B2. They do not undergo any logical transformations (Buffering) and are used in the PINDLY block as internal (additional) nodes. SUM1_O signals SUM2_O C2_O are LOGICEXP outputs and PINDLY inputs. For these signals in the delay unit are created paths through which the signal receives the required delay. Signals cannot be input to the delay unit directly from external inputs, so they are transmitted through the logic unit even if they do not undergo logic transformations. The delay of the output signal depends on which input a specific event (switching) comes from. Changes in states at the inputs are controlled by the following functions:

```

CHANGED (<node>, <time_interval>)
CHANGED_LH (<node>, <time_interval>)
CHANGED_HL (<node>, <time_interval>).

```

The CHANGED function returns TRUE if the node changed its condition over time. The CHANGED_LH function controls the front, and the CHANGED_HL function controls the signal drop at the node. The TRN_LH

and TRN_HL functions monitor the change of output nodes for which delays are calculated. Instead of L (low level) and H (high level) levels, you can also use Z (high impedance output) and \$ (any level). These functions have no arguments and at this point in time simply record changes in the states of the source nodes. Functional adder macromodel (ADDER) taking into account delays:

```
.SUBCKT Adder_log_dly C0 A1 B1 A2 B2 SUM1 SUM2 C2
; new macromodel name
U82DLY PINDLY(3,0,5) DPWR DGND
; the beginning of the primitive "PINDLY"
+ SUM1 SUM2 C2
; logic block outputs
+ C0 A1 B1 A2 B2
; inputs on which the delay depends
+ SUM1_O SUM2_O C2_O
; nodes for which the delay is calculated
+ IO_STD MNTYMXDLY={MNTYMXDLY} IO_LEVEL={IO_LEVEL}
; standard input / output model
BOOLEAN:
+ ANY_CH_AB = { CHANGED(A2,0) | CHANGED(B2,0) |
CHANGED(A1,0) | CHANGED(B1,0) }
; the section defines the intermediate variables that will be
; used to calculate the delay
+ PINDLY:
+ SUM1_O = {
; the operator calculates a specific delay
+ CASE(
+ CHANGED(C0,0) & TRN_HL, DELAY(-1,-1,40NS),
+ ANY_CH_AB & TRN_LH, DELAY(-1,-1,40NS),
+ ANY_CH_AB & TRN_HL, DELAY(-1,-1,35NS),
+ CHANGED(C0,0) & TRN_LH, DELAY(-1,-1,34NS),
+ DELAY(-1,-1,41NS); DEFAULT
+ )
+ }
```

```

+ SUM2_O = {
+   CASE(
+     ANY_CH_AB & TRN_LH, DELAY(-1,-1,40NS),
+     CHANGED(C0,0) & TRN_HL, DELAY(-1,-1,42NS),
+     CHANGED(C0,0) & TRN_LH, DELAY(-1,-1,38NS),
+     ANY_CH_AB & TRN_HL, DELAY(-1,-1,35NS),
+     DELAY(-1,-1,43NS); DEFAULT
+   )
+ }
+ C2_O = {
+   CASE(
+     ANY_CH_AB & TRN_LH, DELAY(-1,-1,40NS),
+     ANY_CH_AB & TRN_HL, DELAY(-1,-1,35NS),
+     CHANGED(C0,0) & TRN_HL, DELAY(-1,17NS,27NS),
+     CHANGED(C0,0) & TRN_LH, DELAY(-1,12NS,19NS),
+     DELAY(-1,18NS,41NS) ; DEFAULT
+   )
+ }
.ENDS Adder_log_dly

```

In PINDLY, delay expressions that include the CASE selection statement can be used. It checks the conditional expressions in the order in which they appear until the first true expression is detected, after which the delay is assigned a value from the string that matches the fulfilled condition. The DELAY operator (<min>, <typ>, <max>) specifies the minimum, default, and maximum delays, respectively.

Example of creating an inverter

After getting acquainted with the structure and description of the macromodel and several primitives (LOGICEXP and PINDLY) we will create our own inverter. The logic of this element is simple: receiving a signal at the input – at the output we get a signal opposite in value.

Functional macromodel of the inverter taking into account delays:

```

.SUBCKT Example X1_I Y_O
+ OPTIONAL: DPWR = $G_DPWR DGND = $G_DGND

```

```

+ PARAMS: MNTYMXDLY=0 IO_LEVEL=0
U82LOG LOGICEXP(1,2) DPWR DGND
+ X1_I
+ X1 Y
+ D0_GATE IO_STD IO_LEVEL={IO_LEVEL}
+
+ LOGIC:
+ X1 = {X1_I}
+ Y = {(~X1)}
U82DLY PINDLY(1,0,1) DPWR DGND
+ Y
+ X1
+ Y_O
+ IO_STD MNTYMXDLY={MNTYMXDLY} IO_LEVEL={IO_LEVEL}
+
+ BOOLEAN:
+ ANY_CH_AB = { CHANGED(X1,0) }
+
+ PINDLY:
+ Y_O = {
+ CASE(
+ ANY_CH_AB & TRN_LH, DELAY(-1,-1,40NS),
+ ANY_CH_AB & TRN_HL, DELAY(-1,-1,40NS),
+ CHANGED(X1,0) & TRN_HL,DELAY(-1,17NS,27NS),
+ CHANGED(X1,0) & TRN_LH,DELAY(-1,12NS,19NS),
+ DELAY(-1,18NS,41NS);DEFAULT
+ )
+ }
.ENDS

```

When adding the created element to the project it is necessary to specify a path to the file with the description (example.lib) in *Simulation Profile – Configuration Files – Library*. Otherwise, an error will occur indicating that the "subcircuit" of the element is not defined.

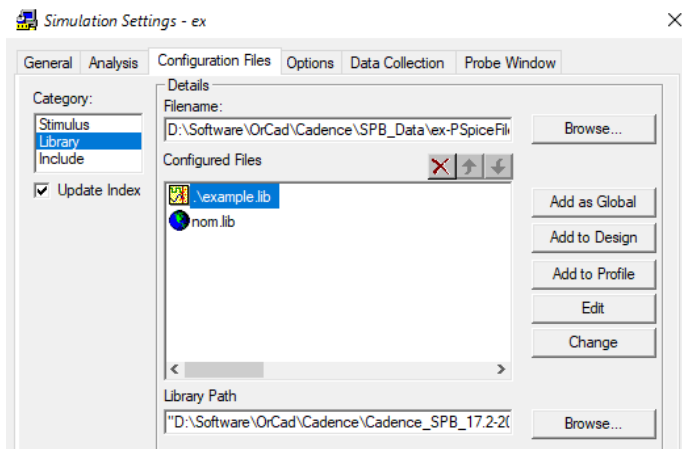


Fig. 2.3.1 – Simulation profile settings window

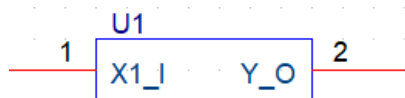


Fig. 2.3.2 – Created item

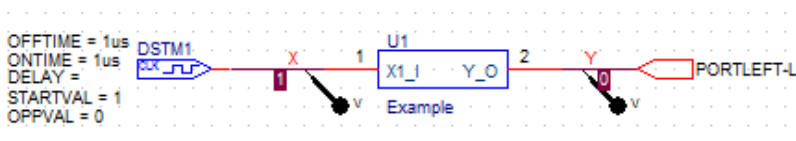


Fig. 2.3.3 – The assembled scheme

The results of delayed simulation are shown in Fig. 2.3.4.

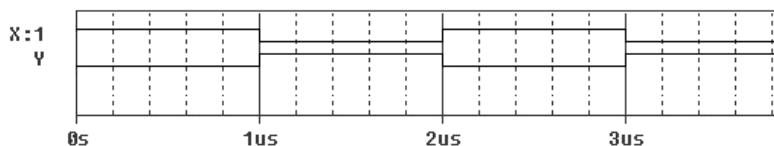


Fig. 2.3.4 – Simulation results

Creating an adder

Using the information above, create a file with a description (adder.lib) of the element in SPICE-format with the following content:

* 2-BIT BINARY FULL ADDER

```
.SUBCKT Adder C0_I A1_I B1_I A2_I B2_I SUM1_O SUM2_O C2_O
+ OPTIONAL: DPWR = $G_DPWR DGND = $G_DGND
+ PARAMS: MNTYMXDLY=0 IO_LEVEL=0
U82LOG LOGICEXP(5,8) DPWR DGND
+ C0_I A1_I B1_I A2_I B2_I
+ C0 A1 B1 A2 B2 SUM1 SUM2 C2
+ D0_GATE IO_STD IO_LEVEL={IO_LEVEL}
+
+ LOGIC:
+ C0 = { C0_I }
+ A1 = { A1_I }
+ B1 = { B1_I }
+ A2 = { A2_I }
+ B2 = { B2_I }
+ A2BAR = { ~A2 }
+ B2BAR = { ~B2 }
+
+ C0A1B1 = { ~( (C0 & A1) | (C0 & B1) | (A1 & B1) ) }
+ SUM1 = { (C0 & C0A1B1) | (A1 & C0A1B1) | (B1 & C0A1B1) | (C0 & A1
& B1) }
+ C2 = { ~( ( C0A1B1 & A2BAR) | (C0A1B1 & B2BAR) | (A2BAR &
B2BAR) ) }
+ SUM2 = { ~( (C0A1B1 & C2) | (C2 & A2BAR) | (C2 & B2BAR) |
(A2BAR & B2BAR & C0A1B1) ) }
U82DLY PINDLY(3,0,5) DPWR DGND
+ SUM1 SUM2 C2
+ C0 A1 B1 A2 B2
+ SUM1_O SUM2_O C2_O
+ IO_STD MNTYMXDLY={MNTYMXDLY} IO_LEVEL={IO_LEVEL}
+
+ BOOLEAN:
+ ANY_CH_AB = { CHANGED(A2,0) | CHANGED(B2,0) |
CHANGED(A1,0) | CHANGED(B1,0) }
+
```

```

+ PINDLY:
+ SUM1_O = {
+ CASE(
+ CHANGED(C0,0) & TRN_HL, DELAY(-1,-1,40NS),
+ ANY_CH_AB & TRN_LH, DELAY(-1,-1,40NS),
+ ANY_CH_AB & TRN_HL, DELAY(-1,-1,35NS),
+ CHANGED(C0,0) & TRN_LH, DELAY(-1,-1,34NS),
+ DELAY(-1,-1,41NS) ; DEFAULT
+ )
+ }
+ SUM2_O = {
+ CASE(
+ ANY_CH_AB & TRN_LH, DELAY(-1,-1,40NS),
+ CHANGED(C0,0) & TRN_HL, DELAY(-1,-1,42NS),
+ CHANGED(C0,0) & TRN_LH, DELAY(-1,-1,38NS),
+ ANY_CH_AB & TRN_HL, DELAY(-1,-1,35NS),
+ DELAY(-1,-1,43NS) ; DEFAULT
+ )
+ }
+ C2_O = {
+ CASE(
+ ANY_CH_AB & TRN_LH, DELAY(-1,-1,40NS),
+ ANY_CH_AB & TRN_HL, DELAY(-1,-1,35NS),
+ CHANGED(C0,0) & TRN_HL, DELAY(-1,17NS,27NS),
+ CHANGED(C0,0) & TRN_LH, DELAY(-1,12NS,19NS),
+ DELAY(-1,18NS,41NS) ; DEFAULT
+ )
+ }
.ENDS

```

In the OrCAD package, in the *File* menu item, execute *New – Library* and enter the name. Then, when the active project manager is running *Tools – Generate Part*, specify the file we created *adder.lib* and type (*PSpice model library*), the name of the library in which it will be placed. A symbolic description of the element will be generated (Fig. 2.3.5). Edit it (change the

size of the case, contact parameters) and create a project for modeling (Fig. 2.3.6). The results of delayed simulation are shown in Fig. 2.3.7.

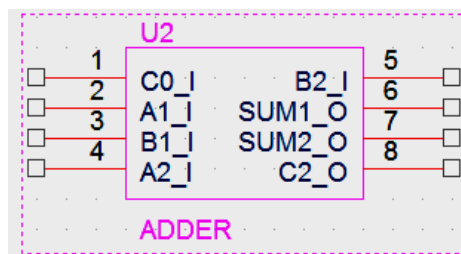


Fig. 2.3.5 – Created item

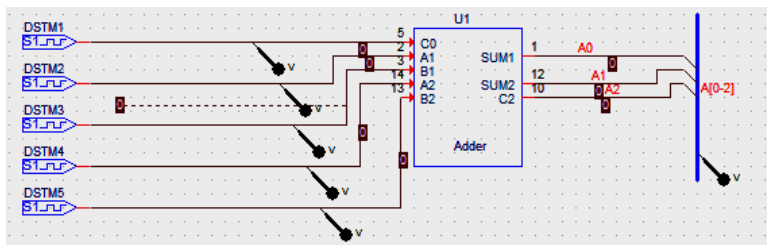


Fig. 2.3.6 – The assembled scheme

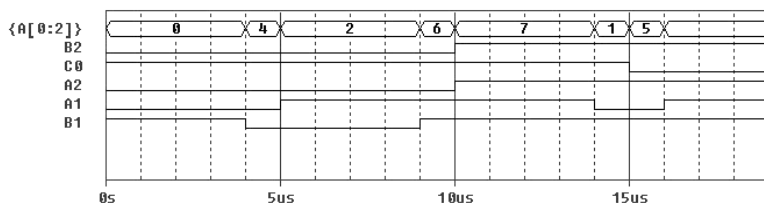


Fig. 2.3.7 – Simulation results

Creating a multiplexer

Using the information above, create a file with a description (mux_16_1.lib) of the element in SPICE-format with the following content:

```
.SUBCKT 74150 Enable A B C D E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 E10 E11
E12 E13 E14 E15 W
; designation of names of inputs and outputs of an element
+ OPTIONAL: DPWR=$G_DPWR DGND=$G_DGND
```

```

+ PARAMS: MNTYMXDLY=0 IO_LEVEL=0
*
UMUXLOG LOGICEXP(21,22) DPWR DGND
; specify the number of contacts of the element
+ Enable A B C D E0 E1 E2 E3 E4 E5 E6 E7 E8 E9 E10 E11 E12 E13 E14
E15
*
+ GBAR_I A_I B_I C_I D_I E0_I E1_I E2_I E3_I E4_I E5_I E6_I E7_I E8_I
E9_I
+ E10_I E11_I E12_I E13_I E14_I E15_I W_O O
; _I – entrance designation, _O – exit designation
*
+ D0_GATE IO_STD IO_LEVEL={IO_LEVEL}
+ LOGIC:
+ GBAR_I = { Enable }
+ A_I = { A } ; assigning input A to the name A_I
+ B_I = { B }
+ C_I = { C }
+ D_I = { D }
+ E0_I = { E0 }
+ E1_I = { E1 }
+ E2_I = { E2 }
+ E3_I = { E3 }
+ E4_I = { E4 }
+ E5_I = { E5 }
+ E6_I = { E6 }
+ E7_I = { E7 }
+ E8_I = { E8 }
+ E9_I = { E9 }
+ E10_I = { E10 }
+ E11_I = { E11 }
+ E12_I = { E12 }
+ E13_I = { E13 }
+ E14_I = { E14 }
+ E15_I = { E15 }

```

```

+ IA = { ~A_I } ; IA has a value that is inverted from A_I
+ IB = { ~B_I }
+ IC = { ~C_I }
+ ID = { ~D_I }
+ IG = { GBAR_I }
+ IE0 = { E0 & IA & IB & IC & ID & IG }; logical "AND"
+ IE1 = { E1 & A_I & IB & IC & ID & IG }
+ IE2 = { E2 & IA & B_I & IC & ID & IG }
+ IE3 = { E3 & A_I & B_I & IC & ID & IG }
+ IE4 = { E4 & IA & IB & C_I & ID & IG }
+ IE5 = { E5 & A_I & IB & C_I & ID & IG }
+ IE6 = { E6 & IA & B_I & C_I & ID & IG } ; logical "OR"
+ IE7 = { E7 & A_I & B_I & C_I & ID & IG }
+ IE8 = { E8 & IA & IB & IC & D_I & IG }
+ IE9 = { E9 & A_I & IB & IC & D_I & IG }
+ IE10 = { E10 & IA & B_I & IC & D_I & IG }
+ IE11 = { E11 & A_I & B_I & IC & D_I & IG }
+ IE12 = { E12 & IA & IB & C_I & D_I & IG }
+ IE13 = { E13 & A_I & IB & C_I & D_I & IG }
+ IE14 = { E14 & IA & B_I & C_I & D_I & IG }
+ IE15 = { E15 & A_I & B_I & C_I & D_I & IG }
+ W_O = { (IE0 | IE1 | IE2 | IE3 | IE4 | IE5 | IE6 | IE7 | IE8 |
+ IE9 | IE10 | IE11 | IE12 | IE13 | IE14 | IE15) }
* nW_O = { ~W }
*
UMUXDLY PINDLY (1,0,21) DPWR DGND
+ W_O
*
+ GBAR_I A_I B_I C_I D_I E0_I E1_I E2_I E3_I E4_I E5_I E6_I E7_I E8_I
E9_I
+ E10_I E11_I E12_I E13_I E14_I E15_I
*
+ W
+ IO_STD MNTYMXDLY={MNTYMXDLY} IO_LEVEL={IO_LEVEL}
+ BOOLEAN:

```

```

+ DATA = { CHANGED(E0_I,0) | CHANGED(E1_I,0) | CHANGED(E2_I,0)
| CHANGED(E3_I,0) |
+ CHANGED(E4_I,0) | CHANGED(E5_I,0) | CHANGED(E6_I,0) |
CHANGED(E7_I,0) |
+ CHANGED(E8_I,0) | CHANGED(E9_I,0) | CHANGED(E10_I,0) |
CHANGED(E11_I,0) |
+ CHANGED(E12_I,0) | CHANGED(E13_I,0) | CHANGED(E14_I,0) |
CHANGED(E15_I,0) }
+ SELECT = { CHANGED(A_I,0) | CHANGED(B_I,0) | CHANGED(C_I,0) |
CHANGED(D_I,0) }
+ PINDLY:
+ W = {
+ CASE(
+ SELECT & TRN_LH, DELAY(-1,23NS,35NS),
+ SELECT & TRN_HL, DELAY(-1,22NS,33NS),
+ CHANGED(GBAR_I,0) & TRN_HL, DELAY(-1,21NS,30NS),
+ CHANGED(GBAR_I,0) & TRN_LH, DELAY(-1,15.5NS,24NS),
+ DATA & TRN_HL, DELAY(-1,13NS,20NS),
+ DATA & TRN_LH, DELAY(-1,8.5NS,14NS),
+ DELAY(-1,24NS,36NS)
+ )
+ }
* nW = {
* CASE(
* SELECT, DELAY(-1,26NS,38NS),
* CHANGED(GBAR_I,0) & TRN_HL, DELAY(-1,27NS,36NS),
* CHANGED(GBAR_I,0) & TRN_LH, DELAY(-1,21.5NS,30NS),
* DATA, DELAY(-1,18NS,25NS),
* DELAY(-1,29NS,41NS)
* )
* }
; setting a delay for an item
*
.ENDS
*$

```

We start OrCAD, in the File menu item we execute *New – Library*, we enter the name, then we execute *Tools – Generate Part*, we specify the model file mux_16_1.lib, created by us, file type (*PSpice model library*), the name of library in which it will be placed. A symbolic description of the element is generated (Fig. 2.3.8). Edit it (change the size of the case, contact parameters) and create a project for modeling (Fig. 2.3.9). The results of delayed simulation are shown in Fig. 2.3.10.

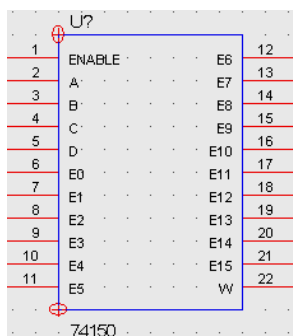


Fig. 2.3.8 – GS of element

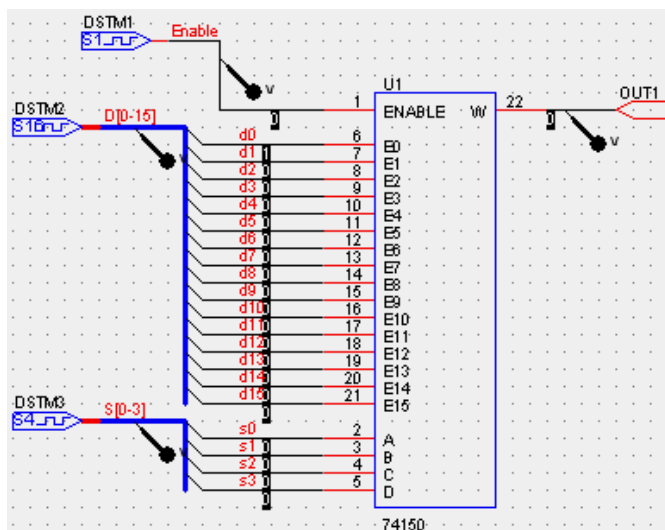


Fig. 2.3.9 – The assembled scheme

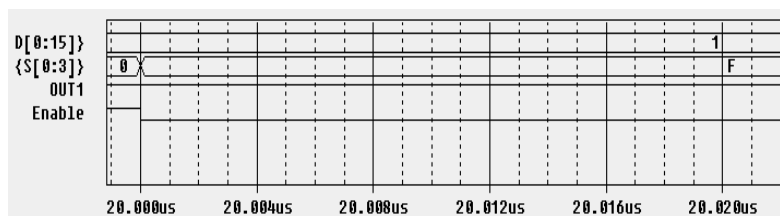


Fig. 2.3.10 – Simulation results

Tasks for practical work

Create your own element (Table 2.4) in the OrCAD.

Table 2.4

Nº	Logical expression
1	$F = \overline{X1} \cdot \overline{X2} \cdot X3 + \overline{X1} \cdot X2 \cdot X3 + X1 \cdot X2 \cdot \overline{X3}$
2	$F = X1 \cdot \overline{X2} + \overline{X1} \cdot X2$
3	$F = \overline{X1} \cdot \overline{X2} + \overline{X1} \cdot X2 + \overline{X3}$
4	$F = X1 \cdot X2 + X1 \cdot X3 + X2 \cdot X3$
5	$F = \overline{X1} \cdot X3 + X1 \cdot X2 \cdot \overline{X3}$
6	$F = X1 \cdot \overline{X2} \cdot X3 + X1 \cdot X2 \cdot \overline{X3} + X1 \cdot X2 \cdot X3$
7	$F = \overline{X1} \cdot \overline{X2} \cdot X3 + \overline{X1} \cdot X2 \cdot \overline{X3} + X1 \cdot \overline{X2} \cdot \overline{X3} + X1$
8	$F = X1 \cdot \overline{X2} \cdot X3 + \overline{X1} \cdot X2 \cdot X4$
9	$F = \overline{X1} \cdot \overline{X2} \cdot X3 + \overline{X1} \cdot X2$
10	$F = \overline{X1} \cdot \overline{X2} \cdot X3 + \overline{X2} + X1 \cdot X2$
11	$F = \overline{(X1 \cdot X2 \cdot X3)} + \overline{(X1 \cdot X2)}$
12	$F = \overline{(X1 \cdot X2 \cdot X3)} + \overline{(X1 \cdot X2)} + \overline{X1} \cdot X2$
13	$F = \overline{(X1 \cdot X2 \cdot X3)} + \overline{X1} + X2$
14	$F = \overline{(X1 \cdot X2)} + \overline{X2} + X2 \cdot X3 \cdot X4$
15	$F = \overline{(X1 + X2)} + \overline{(X1 + X3)}$

Practical work 4

USING THE PSPICE PROGRAM OF THE ORCAD SYSTEM FOR ANALOG MODELING OF DIGITAL BLOCKS

The purpose of the work: Perform analog device simulation using PSpice.

In this practical work for modeling of schemes the program *PSpice* will be used (*Personal Simulation Program with Integrated Circuit Emphasis*) – analog and digital logic simulation program described in SPICE, which is designed for personal computers. *PSpice* was the first version of SPICE, later added to the program for viewing and analyzing oscillograms (*Probe*). Subsequent versions improved performance and expanded the list of supported platforms. *PSpice* calculates the following characteristics of electronic circuits: DC circuit mode in the "operating point" (*Bias Point*); low-signal transfer functions in the mode on a direct current (*Transfer Function*); characteristics of the linearized circuit in the frequency domain under the influence of one or more signals (*AC Sweep*); transients under the influence of signals of different shapes (*Transient Analysis*); spectral analysis (*Fourier Analysis*); "statistical tests" by the Monte Carlo method and calculation of the worst case (*Monte Carlo / Worst Case*); multivariate analysis with temperature variation (*Temperature*) and a number of other parameters. Parametric optimization is performed using the *PSpice Optimizer* module. Each type of calculation corresponds to a specific directive. Their full list is given in Table 2.5.

Table 2.5 – Modeling directives

Name	Purpose
	Calculation of standard characteristics
.AC	Calculation of frequency characteristics
.DC	Calculation of a mode on a direct current
.FOUR	Spectral analysis
.NOISE	Calculation of the internal noise level
.OP	Transfer to the output file of the parameters of the scheme, which is linearized near the operating point
.SENS	Calculation of low-signal sensitivity in DC mode

Results delivery management	
.PLOT	Presentation of calculation results in the source file in the form of graphs constructed in text mode
.PRINT	Presentation of calculation results in the source file in the form of tables
.PROBE	Data transfer to the graphics postprocessor Probe
.VECTOR	Creating a file with the results of modeling digital devices
.WATCH	Issuance of intermediate analysis results on the screen of the program <i>PSpice</i> in text form
.WIDTH	Assign line length to the source file
Multivariate analysis	
.STEP	Variation of parameters
.TEMP	Assignment of ambient temperature
Auxiliary files, definition of functions and parameters	
.END	End of task
.FUNC	Function definition
.INC	Include another file in the input file
.LIB	Connecting a library of component models
.PARAM	Definition of global parameters
Statistical analysis	
.MC	Statistical analysis by the Monte Carlo method
.WCASE	Calculation of the worst case
Device models	
.ENDS	End of macromodel description
.DISTRIBUTION	Tabular definition of the rules of distribution of random variables
.MODEL	Description of component models
.SUBCKT	Beginning of the description of the macromodel
Tasks of initial conditions	
.IC	Tasks of initial conditions
.LOADBIAS	Reading from the file of node potentials of the scheme
.NODESET	The problem of nodal potentials for direct current at

	the initial iteration
.SAVEBIAS	Write to the file the node potentials of the circuit
Other directives	
.ALIASES	The beginning of the list of correspondences of names of conclusions of graphic designations of components to names of circuits of the scheme to which they are connected
.ENDALIASES	End of match list
.EXTERNAL	Specification of external ports
.OPTIONS	Setting parameters and modes of operation of the program
.STIMLIB	Assign a file name with a description of external influences
.STIMULUS	Tasks of external influences
.TEXT	Assign text variables, text expressions, or file names
*	Comment
;	Comment at the end of the line
+	Line continuation

Launch the program and create an input file

To use *Spice* for chain analysis, you must create an input file that can be created using existing text editors. The files used by *PSPice* have the extension *.cir.

Run the PSPice program and select *File – New – Text File* from the menu. When creating an input file for a schema, you should always start with a sketch and markup of the nodes. The circuit must have a node (0), which is the reference. Other nodes must be numbered or alphanumeric. Choose the name of the input file with the extension *.cir, which indicates the input file in which we include the command for each component of the scheme. Commands for components can go in any order, the first command of the input file should be the name or description. If the first line contains a description of the component, it will be ignored. The last command should be .END.

Note: Do not press ENTER after entering the last line (.END), because in this case the program will decide that you want to start the next analysis.

Logic circuits

The working version of *PSpice* contains more than a hundred logical devices. There are most logic circuits of the 7400 series, triggers, counters. In Fig. 2.4.1 shows a diagram that contains a logical element NOR. The voltages at the two inputs A and B have an amplitude of 5 V. The NOR circuit is entered by calling the subroutine (command X), in which nodes 1, 2, 3 refer to inputs A, B and output Y.

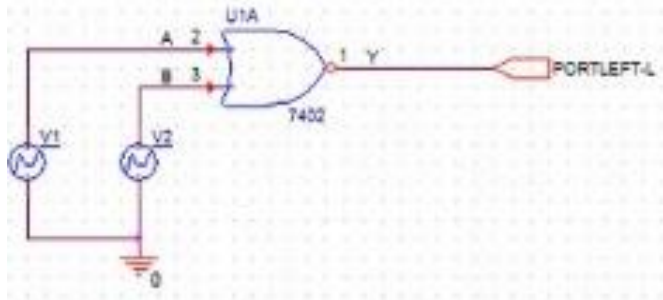


Fig. 2.4.1 – Logical element NOR

When calling the subroutine, the device is called 7402:

Digital Circuit Using NOR gate

VCC 4 0 5V

X 1 2 3 7402

V1 1 0 PWL(0s 0V 0.1ms 5V 1s 5V 1.0001s 0V 2s 0V 2.0001s 5V 3s 5V
3.0001s 0V 4s 0V 4.0001s 5V 5s 5V)

V2 2 0 PWL(0s 0V 1.5s 0V 1.50001s 5V 2.5s 5V 2.50001s 0V 3.5s 0V
3.50001s 5V 3.7s 5V 3.70001s 0V 5s 0V)

R 4 3 100k

.opt nopage

.lib eval.lib

.tran 0.01ms 5s

.probe

.end

Consider the contents of the input file. It includes the input data of the circuit components. For each component, they are shown in a separate line containing data: element type, nodes, values of the main parameter (resistance,

voltage). The .OPT (short for .OPTIONS) nopage prevents the simulation from being interrupted after the end of the source file page. The .END command is required for any input file. The .PROBE command indicates that the *Probe* charting program should be used in the analysis. Note that the order of nodes in the record R 4 3 100k means that the conditional positive direction of current will be the direction from node 4 to node 3. To start the simulation, choose *Simulation – Run*. When the simulation is complete, an area will appear on the screen where the graph can be displayed. This is the window of the *Probe* program connected to PSpice. To get a graph called *Trace*, choose *Trace – Add Trace* (Fig. 2.4.2).

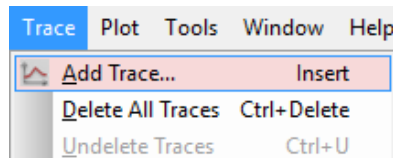


Fig. 2.4.2 – Select the items that will be displayed on the graph

In the *Trace Expressions* window, select the voltage V (1) V (2) V (3), as shown in Fig. 2.4.3.

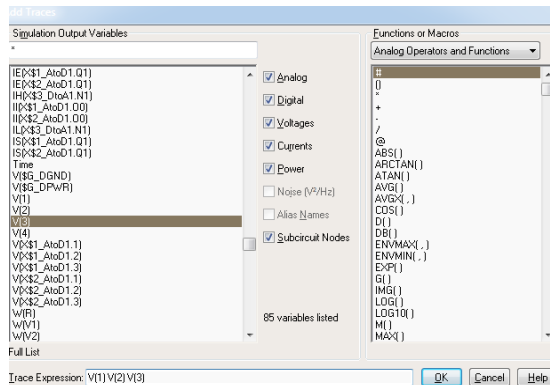


Fig. 2.4.3 – Voltage tracing

As a result, we get a window that shows the signals. If necessary, the program allows you to change the appearance of the schedule. To do this,

select *Tools – Options*. In the open window, select the *Color Settings* tab, and in the *Background* field, select the white color (Fig. 2.4.4).

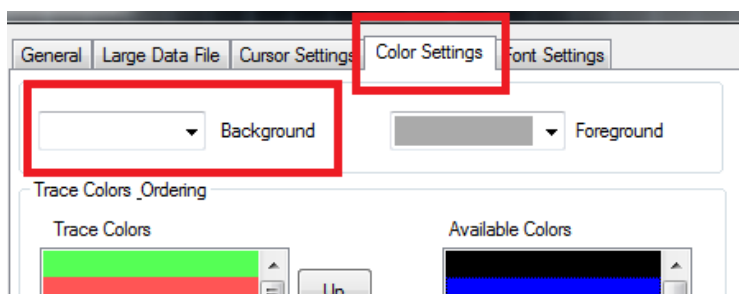


Fig. 2.4.4 – Select the tab to change the color of the graphs

To change the color of the graph itself, you need to right-click on the line of the graph and select *Trace Property*. In the open window, change the color (*Color*) and thickness (*Width*) of the line.

To obtain in the *Probe* voltage V (1) V (2) V (3) in the form of three separate graphs, located one below the other, you must select *Plot – Add Plot to Window*, as shown in Fig. 2.4.5.

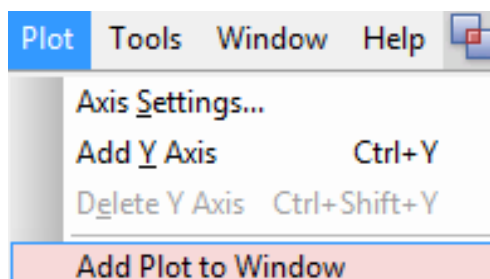


Fig. 2.4.5 – Menu for adding graphs

Then, right-click on the field that appears, and select *Add Trace*, add the voltage V (2). Similarly, perform the steps for V (1). The result is shown in Fig. 2.4.6. To view the source file, select *View – Output File* (Fig. 2.4.7). Consider the scheme shown in Fig. 2.4.8.

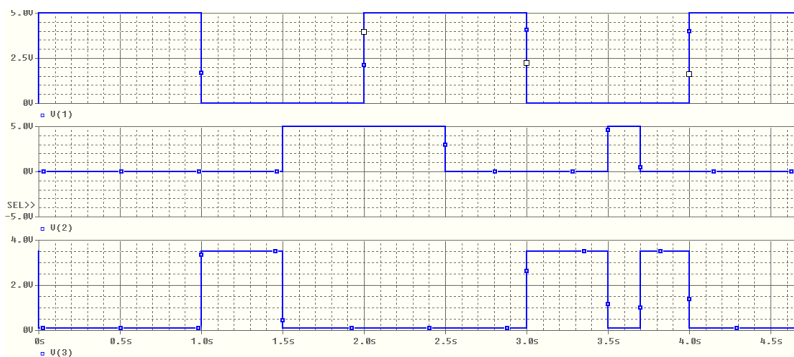


Fig. 2.4.6 – Simulation result

```

**** 07/20/18 07:43:58 ***** PSpice Lite (April 2011) ***** ID# 10813 ****

Digital Circuit Using NOR gate

****    CIRCUIT DESCRIPTION
*****

VCC 4 0 5v
X1 2 3 7402
V1 1 0 PWL(0s 0v 0.1ms 5v 1s 5v 1.0001s 0v 2s 0v 2.0001s 5v 3s 5v 3.0001s 0v 4s 0v 4.0001s 5v 5s 5v)
V2 2 0 PWL(0s 0v 1.5s 0v 1.50001s 5v 2.5s 5v 2.50001s 0v 3.5s 0v 3.50001s 5v 3.7s 5v 3.70001s 0v 5s 0v)
R 4 3 100k
.opt nopage
.lib eval.lib
.tran 0.01ms 5s
.probe
.end

```

Fig. 2.4.7 – View the program description of the scheme

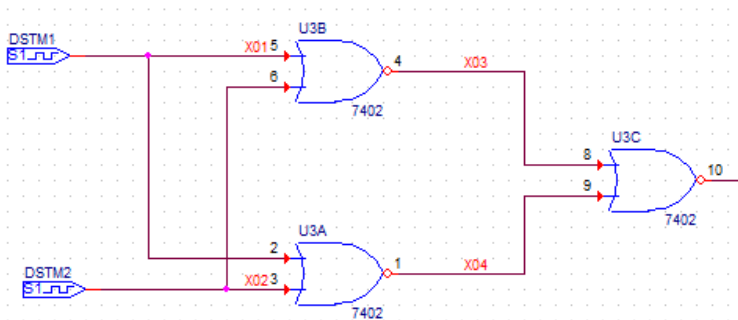


Fig. 2.4.8 – Scheme with three elements NOR

The input file will look like this:

Digital Circuit Using NOR gate

VCC 6 0 5V

X02 1 2 3 7402

X12 1 2 4 7402

X22 3 4 5 7402

V1 1 0 PWL(0s 0V 0.2s 5V 1s 5V 1.2s 0V 2s 0V 2.0001s 5V 3s 5V
3.0001s 0V 4s 0V 4.0001s 5V 5s 5V)

V2 2 0 PWL(0s 0V 1.5s 0V 1.7s 5V 2.5s 5V 2.50001s 0V 3.5s 0V
3.50001s 5V 3.7s 5V 3.70001s 0V 5s 0V)

R 6 5 100k

.opt nopage

.lib eval.lib

.tran 0.01ms 5s

.probe

.end

The simulation result is shown in Fig. 2.4.9.

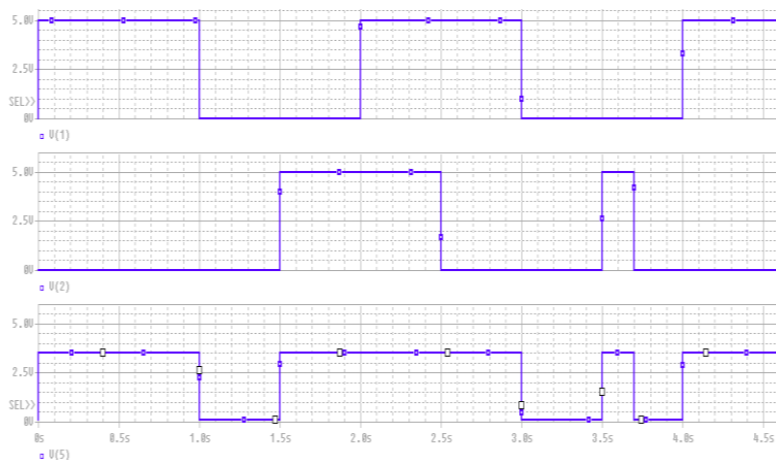


Fig. 2.4.9 – The result of modeling the scheme

Consider the scheme shown in Fig. 2.4.10 (resistor R was added to the circuit to perform analog simulation).

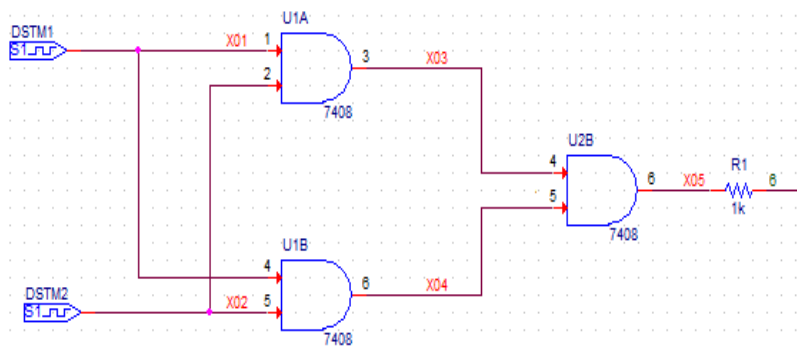


Fig. 2.4.10 – Circuit with three elements 2AND and a resistor

The input file will look like this:

Digital Circuit Using AND gate

VCC 6 0 5V

X02 1 2 3 7408

X12 1 2 4 7408

X22 3 4 5 7408

V1 1 0 PWL(0s 0V 0.1ms 5V 1s 5V 1.0001s 0V 2s 0V 2.0001s 5V 3s 5V
3.0001s 0V 4s 0V 4.0001s 5V 5s 5V)

V2 2 0 PWL(0s 0V 1.5s 0V 1.50001s 5V 2.5s 5V 2.50001s 0V 3.5s 0V
3.50001s 5V 3.7s 5V 3.70001s 0V 5s 0V)

R 6 5 100k

.opt nopage

.lib eval.lib

.tran 0.01ms 5s

.probe

.end

The simulation result is shown in Fig. 2.4.11.

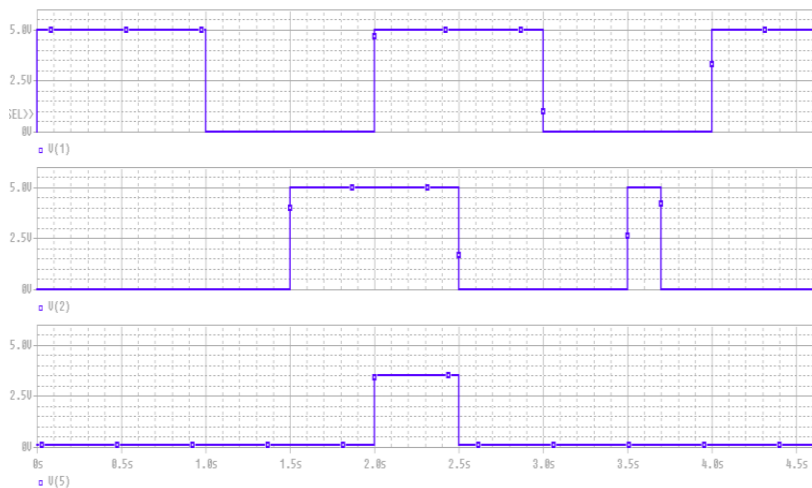


Fig. 2.4.11 – The result of the simulation

Consider the scheme in Fig. 2.4.10 with other input parameters for V1 and V2:

```

Digital Circuit Using NOR gate
VCC 6 0 5V
X02 1 2 3 7402
X12 1 2 4 7402
X22 3 4 5 7402
V1 1 0 PWL(0s 0V 0.2s 5V 1s 5V 1.2s 0V 2s 0V 2.2s 5V 3s 5V 3.2s 0V
4s 0V 4.2s 5V 5s 5V)
V2 2 0 PWL(0s 0V 1.5s 0V 1.7s 5V 2.5s 5V 2.7s 0V 3.5s 0V 3.7s 5V
3.8s 5V 4s 0V 5s 0V)
R 6 5 100k
.opt nopage
.lib eval.lib
.tran 0.01ms 5s
.probe
.end

```

The simulation result is shown in Fig. 2.4.12 and Fig. 2.4.13.

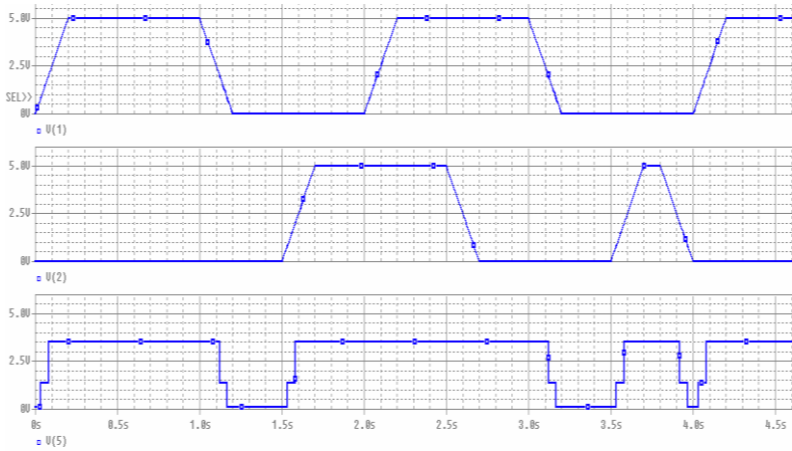


Fig. 2.4.12 – The result of the simulation

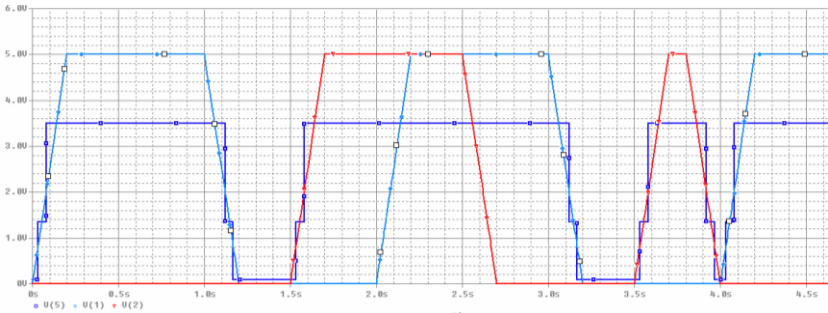


Fig. 2.4.13 – Simulation results

To make circuits from many elements, it will be convenient to represent the scheme in the form of an oriented graph, in which the logical elements are vertices, and the connection between them is the edges of the graph. Thus, the resulting graph can be represented as a list of vertices with input and output edges.

Consider an example of a circuit that performs a logical expression:

$$Y = \bar{A}B + C.$$

Remark. In the framework of this practical work it is not necessary to create such a scheme, it is only necessary to understand the device that needs to be modeled. In Fig. 2.4.14 shows a scheme for calculation, created in the form of Schematic.

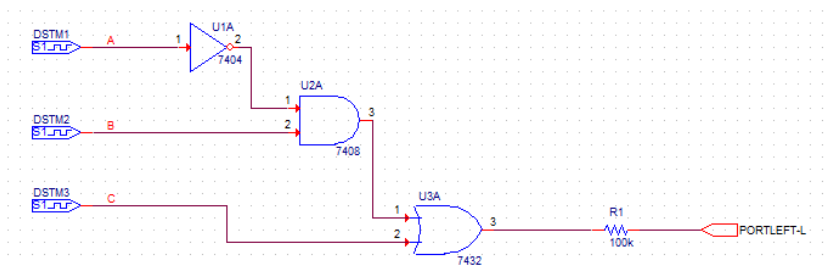


Fig. 2.4.14 – Scheme

In Fig. 2.4.15 presents a graph for description.

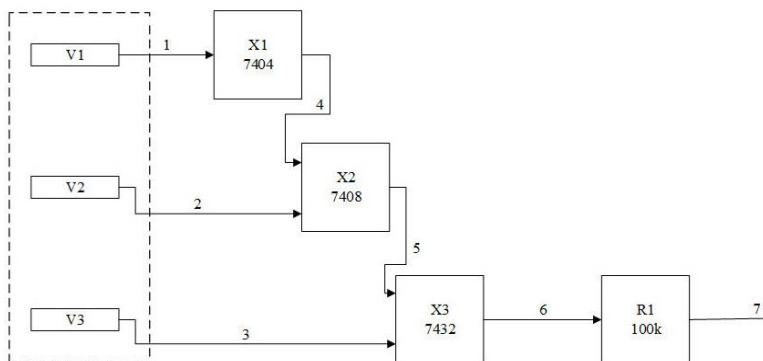


Fig. 2.4.15 – Graph of the device

As can be seen in Fig. 2.4.15 each vertex has a name and a corresponding code of the logic element, or, as in the case of a resistor, its resistance. The ribs are numbered at the request of the user.

The general structure of the .cir file can be represented as follows:

1. Name line

Digital Circuit Using Func3 gate

2. Power supply circuit

VCC 7 0 5V

In this line, the first number is the output of the circuit, the second is the input (0 indicates that it is a signal source).

3. Description of logical nodes in the format

NAME in1...inN out1...outN element description

X1 1 4 7404

4. Description of signal sources

NAME out 0 PWL (t1 V1 t2 V2 tN VN)

V1 1 0 PWL(0s 0V 0.2s 5V 1s 5V 1.2s 0V 2s 0V 2.2s 5V 3s 5V 3.2s 0V 4s 0V 4.2s 5V 5s 5V)

5. Description of the resistor

Due to the fact that both outputs of the resistor are equal, the order of writing the input and output edges is irrelevant.

R 6 7 100k

6. Closing lines

.opt nopage

.lib nom.lib

.tran 0.01ms 5s

.probe

.end

Thus, the code for modeling the device will look like this:

Digital Circuit Using Logic gate

VCC 7 0 5V

X1 1 4 7404

X2 4 2 5 7408

X3 3 5 6 7432

V1 1 0 PWL(0s 0V 0.2s 5V 1s 5V 1.2s 0V 2s 0V 2.2s 5V 3s 5V 3.2s 0V 4s 0V 4.2s 5V 5s 5V)

V2 2 0 PWL(0s 0V 1.5s 0V 1.7s 5V 2.5s 5V 2.7s 0V 3.5s 0V 3.7s 5V 3.8s 5V 4s 0V 5s 0V)

V3 3 0 PWL(0s 0V 0.2s 5V 1.2s 5V 1.4s 0V 2.2s 0V 2.4s 5V 3.4s 5V 3.6s 0V 3.8s 0V 4s 5V 4.2s 5V)

```

R 6 7 100k
.opt nopage
.lib nom.lib
.tran 0.01ms 5s
.probe
.end

```

The simulation results of the described device are shown in Fig. 2.4.16.

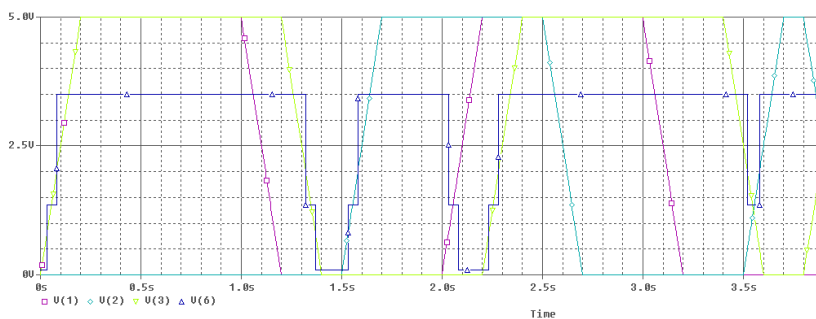


Fig. 2.4.16 – Simulation results

Tasks for practical work

Write a program and perform analog simulation of a device created in practical work № 3. Apply trapezoidal signals to the input of the designed circuit.

Practical work 5

USING THE STIMULUS EDITOR PROGRAM FOR TESTING THE DEVICE

Purpose of the work: to consider the process of describing and creating input stimuli used for device testing using the *Stimulus Editor* application CAD OrCAD.

Designing modern electronic equipment is an iterative process that consists of the stages of functional design, development of a schematic diagram and printed circuit board, manufacturing, testing and making changes to the initial design. OrCAD electronic device CAD is an example of a fully complete end-to-end design system that includes tools for drawing schematic and functional diagrams, electrical modeling tools, printed circuit board development tools, and drawing tools.

This system contains the following components: *Capture* graphic diagram editor; software tool for designing printed circuit boards *OrCAD Layout*; a tool for modeling digital devices and an interface with programmable logic integrated circuit design programs, *OrCAD Express*; analog device modeling program *OrCAD PSpice*; *PSpice A/D* analog-digital device modeling program; *PSpice Optimizer* parametric optimization program; editor of electrical element model files *OrCAD PSpice Model Editor*; editor of files containing descriptions of complex-shaped signals *OrCAD PSpice Stimulus Editor*.

The process of creating an electronic device using OrCAD is divided into several stages: creating a device diagram; testing the created printed circuit board (simulating the device's operation); preparing design documentation.

Purpose of the Stimulus Editor utility

The Stimulus Editor utility allows you to create and verify input signals for transient analysis in device components. *Stimulus Editor* allows you to create and edit voltage sources, current sources, and digital signals.

For the created analog (digital) signals, *Stimulus Editor* creates files (with the extension *.STL). The signals are created in an automated mode and you do not need to know their syntax to work with them.

Stimulus Files List on the *Configuration Files* tab of the *Simulation*

Settings dialog box (Fig. 2.5.1) allows you to view a list of the signal files that belong to the current scheme. On the same tab of the dialog box, you can add, delete, and modify a configuration file. The list displays all current changes to the signal files.

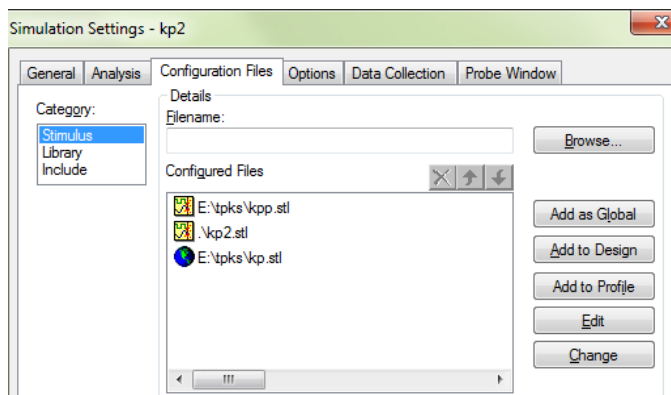


Fig. 2.5.1 – *Simulation Settings* dialog box

Signal files can be as global (marked with an icon in front of the name ) – for the *Capture circuit editor*, or local – for the current project.

When *Stimulus Editor* is launched from the *Capture circuit editor*, signal files are automatically added to the list as local to the current project.

New signal files can also be added to the list by entering a file name in the *Filename* text box and then clicking *Add to Profile*, *Add to Design*, or *Add as Global*.

Inserting and configuring signal generator parameters

Launch the *Stimulus Editor* from the *Cadence – Release – PSpice Accessories* menu. Select *File – New*. When the *Stimulus Editor* is launched for the first time, you need to adjust the scaling settings for plotting. You can use the *Axis Settings* command in the *Plot* menu (Fig. 2.5.2).

The parameters displayed in *Displayed Data Range* determine how much of these signals will be displayed on the screen. The *Extent of the Scrolling Region* parameter sets the absolute limits of the visible range. *Minimum Resolution* determines the minimum resolution along the *X* and *Y* axes.

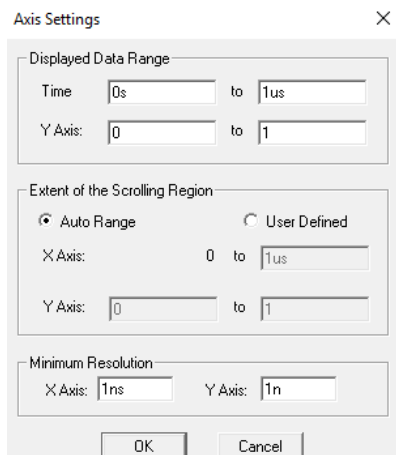


Fig. 2.5.2 – Setting parameters

Determining the type of signal

To determine the type of signal, you need to:

1. Place a sample signal generator symbol from the SOURCSTM.OLB symbol library on the diagram. For the example, we will select DIGSTIM1 (Fig. 2.5.3).

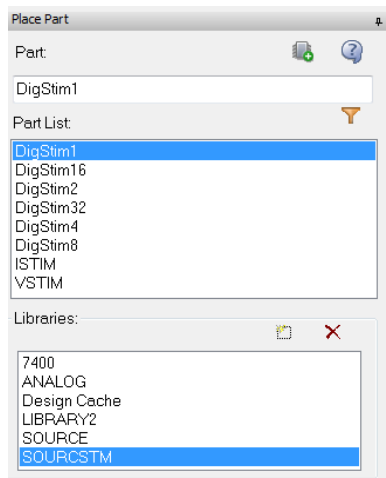


Fig. 2.5.3 – Selecting a signal generator

2. Left-click on the generator name to select it.
3. To start *the Stimulus Editor*, right-click on the generator symbol in the circuit. In the context menu (Fig. 2.5.4), select *Edit PSpice Stimulus*.
4. In the *New Stimulus dialog box*, type DIGSTIM1 in the *Name text box*. In *Digital*, select *Signal* (Fig. 2.5.5).

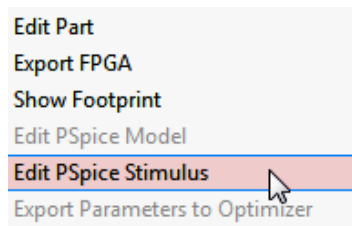


Fig. 2.5.4 – Generator context menu

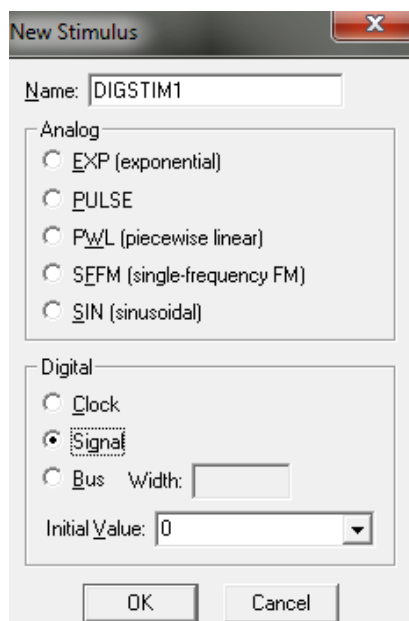


Fig 2.5.5 – *New Stimulus* dialog box

Setting the logical state parameters of a digital signal

When defining a digital signal, you can perform the following actions: add a transition; delete, edit, move a transition (these operations cannot be applied to a *Clock* generator signal).

To add the time points of the logical state change, you need to:

1. Select *File – New*.
2. In the *Plot* menu select *Axis Settings*.
3. Enter the required values in the *Displayed Range for Time* and *Timing Resolution* text boxes to add the logical state change times (Fig. 2.5.6).
4. Select the digital signal that needs to be edited. When you select *Edit – Add*, the mouse cursor changes to a pencil (Fig. 2.5.7).
5. Left-click on the signal at the location where you want to change the logical state.

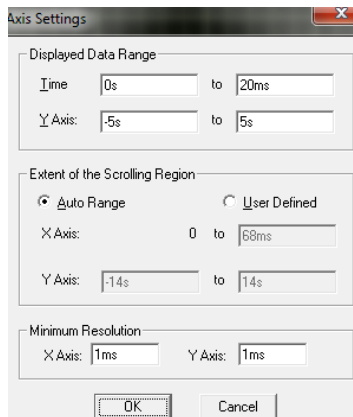


Fig. 2.5.6 – Axis Settings dialog box

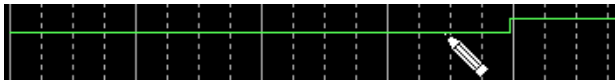


Fig. 2.5.7 – Example of editing a digital signal

5. Repeat step 4 to add additional logical state change times.
6. When you are finished editing, right-click to exit editing mode.

To move the points in time of a logical state change:

1. Click the mouse to mark the transition point to a new state.
2. If necessary, you can use the SHIFT key to select additional functions.

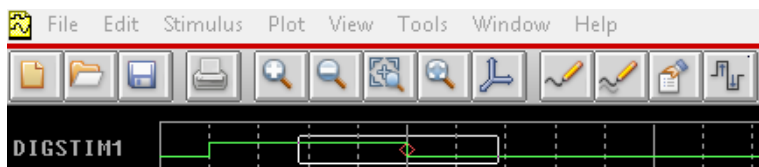


Fig. 2.5.8 – Shifting the signal change time

To edit the time points of the logical state change, you must:

- 1) perform one of the following actions:
 - select the time of the change in the logical state that you want to edit (then in the *Edit menu* – select the *Attributes* item);
 - left-click on the logical state change point that needs to be edited;
- 2) in the *Edit Digital Transition* dialog box (Fig. 2.5.9), edit the time and value for the moment of the logical state change;
- 3) click the *OK button*.

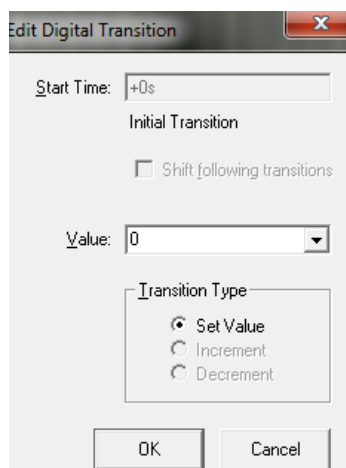


Fig. 2.5.9 – *Edit Digital Transition* dialog box

To delete the moments of change of logical state, you must:

1. Left-click on the moment of change of logical state.
2. If necessary, you can use the SHIFT key.
3. In the *Edit* menu, select *Delete*.

To create a *Clock* signal you need:

1. Select the signal generator you want to use as *Clock* in the *Stimulus Editor*.

2. In the *Stimulus* menu, select *Change Type* (Fig. 2.5.10). The *Change Stimulus Type* dialog box will open. In *Type*, select the *Clock* radio button (Fig. 2.5.11). Click the *OK* button.

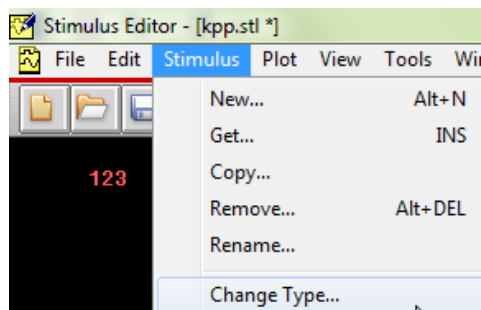


Fig. 2.5.10 – *Stimulus* menu commands

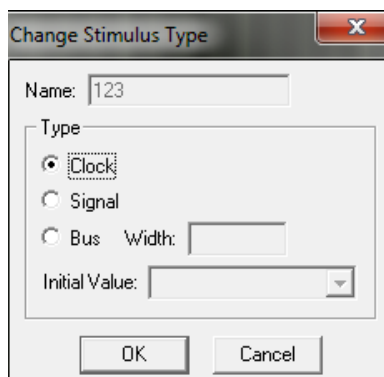


Fig. 2.5.11 – *Change Stimulus Type* dialog box

3. Enter values for *the Clock signal attributes using the Clock Attributes* dialog box. As an example, in Fig. 2.5.12, the following parameters are entered in *Clock Attributes*: frequency – 20 MHz, duty cycle – 0.5, initial value – 1, time delay – 5 ns.

4. In the *File menu* select *Save*.

To change the attributes of *the Clock signal*, you must either double-click *the Clock* signal name to the left of the axis, or click *the Clock* signal name and select *Attributes* from the *Edit menu*. You can then make the necessary changes to *the Clock* signal attributes.

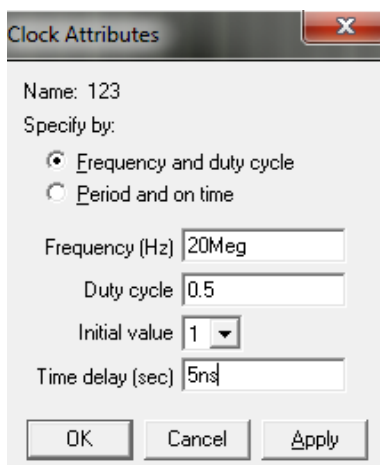


Fig 2.5.12 – *Clock Attributes* dialog box

When creating a new signal, select the signal type in the menu. Signals are divided into two groups – analog and digital. In this case, select *Clock* from the *Digital group*. Enter the signal name in the text field (Fig. 2.5.13). Click *OK*.

The signal setup window will appear, as well as the signal, which is still a straight line. In the signal setup window, you can define two modes: frequency and time.

There is an *Apply button*, which allows you to view the results without closing the edit window. The *OK button* will save the current state of the signal and close the window.

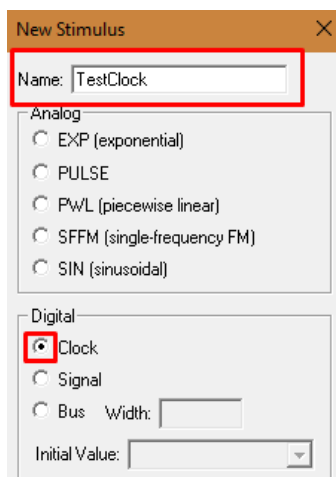


Fig. 2.5.13 – Selecting the signal type

Frequency mode

Frequency – specified in Hz.

Duty cycle – determines the ratio of the time the signal is in the "1" position (0.5 – equally, 1 – only in the "1" position, 0 – only in the "0" position).

Initial value – the initial value of the signal.

Time delay – start delay.

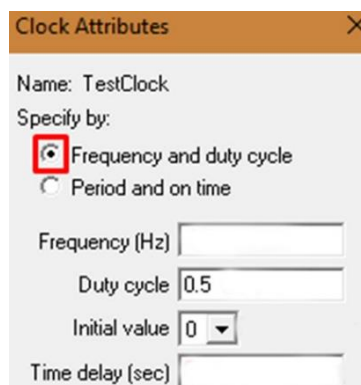


Fig. 2.5.14 – Frequency mode

Time mode

Period – the time it takes for a full cycle.

On time is the same as in frequency mode. The only difference is that now it is a number from "0" to the period value (the larger the value, the longer the signal will be in the "1" position).

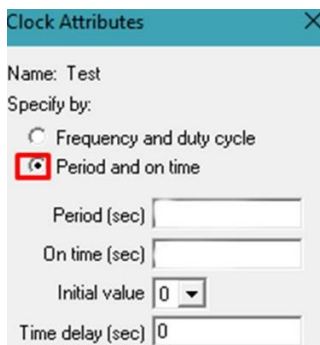


Fig. 2.5.15 – Time mode

Creating an arbitrary digital signal

Create a new signal, but now select *Signal*. You can also specify the initial value of the signal (Fig. 2.5.16). Click OK.

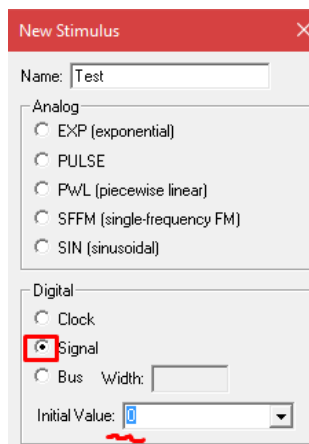


Fig. 2.5.16 – Setting the initial signal value

A signal will appear, which is a straight line. To edit it, use the editing tool: *Edit – Add*.

Left-click on the signal. If a message appears warning that it is impossible to set the point (Fig. 2.5.17), go to *Plot – Axis Settings* and in the lower field of the window that appears (Fig. 2.5.18), set the value to 1 ns.

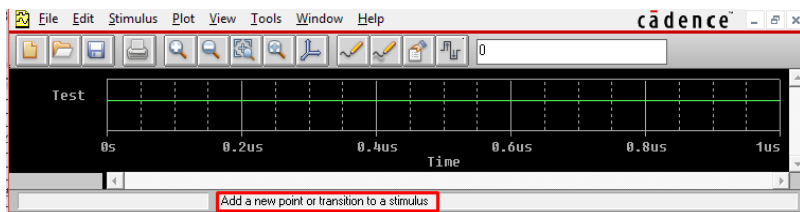


Fig. 2.5.17 – Error message

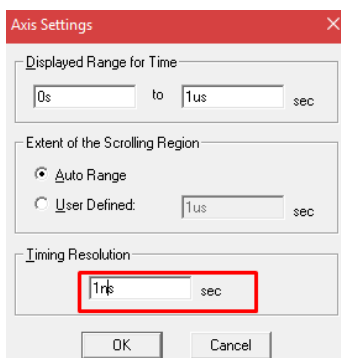


Fig. 2.5.18 – Setting *Timing Resolutions*

Click on the signal with the "pencil" - this will create a reference point. The signal value is inverted after the current point and to the end of the signal, provided that there are no other reference points further (Fig. 2.5.19).

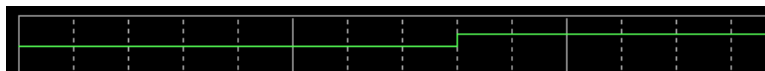


Fig. 2.5.19 – Modified signal

To create multi-bit signals (buses), you must:

- create a multi-bit digital signal;
- arrange the moments of change of logical state;
- additionally determine the type of number system for a multi-bit signal.

To create multi-bit signal sources, you need to:

1. In the *Stimulus menu*, execute the *New command*.
2. In the *Name text field of the New Stimulus* dialog box (Fig. 2.5.20), enter the source name – *Bus*.
3. In *Digital*, select the *Bus switch*.
4. To set the number of outputs (*bus width*), enter the required number in the *Width text field*.
5. Click the *OK button*.

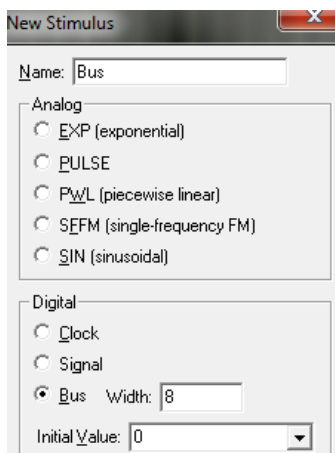


Fig. 2.5.20 – *New Stimulus* dialog box

To insert the time points of the logical state change, you must:

1. In the *Plot menu*, select *Axis Settings*.
2. Enter the required values in the *Displayed Range for Time* and *Timing Resolution* text boxes to add the time points of the logical state change:
 - a) constant value: 12,
 - b) increase: +12; H,
 - c) negative growth: -12; O.

3. In the *Edit* menu select *Add*.

4. In the numeric value field on the toolbar enter a text description that defines the form of the multi-bit signal. If you do not enter a value for the form of the number system (*radix*), *Stimulus Editor* uses the default form of the number system.



Fig. 2.5.21 – Digital signal value field on the toolbar

5. Left-click on the signal.

6. Repeat steps 4 – 5 as many times as necessary.

7. When finished, right-click to exit editing mode (Fig. 2.5.22).

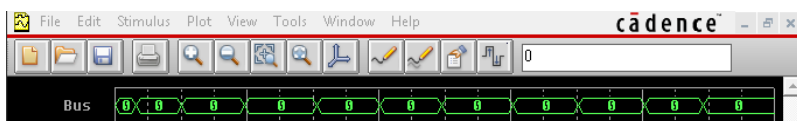


Fig. 2.5.22 – Received multi-bit signal sources

To set the default number system, you need to:

1. In the *Tools* menu, select *Options*.

2. In the *Radix* list of the *Bus Display Defaults* frame, select the desired default number system type.

3. Click the *OK* button.

You can also edit the bus with a "pencil", but this is not very convenient. The easiest way to edit the bus is using the text mode. Let's make some changes. Fig. 2.5.23 shows the signal graph before editing, and Fig. 2.5.24 shows the signal graph after editing.

Consider the problem of creating a signal that is similar to an existing one. For example, there is a signal that is shown in Fig. 2.5.25. It is necessary to create a signal that consists of three sequences (Fig. 2.5.26).

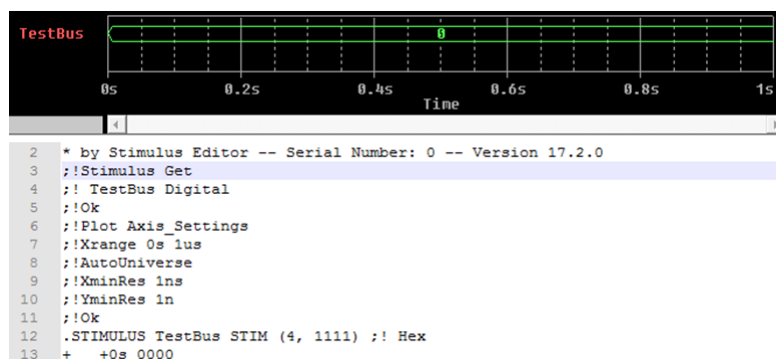


Fig. 2.5.23 – View before editing

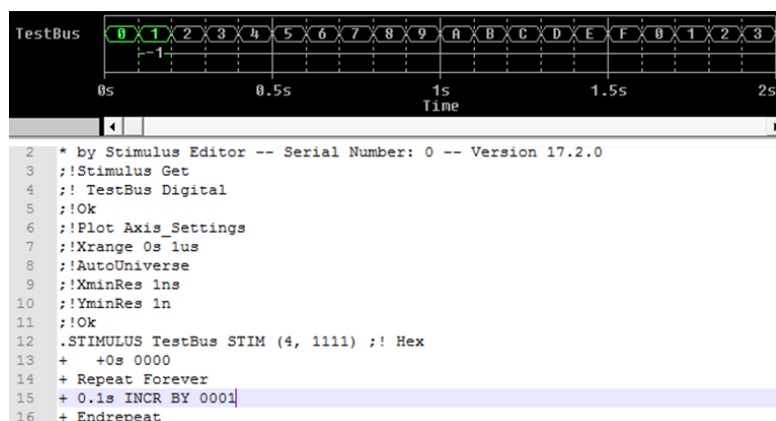


Fig. 2.5.24 – View after editing

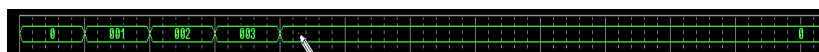


Fig. 2.5.25 – Digital multi-bit signal



Fig. 2.5.26 – Generated digital multi-bit signal

This problem can be solved using a standard text editor. This file contains a sequence of time points of change of logical state, which is created in the initial signal. You can change the description of the signal by repeating

it several times, for this you need to do:

1. In *the Stimulus Editor*, save and close the signal file.
2. Open the signal file in a standard text editor.
3. Find the sequence of lines containing the text to be repeated. Each line begins at the time of the logical state change or signal value change. Before this line, insert the following line:

+ *Repeat for n_times*,

where *n_times* is either a positive integer (the number of repetitions) or the FOREVER keyword, which ensures that this sequence is repeated an unlimited number of times.

5. Insert the following line below this line:

+ *Endrepeat*.

6. In the *File menu*, select *Save*.

If the sequence needs to be repeated three times, then the signal file needs to be modified as shown below (adding the lines in bold):

+ **Repeat for 3**

+ +0s 000000000

+ 250us INCR BY 000000001

+ 500us 000000010

+ 750us INCR BY 000000001

+ 1ms 000000000

+ **End repeat**

Creating analog signals

Creating an exponential signal

Create a new signal by selecting *EXP* (Fig. 2.5.27), enter a name and click *OK*.

Fig. 2.5.28 shows the settings window:

Initial Value – initial value (can be set to any);

Peak value – maximum value (any value can be set);

Rise delay – delay the start of the signal for a certain time;

Rise time constant – indicates how many seconds it will take for the signal value to be higher than 60 %;

Fall Delay – the point in time when the signal starts to fall;

Fall time constant – indicates how many seconds it will take for the signal value to fall below 60 %.

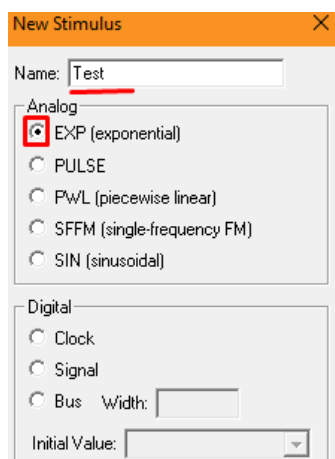


Fig. 2.5.27 – Selecting the signal type

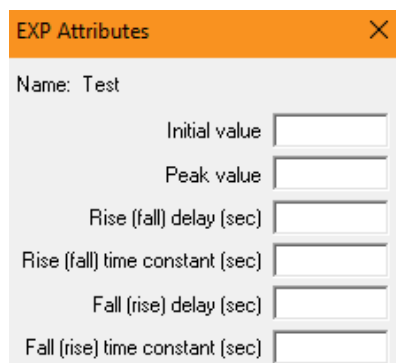


Fig. 2.5.28 – Signal settings window

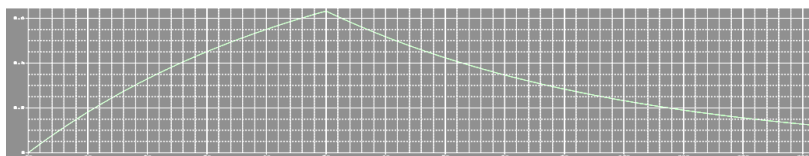


Fig. 2.5.29 – Exponential signal

Creating a pulse signal.

Create a new signal by selecting *PULSE*, enter a name, and click *OK*.

In Fig. 2.5.30 shows the signal setup window:

Initial Value – initial value (can be set to any);

Peak value – maximum value (any value can be set);

Delay – delay;

Rise time – duration of the front;

Fall time – duration of decline;

Pulse width – the time at which the signal will be at its maximum;

Period – signal period.

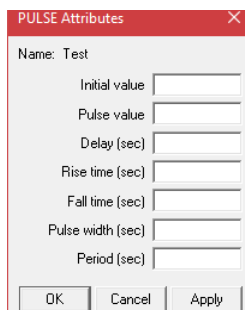
A screenshot of a software dialog box titled "PULSE Attributes". It contains several input fields for configuring a pulse signal. The fields are labeled: "Name: Test", "Initial value", "Pulse value", "Delay (sec)", "Rise time (sec)", "Fall time (sec)", "Pulse width (sec)", and "Period (sec)". Each label is followed by a text input box. At the bottom of the dialog are three buttons: "OK", "Cancel", and "Apply".

Fig. 2.5.30 – Signal settings window

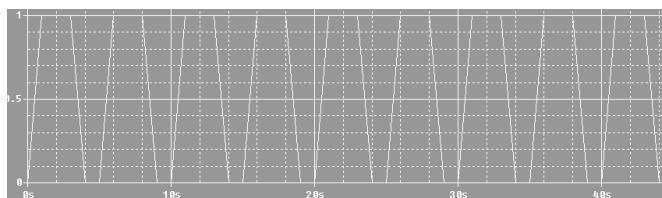


Fig. 2.5.31 – Pulse signal

Arbitrary straight line graph

Create a new signal by selecting the *PWL* item and enter a name. Using the "pencil" we draw lines. It is convenient to specify a broken line in text mode. Fig. 2.5.32 shows the code for the figure shown in Fig. 2.5.33. The coordinates of the points are circled with a rectangle.

```

12 :!Ok
13 .STIMULUS_333_FWL
14 + TIME_SCALE_FACTOR = 1
15 + VALUE_SCALE_FACTOR = 1
16 + ( 0, 0 )
17 + ( 1e-007, 4.95e-007 )
18 + ( 1.05e-007, 3.74e-007 )
19 + ( 2.2e-007, 3.42e-007 )
20 + ( 2.28e-007, 3.82e-007 )
21 + ( 2.29e-007, 3.44e-007 )
22 + ( 4e-007, 3.75e-007 )
23 + ( 4.25e-007, 4.24e-007 )
24 + ( 4.38e-007, 4.51e-007 )
25 + ( 4.86e-007, 3.79e-007 )
26 + ( 4.87e-007, 3.78e-007 )
27 + ( 5.27e-007, 2e-009 )
28

```

Fig. 2.5.32 – Shape code

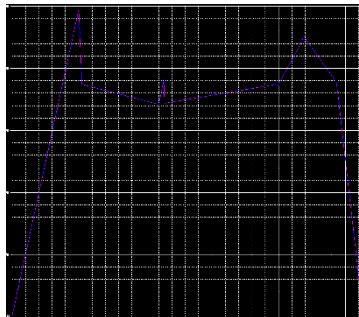


Fig. 2.5.33 – Figure created using polylines

Sinusoidal signals

Create a new signal by selecting *SIN*, enter a name, and click *OK*.

Fig. 2.5.34 shows the signal setup window:

Offset value – vertical offset value (determines how many points to shift the graph relative to the *Y axis*);

Amplitude – amplitude (maximum or minimum value);

Frequency – frequency;

Time delay – shift relative to the *X axis* to the right for a certain time;

Damping factor – the larger the value, the more the oscillations will be damped out and then increase again. If you set it to "1", then the oscillations will be weaker or stronger by 2 times in each new period compared to the previous one;

Phase angle – the phase of oscillation (specified in degrees).

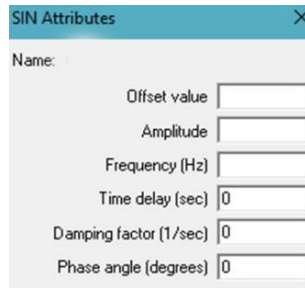


Fig. 2.5.34 – Sinusoidal signal parameters window

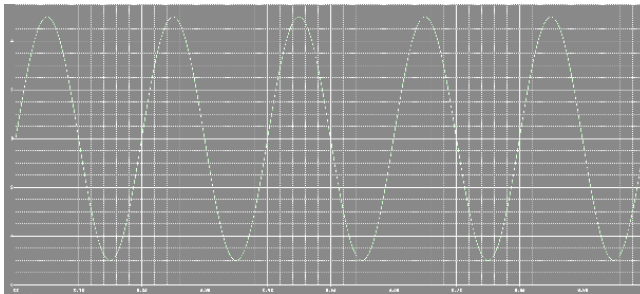


Fig. 2.5.35 – Sinusoidal signal

Creating a piecewise linear signal

To create a piecewise linear signal, you need to:

1. Open an existing diagram or create a new one.
2. In the *Place menu*, execute the *Part command*. Find the symbol library file *sourcestm.olb*, select *VSTIM* and mark it on the diagram.
3. Left-click on the *Implementation property*.
4. In the *Display Properties dialog box* (Fig. 2.5.36), enter the name of the created signal in the *Value field* – *Vfirst*.
5. If the work is carried out in a new scheme, execute the *Save command* from the *File* menu.
6. To select the *VSTIM* symbol, click on it with the left mouse button.
7. Right-click on the generator in the diagram and select *Edit PSpice Stimulus* in the context menu.
8. *The Stimulus Editor will launch* and the *New Stimulus dialog box* will be displayed (Fig. 2.5.37), showing that the signal is named *Vfirst*.

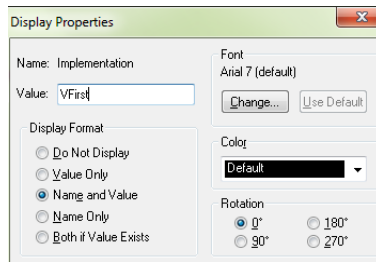


Fig. 2.5.36 – *Display Properties* dialog box

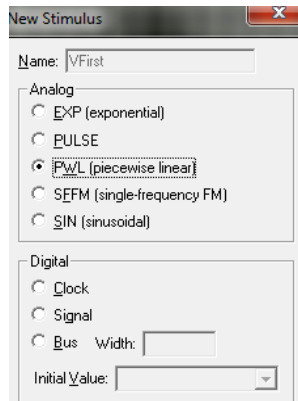


Fig. 2.5.37 – *New Stimulus* dialog box

8. Select the *PWL* radio button and press the *OK* button. The cursor will change to a pencil. A message in the status bar at the bottom of the screen will indicate that the process of adding new data points to the signal is in progress.

9. Move the cursor and left-click. This adds and connects the points that define the signal (Fig. 2.5.38).

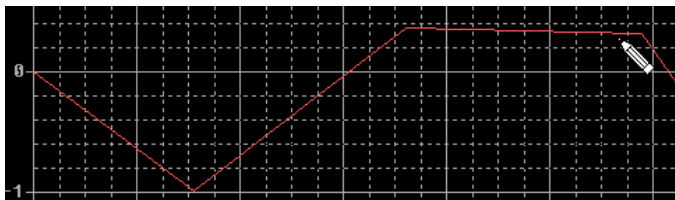


Fig. 2.5.38 – Editing (creating) a signal

10. Right-click to stop adding points.
11. On the *File menu* click *Save*.
12. To delete a point, click on the manipulator label and press the *Delete* button. To add *PWL* signal breakpoints, execute the *Add command* in the *Edit menu* (you can use the *Add Point* toolbar button).

To create a sinusoidal signal with a frequency of 10 kHz and variable amplitude, you need to perform the following steps:

1. Open an existing schematic and place a *VSTIM* source symbol on it. Right-click on it and select *Edit PSpice Stimulus*.
2. In the *New Stimulus dialog box*, execute the *Cancel command*.
3. Define the signal parameters: in the *Tools menu*, execute the *Parameters command* and in the *Definition text field*, enter $AMP = 1$ (Fig. 2.5.39).

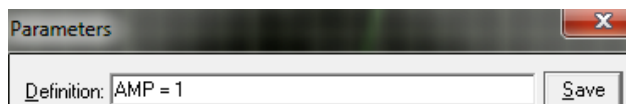


Fig 2.5.39 – *Parameters* dialog box

4. Execute the *New command* from the *Stimulus menu*, enter the signal name *Vsin* in the dialog box and select *SIN* (Fig. 2.5.40).

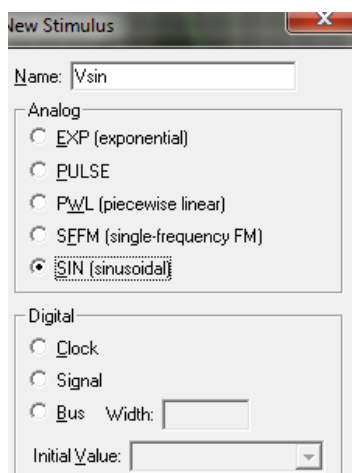


Fig. 2.5.40 – *New Stimulus* dialog box

5. *Attributes* dialog box (Fig. 2.5.41): enter "0" in the *Offset Value* field; enter {AMP} in the *Amplitude* field (the curly brackets indicate that the expression must be evaluated during simulation); enter "10k" in the *Frequency* field and click *OK*; in the *File* menu, execute the *Save* command.

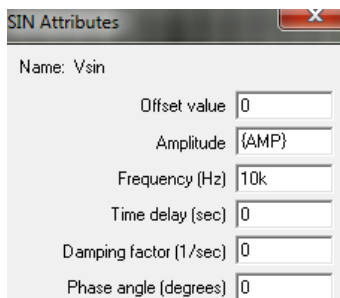


Fig. 2.5.41 – *Attributes* dialog box

6. In the *Capture schema editor*, place and define the *PARAM* symbol:
- in the *Place* menu, execute the *Part* command;
 - select the *PARAM* symbol in *special.olb*;
 - place the symbol on the diagram and double-click on it;

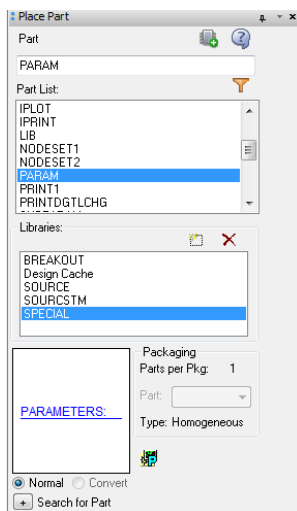


Fig. 2.5.42 – *Place Part* dialog box

- click *New* to add a new user of the AMP property;
- enter AMP (without curly braces) in the *Name* field of the Property Editor;
- set the value of the AMP property to "1" (Fig. 2.5.43).

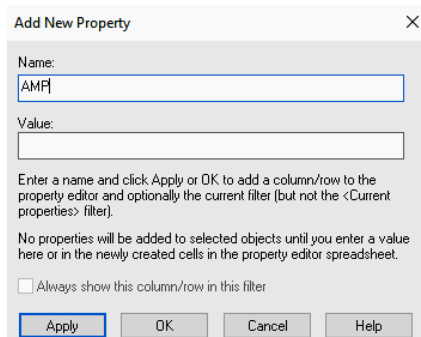


Fig. 2.5.43 – Add New Property dialog box

- Set the following properties:
 - in the *Pspice* menu, select *Edit Simulation Profile* and select the *Parametric Sweep* option;
 - select *Global Parameter* in *Sweep variable*;
 - select *Linear* in *Sweep type*;
- Enter AMP in the *Parameter text* field.
- Define values for the *Start Value*, *End Value*, and *Increment* text fields.

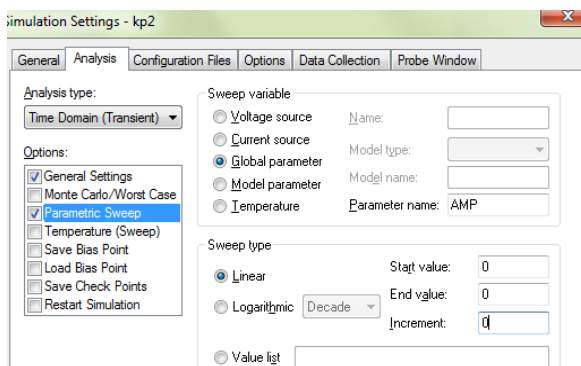


Fig. 2.5.44 – Simulation Settings dialog box

You can set up Transient, AC or DC analysis and run simulations. Use *the Part editor* in *Capture* to create and edit new signal source symbols with the properties listed in Table 2.6.

Table 2.6 – Symbol properties

Property	Value
Implementation type	PSpice Stimulus
Implementation	Signal model name
STIMTYPE	The signal type becomes ANALOG or DIGITAL

To edit existing signals, start *the Stimulus Editor*, select *File/Open* to open the desired signal library. Then double-click on the graph name (below the *X-axis* for analog signals and to the left of the *Y-axis* for digital signals). The *Stimulus Attributes* window will open, where you can change the signal attributes and observe the effect of the changes.

Creating a custom signal in "manual" mode

In *Stimulus Editor*, a signal can be created manually and its specifications saved in a file. These specifications can then be linked to a sample signal on a schematic or a signal source symbol in the symbol library.

To configure the signal manually, perform the following steps:

1. Launch *Stimulus Editor* by selecting *Stimulus Editor* from the Windows Start menu /OrCAD.../PSpice Accessories/.

2. Open the signal file by selecting *Open* from *the File menu*.

3. Create one or more signals for use in the circuit.

For each signal:

- we will put any name;
- we will provide technical requirements for the transient process regime;
- in the *File menu*, select *Save*.

4. In the schematic editor page, configure the *Stimulus Editor* output file for the schematic:

- *Simulation Settings* dialog box, select *Edit Simulation Profile* in the *Pspice menu*;
- *Simulation Settings* dialog box, select the *Configuration Files* tab;

- click *Include* in the *Category* field to display the list of *Include files* (Fig. 2.5.45);
- enter the file name;
- for the current profile, click the *Add to Profile* button. For a local project, click the *Add to Design* button. For a global project, click the *Add as Global* button;
- click *OK*.

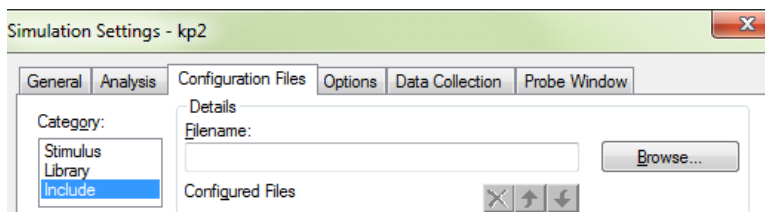


Fig. 2.5.45 – *Simulation Settings* dialog box

5. When modifying any signal sample on the diagram or symbols in the symbol library, we will refer to the new technical requirements of the signal.

6. Let's relate the signal specifications for the transient mode to the sample signal source:

- place the signal generator symbol on the diagram: VSTIM, ISTIM or DIGSTIM and click on each of them;
- in the *Edit menu*, select *Properties*;
- select the *Implementation environment*, enter the signal name and click *Apply*.

Designator	Graphic	ID	Implementation
	VSTIM.Normal		NAME

Fig. 2.5.46 – Fragment of the *Properties* dialog box

- finish the task of any VSTIM or ISTIM examples by selecting *Edit – Properties* from the menu and editing their attributes for DC and AC analysis. In DC, enter the value of the constant component of the voltage (current). In

AC, enter the value of the amplitude of the harmonic component of the voltage (current);

- close *the Property editor*.

7. To globally change the signal for a symbol:

- select the symbol that needs to be edited;

- in the *Edit menu*, select *Part* to launch *the Part editor*;

- create or change the symbol description by defining the property:

Implementation – the name of the signal, as defined in *the Stimulus Editor*.

Tasks for practical work

Use the *Stimulus Editor program* to create sinusoidal, pulsed, and exponential signals with a period of $T = N \mu\text{s}$ (where N – number of variant).

Practical work 6

USING ORCAD TO MODEL DIGITAL BLOCKS BASED ON THE MONTE CARLO METHOD

The purpose of the work: to simulate the device and investigate it using the Monte Carlo method.

The Monte Carlo method is the general name for a group of methods that are based on obtaining a large number of realizations of a random process, formed in such a way that its probabilistic characteristics coincide with similar values of the problem being solved.

The Monte Carlo method is a simulation method for approximately reproducing real phenomena, which allows you to build a model by minimizing and maximizing data values. Model building begins with determining functional dependencies in a real system, after which a solution can be obtained.

There is no single Monte Carlo method, the term describes a widely used class of approaches. These approaches use a single underlying template:

- 1) determine the range of possible input data;
- 2) randomly generate input data using some given probability distribution;
- 3) perform deterministic calculations on the input data;
- 4) combine the intermediate results of individual calculations into the final result.

The idea of the Monte Carlo method is that a process of converting digital data is created in a computer that is similar to the real process. The probabilistic characteristics of both processes (real and simulated) coincide with a certain accuracy.

The main disadvantage of this method is that it cannot replace analytical methods in calculating significantly new phenomena where the disclosure of qualitative patterns is required. The advantage of the Monte Carlo method is that it is able to "work" where other methods fail.

Modeling the operation of the 4NAND element

Let's launch the OrCAD program and assemble the diagram as shown in

Fig. 2.6.1. The data obtained as a result of the simulation (Fig. 2.6.2), although providing information about the behavior of the assembled device, do not allow us to consider some important factors that may affect the operation of the real device. This is why the Monte Carlo simulation method can be applied in this case.

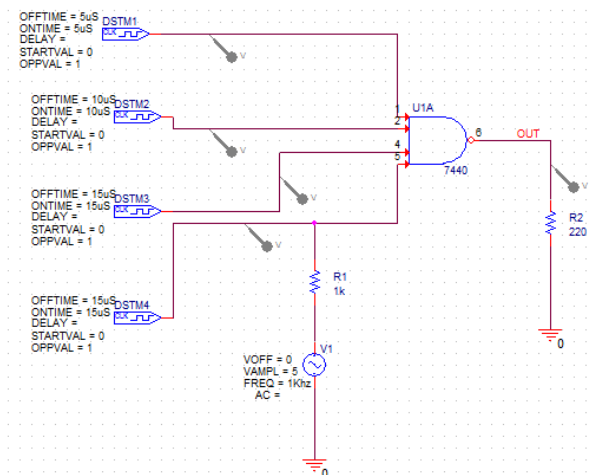


Fig. 2.6.1 – Assembled circuit

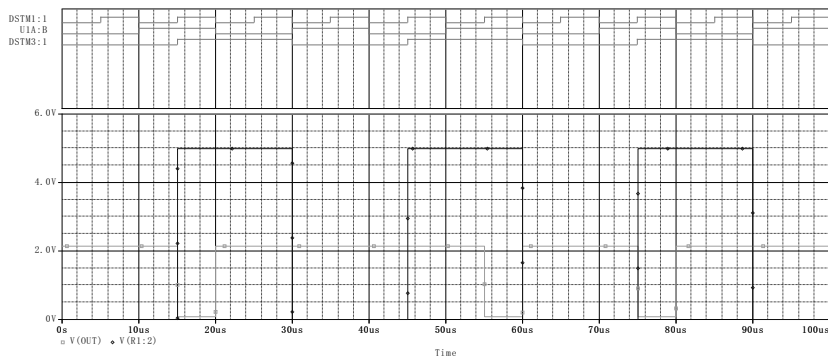


Fig. 2.6.2 – Result of circuit simulation

Monte Carlo simulation of circuit operation using PSpice

Let's make some changes to the circuit by setting the permissible

deviations (the so-called "tolerance") in the settings of both resistors to 10 % (Fig. 2.6.3).

montekarlo*

PAGE1*

SCHEMATL.*

New Column...

Apply

Display...

Delete Property

Filter by: < Current properties >

		Source Package	Source Part	TC1	TC2	TOLERANCE
1	SCHEMATIC1 : PA	R	R.Normal	0	0	10%

Fig. 2.6.3 – Setting the allowable deviation

Now in the simulation profile settings, select *Monte Carlo* and set all the necessary parameters, as shown in Fig. 2.6.4 (the simulation results are shown in Fig. 2.6.5). As can be seen from Fig. 2.6.5, at the output of the circuit we obtain a voltage in the range of 2.13...2.15 V. This is due to the fact that a resistor with a resistance of 220 Ohm and an allowable deviation of 10 % is connected to the output of the device, which allows us to obtain different voltage values for different numbers of iterations.

The change in the output voltage value is due to the fact that when starting each iteration, the parameters of the resistors at the input and output of the circuit change with a permissible deviation randomly. In the absence of a resistor at the output, we would also get different voltage values, but in a different range.

Simulation Settings - qq

General

Analysis

Configuration Files

Options

Data Collection

Probe Window

Analysis type:

Time Domain (Transient)

Options:

☒ General Settings
 ☒ Monte Carlo/Worst Case
 ☐ Parametric Sweep
 ☐ Temperature (Sweep)
 ☐ Save Bias Point
 ☐ Load Bias Point
 ☐ Save Check Points
 ☐ Restart Simulation

☒ Monte Carlo
 ☐ Enable PSpice AA support for legacy
 ☐ Worst-case/Sensitivity

Output variable: V(OUT)

Monte Carlo options

Number of runs: 20

Use distribution: Uniform

Random number seed: 23234 [1..32767]

Save data from: All runs

Worst-case/Sensitivity options

Vary devices that have both DEV and LOT tolerances

Fig. 2.6.4 – Simulation profile settings

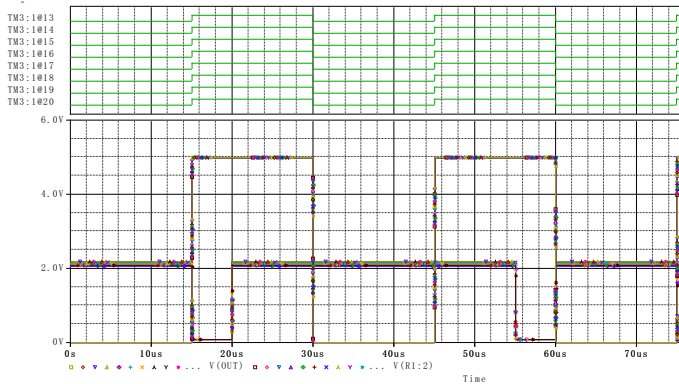


Fig. 2.6.5 – Simulation results using the method Monte Carlo in PSpice

Now we will remove all the graphs (*Trace menu – Delete all Traces*) and add a new graph that displays the maximum values of the output voltage (*Trace menu – Add Trace*, in the right list select MAX (), and in the left – V (OUT)).

If everything is done correctly (Fig. 2.6.6), then the lower text field will contain MAX(V (OUT)). The simulation result is shown in Fig. 2.6.7.

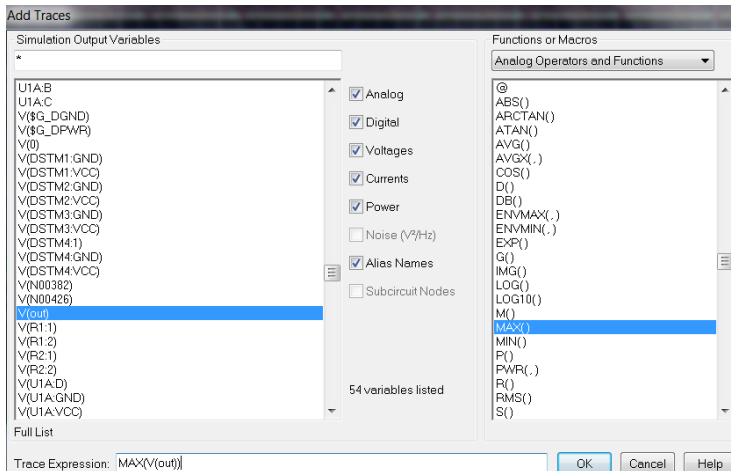


Fig. 2.6.6 – Adding a new graph

Despite the fact that the obtained results allow us to get an idea of what the voltage at the output of the circuit can be, and in what range it can vary, it is almost impossible to use them.

This is due to the fact that for 20 runs such a number of graphs are constructed that it is impossible to adequately place them on one coordinate grid. Therefore, when simulating a circuit using the Monte Carlo method, it is necessary to use not the standard *PSpice*, but *PSpice Advanced*.

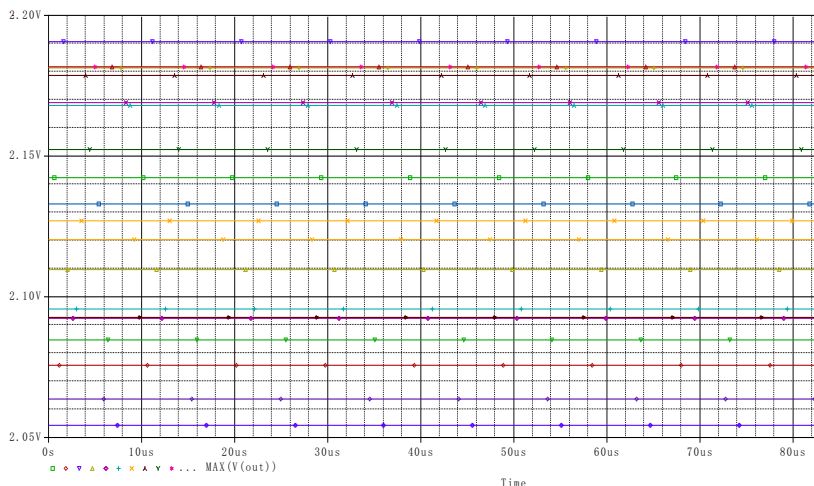


Fig. 2.6.7 – Simulation results

Monte Carlo simulation of circuit operation using PSpice Advanced

We will use the previously developed scheme with the element 4NAND. We select *PSpice – Advanced Analysis – Monte Carlo*. After that, a window will open, as shown in Fig. 2.6.8.

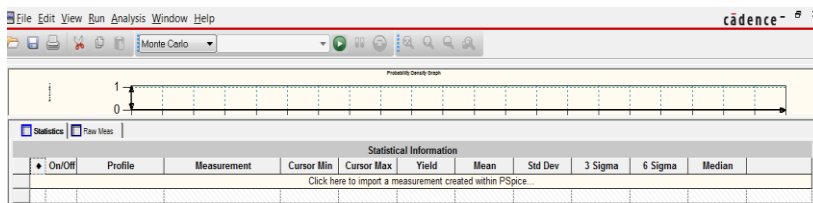


Fig. 2.6.8 – PSpice Advanced Analysis window

Select *Analysis – Monte Carlo – Create New Measurements* from the menu. In the window in the right list, select Max (1), and in the left – V (Out) and click OK.

Now you need to configure the simulation profile by setting the required number of iterations and other parameters. To do this, select the *Edit – Profile Setting* menu item and enter the necessary data in the appropriate fields (Fig. 2.6.9):

Number of runs – 20 iterations (for each iteration, the component parameters with permissible deviations will change randomly);

<u>N</u> umber of Runs:	20
<u>S</u> tarting Run Number:	1
Ran <u>d</u> om Seed <u>V</u> alue:	1
Number of <u>B</u> ins:	10

Fig. 2.6.9 – Settings window

Starting run number – the starting iteration number (default is "1");

Random seed value – random seed value for the pseudo-random number sequence (equal to "1"). The random number generator will use this value to generate the random number sequence;

Number of bins – 10 columns of the graph (histogram). This number determines the number of sections in the histogram.

We run the simulation and obtain a plot of the probability distribution function (PDF) as shown in Fig. 2.6.10. In Fig. 2.6.11 shows a plot of the integral distribution function (CDF).

Simulation of the adder operation using the Monte Carlo method

Let's simulate the operation of a more complex device. First, we assemble a circuit of a single-bit adder (Fig. 2.6.12), based on which we assemble a circuit of a four-bit adder (Fig. 2.6.13).

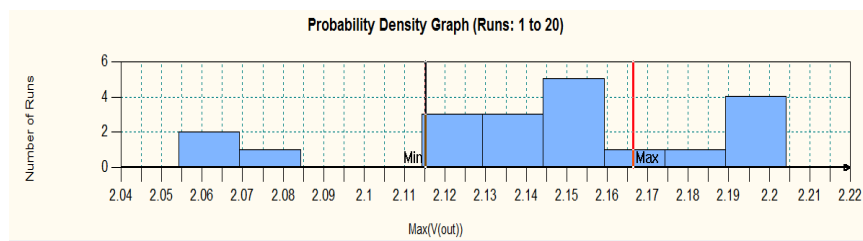


Fig. 2.6.10 – Graph of the probability distribution function

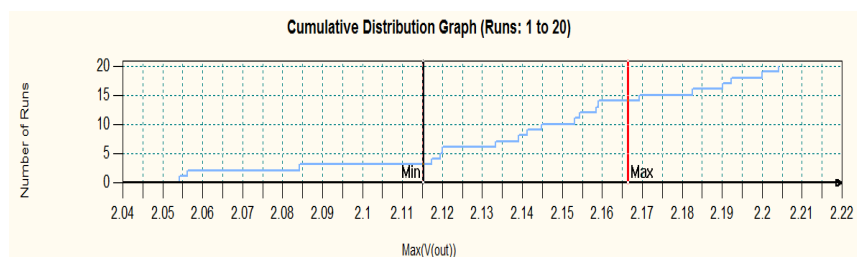


Fig. 2.6.11 – Graph of the integral distribution function

The internal structure of the four-bit adder macro is shown in Fig. 2.6.14. The input signal generator settings are shown in Fig. 2.6.15.

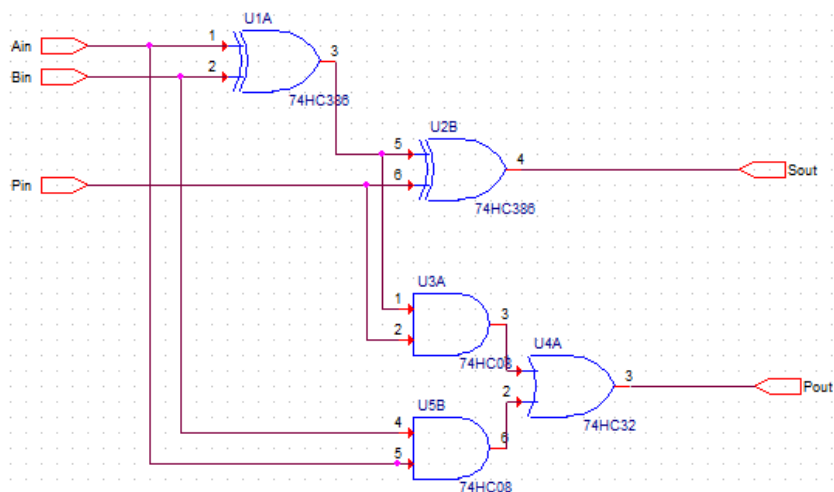


Fig. 2.6.12 – Scheme of a one-bit adder

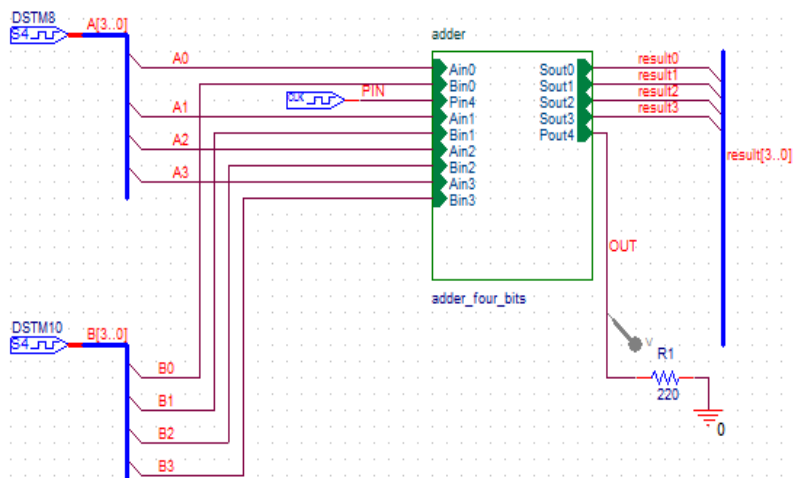


Fig. 2.6.13 – Four-bit adder research scheme

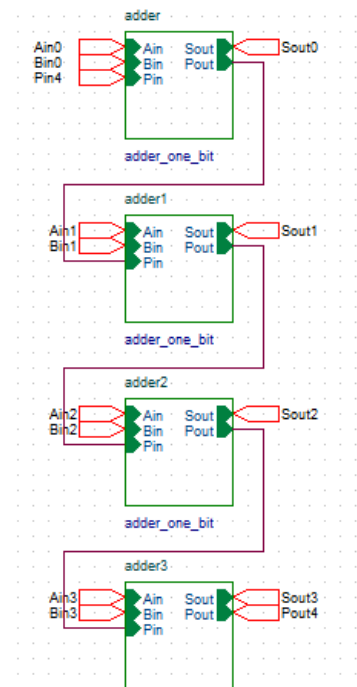


Fig. 2.6.14 – Internal structure of a four-bit adder macro

MonteCarlo...	adder_test...	adder/adder	adder	Start Page	adder_test : ...
New Property...	Apply	Display...	Delete Property	Pivot	Filter by: < Current properties > Help
		Color	COMMAND1	COMMAND2	COMMAND3
1	adder_test: adder_test_page : DSTM8	Default	0s 0000	5us 0001	10us 0010
2	adder_test: adder_test_page : DSTM10	Default	0s 0000	5us 0001	10us 0010

Fig. 2.6.15 – Setting up input signal generators

After that, we perform the previously described steps to simulate the resulting device using the Monte Carlo method: the PDF graph is shown in Fig. 2.6.16, the CDF graph is shown in Fig. 2.6.17.

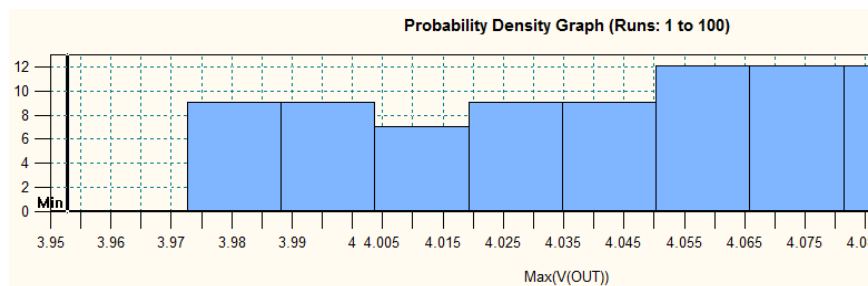


Fig. 2.6.16 – Graph of the probability distribution function

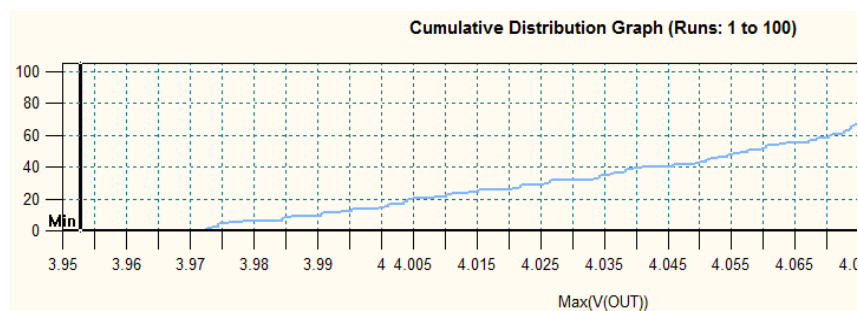


Fig. 2.6.17 – Graph of the integral distribution function

From Fig. 2.6.16 it is seen that the distribution of the maximum voltage is shifted towards the maximum mark. At the same time, the fact that the integral distribution function has a rather large angle of rise becomes obvious. From

this observation it can be concluded that due to various phenomena (for example, due to the "racing" of signals) the voltage values change over time. In this case, delays cause the appearance of random unwanted signals at different times, which is reflected in the graph of the probability distribution function.

Tasks for practical work

Perform Monte Carlo simulation of the given device according to Table 2.7.

Table 2.7

№	Device
1	4-bit adder
2	6OR
3	5NAND
4	encoder 8×3
5	3NOR
6	multiplexer 5×1
7	5NOR
8	multiplexer 6×1
9	6-bit adder
10	<i>D</i> flip-flop
11	<i>RS</i> flip-flop
12	<i>JK</i> flip-flop
13	4-bit register
14	6-output decoder
15	decoder for 10 outputs

Practical work 7

PCB TRACING

Objective: to acquire practical skills in creating, editing, and tracing printed circuit boards in OrCAD using the Allegro PCB editor.

Example of task execution

PCB (*printed circuit board*) *tracing* is a step-by-step process of laying out conductors in CAD. printed circuit boards. As a rule, tracing is the final stage of design and consists in defining the lines connecting the contacts of elements and components. OrCAD uses the *Allegro* component to trace circuits assembled in OrCAD Capture.

To create and trace a PCB you must perform the steps:

- 1) assemble the device diagram and assign the properties of its elements necessary for tracing;
- 2) perform annotation operations, rule checking, and connection list creation;
- 3) place the components on the board and trace the lines connecting them.

The design of any device begins with a schematic diagram – a graphical model that demonstrates the elements of the device and the relationships between them using GS. This diagram does not show the physical structure of the device, but only shows the logical structure and relationship between the elements.

Let us consider all the actions using the example of a 4-bit adder (Fig. 2.7.1). In Fig. 2.7.2 shows the GS of the element 7408 (element AND);

- on top, the element is signed *U6A*, which corresponds to the name of a specific element on the diagram;
- the inscription 7408 at the bottom is the type of element, the name of the microcircuit that implements it;
- numbers 1, 2, 3 near the element's pins are the numbers of the microcircuit pins that correspond to these inputs.

A conventional image of the internal structure of the 7408 chip is shown in Fig. 2.7.3. In some elements of the same type, the names coincide in the first two characters, and their inputs and outputs are numbered consecutively. This means that in a real circuit they are implemented inside one chip. Therefore,

for the correct placement of elements on a printed circuit board in the form of chips, it is necessary to specify the shape of the seat for each of the elements.

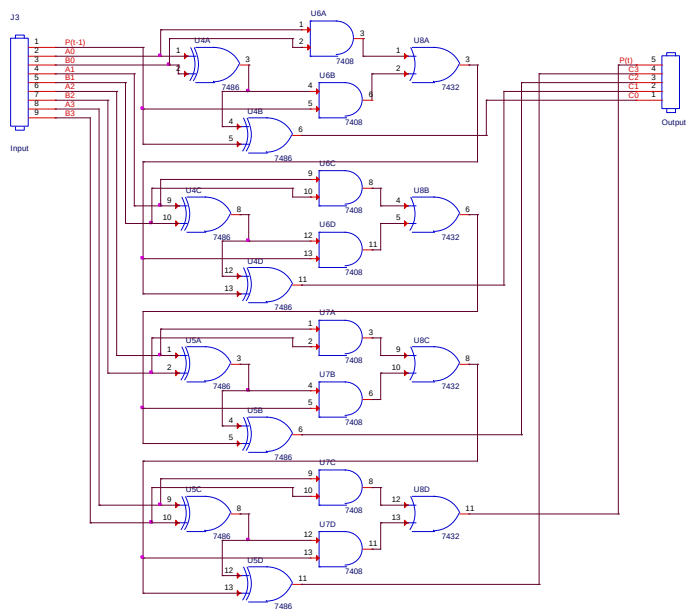


Fig. 2.7.1 – Adder scheme

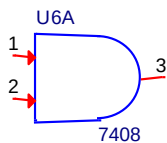


Fig. 2.7.2 – GS of element AND

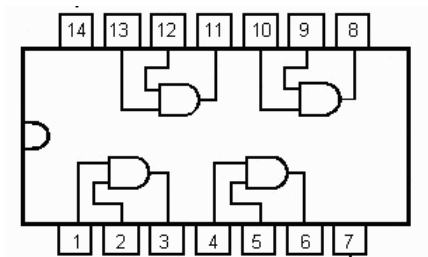


Fig. 2.7.3 – 7408 chip

The *PCB footprint* is set by the parameter of the same name in the element properties. To change it, right-click on the element and select *Edit Properties*, and then edit the *PCB Footprint* column (Fig. 2.7.4).

OrCAD Capture CIS - [Prop]

File Design Edit View Tools Place SI Analysis Macro Accessories Options Window Help

7408
SCHEMATIC1-WDWD

Routing.opj PAGE1 SCHEMATL...

New Property... Apply Display... Delete Property Pivot Filter by: < Current properties >

		MNTYMXDLY	Name	Part Reference	PCB Footprint	
1	+	SCHEMATIC1 : PAGE1	0	INS13602	U6B	DIP14_3
2	+	SCHEMATIC1 : PAGE1	0	INS13662	U6C	DIP14_3
3	+	SCHEMATIC1 : PAGE1	0	INS13725	U6D	DIP14_3
4	+	SCHEMATIC1 : PAGE1	0	INS13542	U6A	DIP14_3
5	+	SCHEMATIC1 : PAGE1	0	INS13847	U7B	DIP14_3
6	+	SCHEMATIC1 : PAGE1	0	INS13906	U7C	DIP14_3
7	+	SCHEMATIC1 : PAGE1	0	INS13969	U7D	DIP14_3
8	+	SCHEMATIC1 : PAGE1	0	INS13784	U7A	DIP14_3

Fig. 2.7.4 – Assignment of element locations

Elements of the same type can be edited together (to select multiple elements at once, hold down the *Ctrl* key). When editing in this way, you can copy the value of one element in a column, and for the rest, "stretch" the cell borders beyond the corner. However, this approach does not apply to elements of different types, as it leads to name conflicts.

The value of the editable column for the logic elements used in the circuit is DIP14_3, and for the connectors – *JumperN*, where *N* is the number of connector inputs. The connectors are located in the *CONNECTOR* library. After setting the required properties for all elements, you can proceed to the next stage. Now the operations will no longer be performed with individual elements, but with the circuit as a whole.

Save all changes and close the schematic pages. Further operations will be performed in the *File* tab of the schematic project window. First, you need to make the designed schematic the root one (indicated by the "/" sign). To do this, right-click on the schematic and select *Make root* (Fig. 2.7.5).

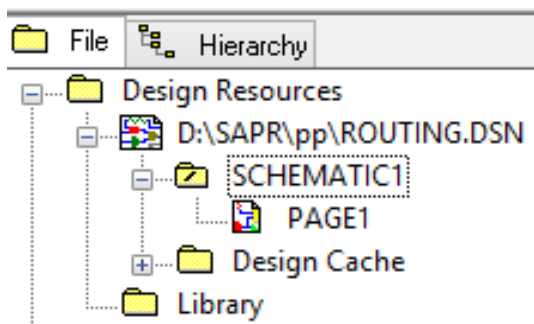


Fig. 2.7.5 – Assigning the root directory

Before transferring the schematic to the PCB editor, it is necessary to generate additional information about it using the following operations:

- annotation (placing position numbers);
- checking the rules;
- compiling a list of connections.

These operations are performed using the commands in the *Tools drop-down menu* (Fig. 2.7.6).

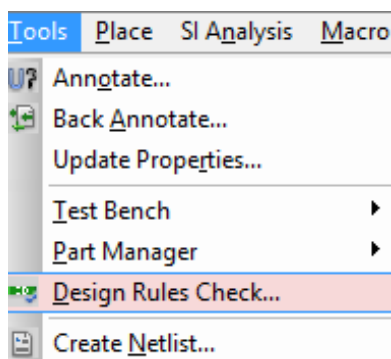


Fig. 2.7.6 – Tools menu

The first operation – placing position numbers is performed using the *Annotate command*. It consists in assigning a unique number to each element of the project. The operation parameters setting window is shown in Fig. 2.7.7.

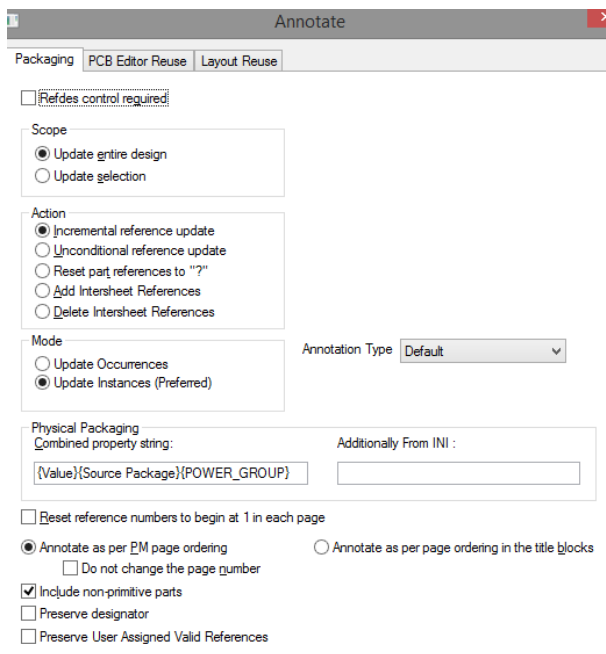


Fig. 2.7.7 – Annotation command window

The values of the main parameters:

Update entire design – number the elements for the project;

Update selection – number the elements for the selected page;

Incremental reference update – element numbering is performed in ascending order;

Unconditional reference update – all numbers are viewed, missing ones are determined and then in ascending order;

Reset part reference to "?" – reset numbering;

Add/Delete Intersheet References – add/delete page number references;

Physical Packaging – specifies the properties of combining elements into one chip;

Reset reference numbers to begin at 1 inpage schematic – numbers elements starting from one in each schematic catalog;

Do not change the page number – schematic pages are not renumbered.

Numbering is done according to the finished project. You can use standard parameters for this. Before making changes to the project, it is recommended to reset the numbering by selecting the *Reset part reference to "?"* action in the annotation window.

Before performing the operation, a request is made to confirm the numbering. To continue executing the command click *OK* (Fig. 2.7.8).

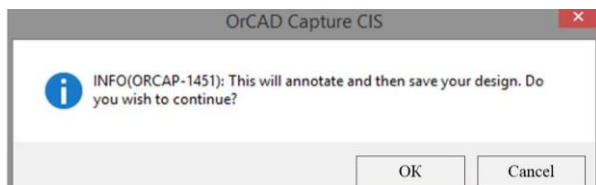


Fig. 2.7.8 – Request for confirmation of instruction

After numbering, it is necessary to check the diagram for errors and compliance with the rules. To do this, use the *Design Rules Check* menu item. When you call it, a window opens (Fig. 2.7.9), in which the check parameters are set.

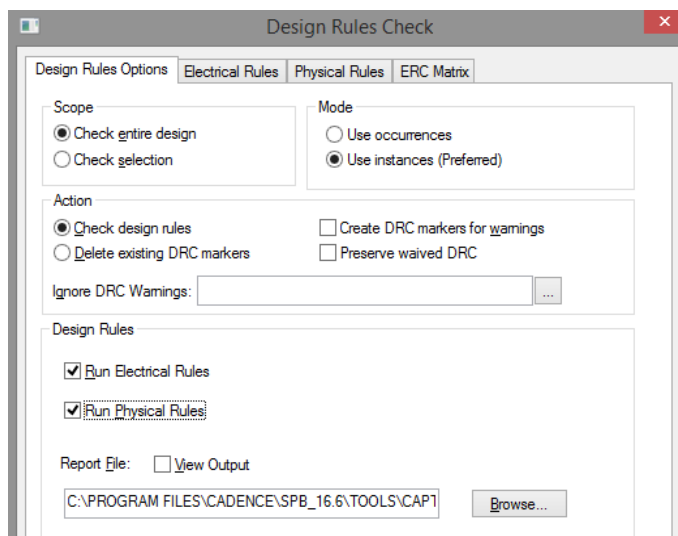


Fig. 2.7.9 – Rules Check Window

The first two groups of options in the first tab are similar to the previous command. The third group defines the common actions of the command:

Check design rules – check for compliance with electrical rules;

Delete existing DRC markers – delete existing marks;

Create DRC markers for warnings – create markers for warnings. The program necessarily places a marker for errors identified by the ERC matrix (if you select this option, it will also create markers for warnings).

The fourth group of options specifies which rules will be checked – electrical and/or physical. Depending on the selection of these options, you can config the parameters of the next two tabs, which determine how exactly these rules will be checked (Fig. 2.7.10 – 2.7.11).

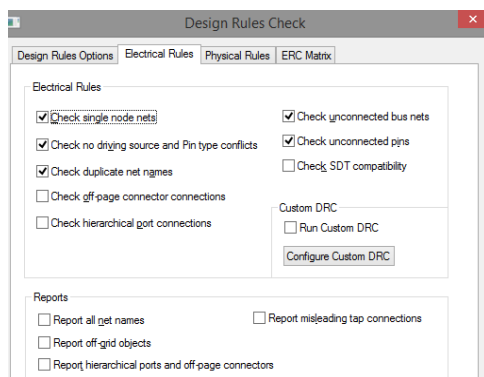


Fig. 2.7.10 – Electrical rules setup window

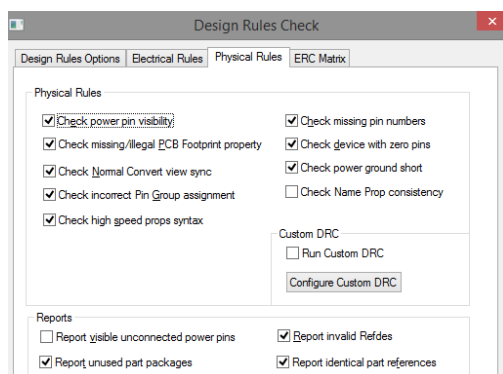


Fig. 2.7.11 – Physical rules configuration window

To check the project, select the electrical and physical rules check. To do this, check the *Run Electrical Rules* and *Run Physical Rules items*, and leave the other parameters at default.

If there are errors, we receive a message (Fig. 2.7.12), which suggests displaying the errors in the session log, which is presented at the bottom of the window.

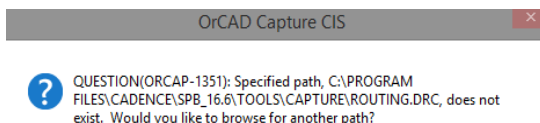


Fig. 2.7.12 – Error message

The last operation, creating a netlist, is most directly related to tracing elements on a printed circuit board. This command creates a list containing a description of the connections between parts of the circuit. It is called through the *Create Netlist item in the Tools* menu.

The window that opens with this command contains different tabs, depending on what this list is intended for since we are working with PCB Editor, we select the first tab (Fig. 2.7.13).

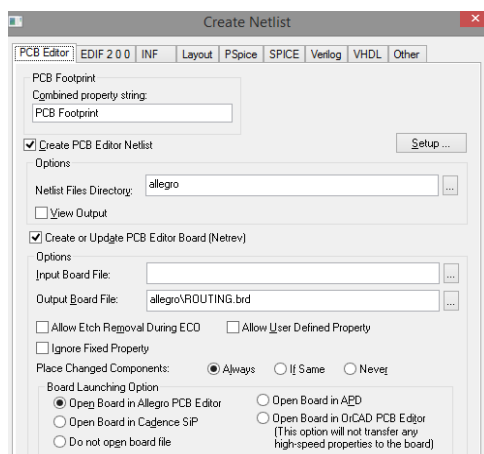


Fig. 2.7.13 – Connection list creation window

Here in the second group you can define the directory where the list will be placed (*Netlist Files Directory*). Then you need to check the *Create or Update PCB Editor Board* item to create the board. After that, its parameters are set:

Input Board File – the file that will be taken as a basis (optional parameter);

Output Board File – the file into which the created board will be placed.

The *Board Launching Option* group provides options for programs that will open the board. Since we are working with PCB Editor, we select the *Open Board in Allegro PCB Editor* item.

After filling in the parameters (in our case *Create or Update PCB Editor Board*), click *OK* to create a list of PCB connections and open it in the editor. A window will open that demonstrates the process of creating the list and updating the board (Fig. 2.7.14).

In case of errors, a corresponding message will be displayed, and the errors themselves are listed in the session log. If everything is successful, PCB Editor will start.

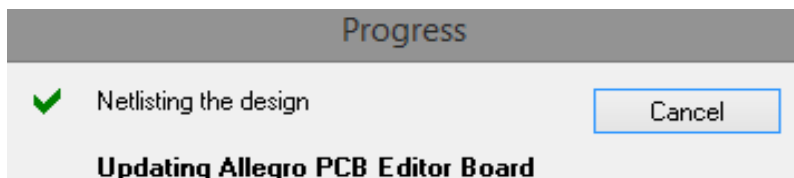


Fig. 2.7.14 – Process of creating a list of connections and boards

Now we move on to the third stage of creating a printed circuit board. After successfully creating the connection list, PCB Editor automatically starts and a window appears with options for which product to use (Fig. 2.7.15). In our case, Allegro PCB SI XL is suitable click *OK*.

To change the color scheme in PCB Editor, you need to select the menu item *Display* → *Color/Visibility*.

The background color can be made white, as shown in Fig. 2.7.16. For greater clarity, to change the color of the board in the *Color Dialog* window in the *Conductor* tab, you can change the colors of the trace lines and points.

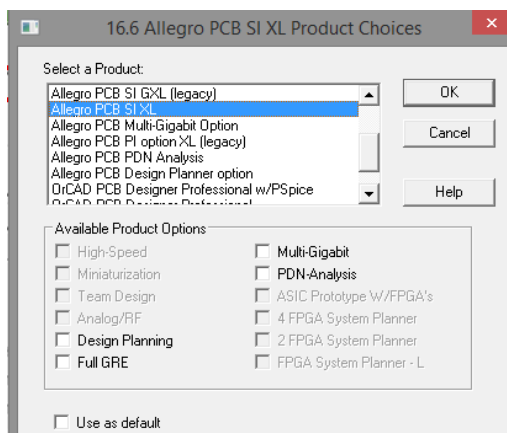


Fig. 2.7.15 – Product selection

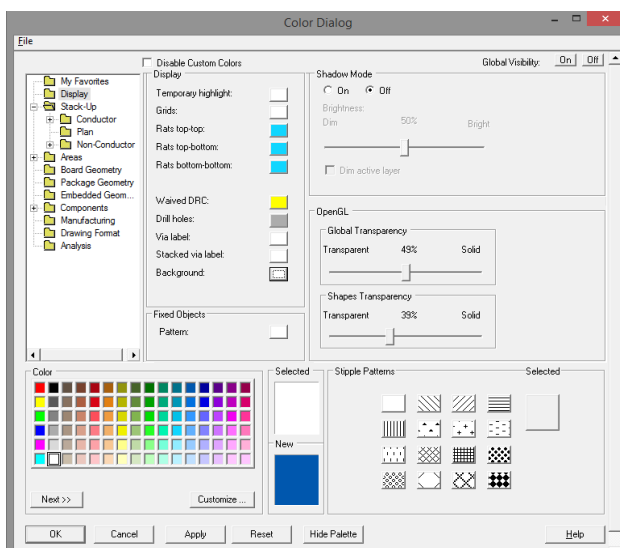


Fig. 2.7.16 – Color scheme change window

After selecting the product, the PCB Editor window will appear, in which you need to perform the final stage of creating a printed circuit board. The following stages must be performed: setting the circuit board contours, placing components and tracing conductors. In the *Setup\Design Parameters* menu item, select the necessary parameters in the *Size block* (Fig. 2.7.17).

Before placing components, you need to create a board on which they can be placed, to do this, select the *Setup menu item*, submenu *Outlines*, item *Board Outline*. The window that opens (Fig. 2.7.18) accompanies the process of drawing the board boundaries left-click on the workspace twice to set the upper left and lower right corners. After completing the drawing, select *Close* in the *Board Outline* window. We will get the boundaries of the printed circuit board, which we can now work (Fig. 2.7.19).

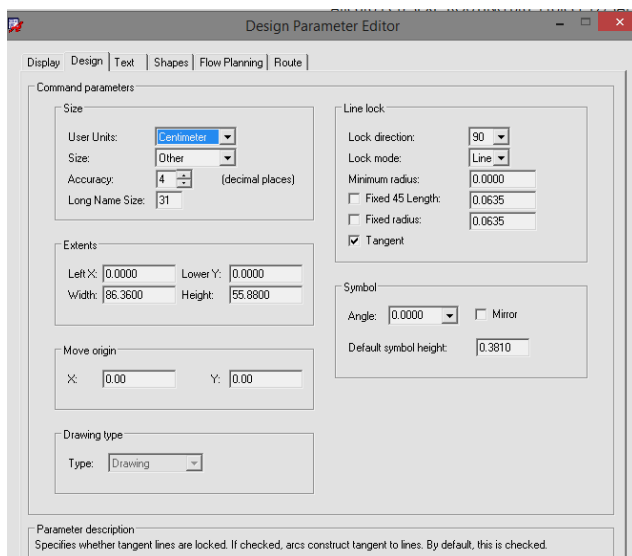


Fig. 2.7.17 – Selecting board parameters

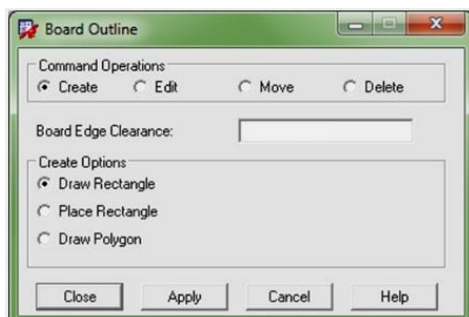


Fig. 2.7.18 – Window accompanying the drawing of board boundaries

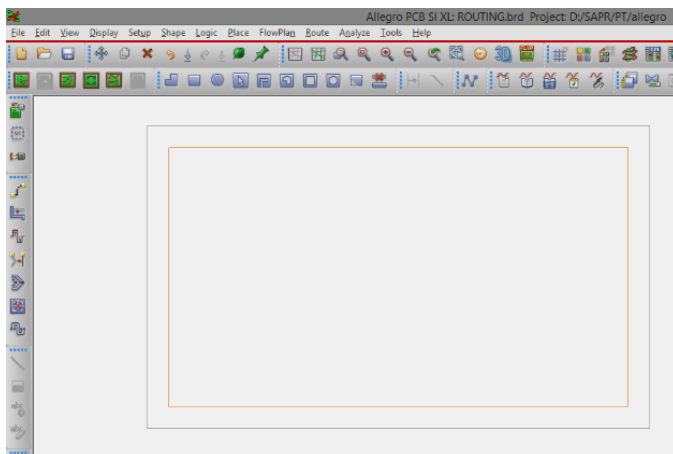


Fig. 2.7.19 – Obtained board boundaries

There are two ways to place elements on the board: automatic and manual – they are located in the *Place* menu item (Fig. 2.7.20).

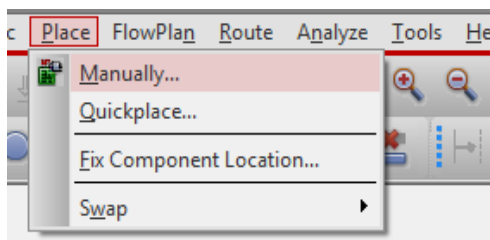


Fig. 2.7.20 - *Place* menu

For automatic placement, select the *Quickplace* item. The corresponding window will open (Fig. 2.7.21), in which you can set the placement parameters.

The first group determines which elements need to be placed and offers a set of filters. To place all components, select the *Place all components* item. The second group determines where the elements will be placed. The placement layer and boundaries are set here. After setting the parameters, you must click the *Place* button to cancel the application *Unplace*. If the result satisfies us, click *OK* (Fig. 2.7.22).

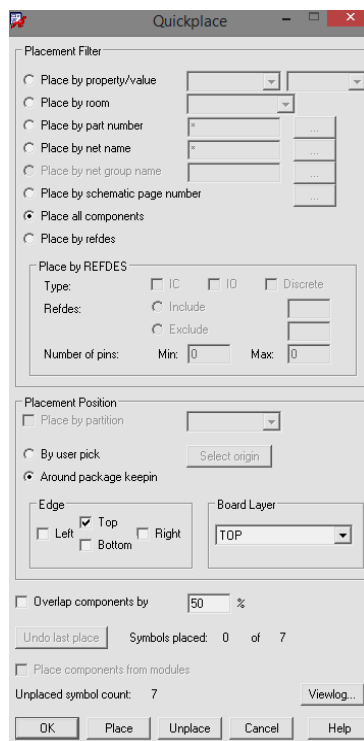


Fig. 2.7.21 – QuickPlace window

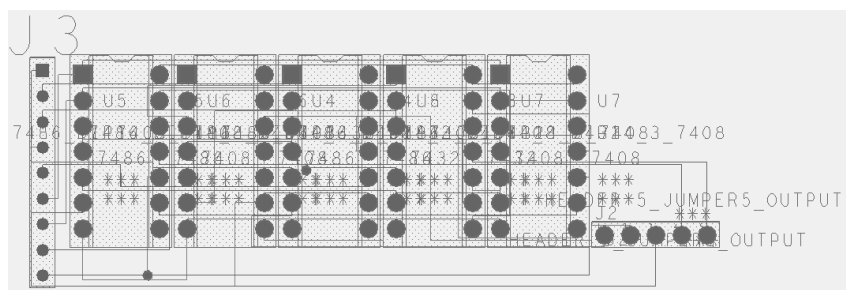


Fig. 2.7.22 – Placed elements

After placing the components, the initial board size turns out to be not optimal. To edit its boundaries, we use the Edit mode (Fig. 2.7.23).

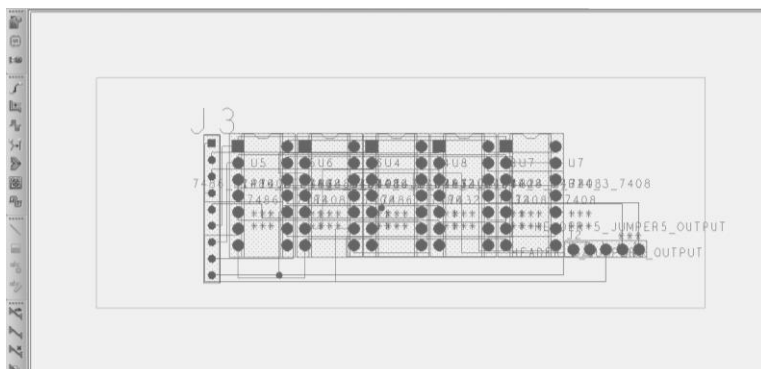


Fig. 2.7.23 – Element placement after editing board boundaries

To manually place elements, select the menu item *Manually*. The *Placement window* will open (Fig. 2.7.23), in which there will be a list of device elements on the left.

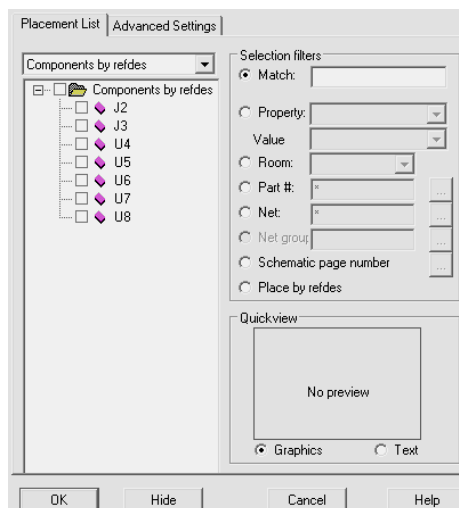


Fig. 2.7.24 – Manual placement window

To place an element, check the box next to its name and left-click to place it on the board. After placing it, click *OK*.

The first group of menu commands is intended for manual placement of elements. To adjust, you need to select the link with the left-click, then select the menu item with the corresponding command and outline the path.

When finished, right-click and select *Done*:

- the *Connect* command allows you to connect points by specifying a trajectory;

- *Slide* moves objects, including laid links;

- *Delay Tune* adds bends to the lines that add delay;

- *Phase Tune* adds bends that cause phase shift;

- *Custom Smooth* smoothes lines.

Placing conductors is a complex mathematical task that takes a lot of time and effort from the user, without guaranteeing optimality.

Automatic tracing is performed through the *Route* menu item, *PCB Router submenu*, *Route Automatic* item. After selecting this command, the automatic tracing window will open (Fig. 2.7.27), in which you can specify some options for the conductor tracing algorithm. After that, you must click the *Route* button to perform the tracing. You can also click *Undo* to cancel the tracing and *Close* to close the window. The result of automatic tracing is shown in Fig. 2.7.28.

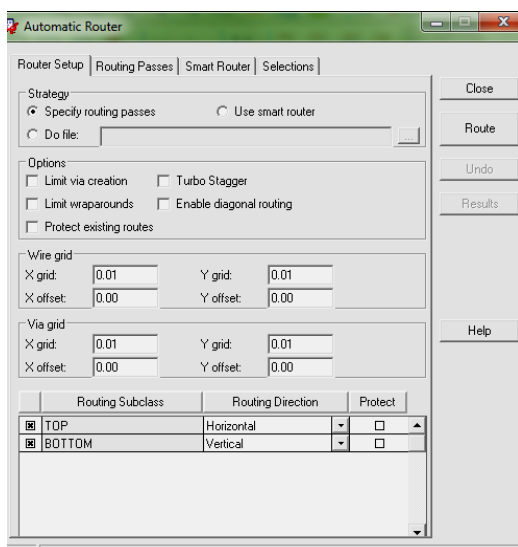


Fig. 2.7.27 – Automatic tracing window

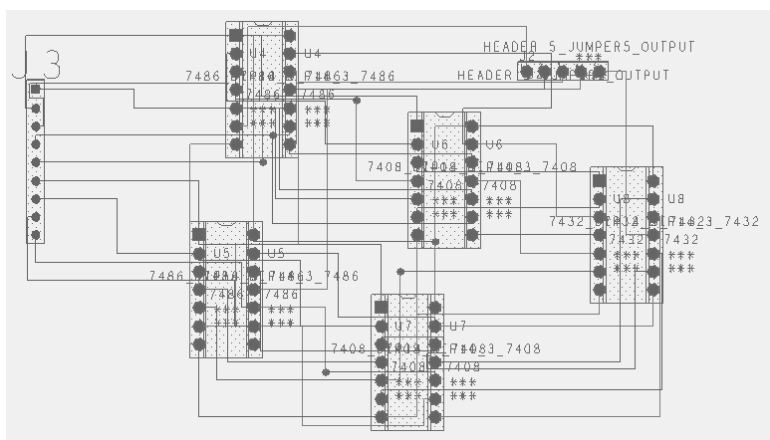


Fig. 2.7.28 – Result of circuit tracing

Manual chip placement and tracing are useful, but they are time-consuming, suboptimal, and can lead to errors. The result of fully automatic tracing is shown in Fig. 2.7.29.

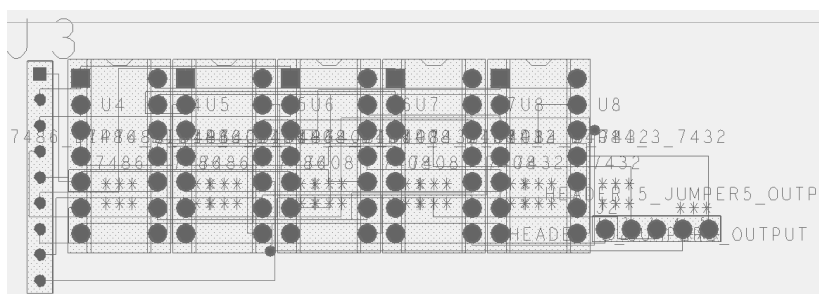


Fig. 2.7.29 – Result after autotracing

Tasks for practical work

Perform PCB tracing for the device developed in practical work № 1 in system OrCAD.

Practical work 8

RESEARCH ON THE OPERATION OF DEVICES TAKING INTO ACCOUNT ELECTROMAGNETIC COMPATIBILITY

Purpose of the work: To conduct a study of the device to evaluate its performance taking into account electromagnetic compatibility.

Example of task execution

As a device for research, we choose the 12→1 encoder (Fig. 2.8.1), the simulation results of which are shown in Fig. 2.8.2.

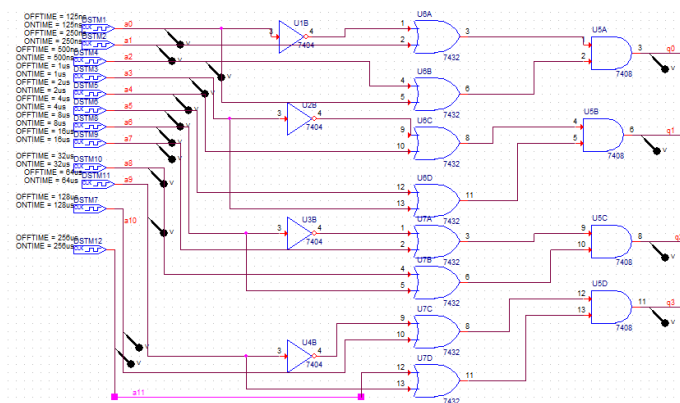


Fig. 2.8.1 – Scheme of the encoder

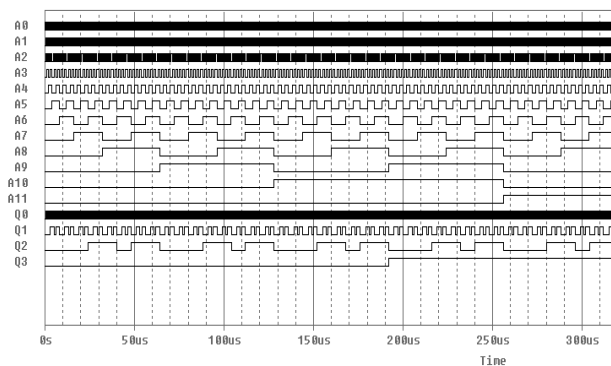


Fig. 2.8.2 – Simulation results

We will replace the generators at the input of the circuit with a 12-bit connector, and at the output we will put a 4-bit connector (Fig. 2.8.3).

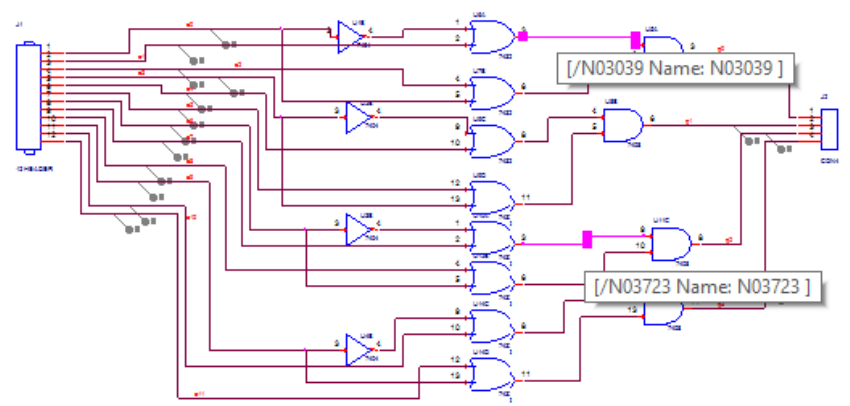


Fig. 2.8.3 – Scheme of the encoder with connectors

We set the value of the PCB Footprint parameter for all circuit elements, as shown in Fig. 2.8.4 (we assign the type DIP14_3 to the logical elements, Jumper to the connectors, the other parameters are standard).

encoder_t...*

PAGE1*

encoder_r...*

New Property...

Apply

Display...

Delete Property

Pivot

Filter by: < Current properties >

Help

		PCB Footprint	Implementation	Designator	Color	Part Reference
1	encoder_routing : PAGE1 : J1	Jumper4			Default	J1
2	encoder_routing : PAGE1 : J3	Jumper12			Default	J3
3	encoder_routing : PAGE1 : U32	DIP14_3	7408	C	Default	U32C
4	encoder_routing : PAGE1 : U32	DIP14_3	7408	D	Default	U32D
5	encoder_routing : PAGE1 : U32	DIP14_3	7408	A	Default	U32A
6	encoder_routing : PAGE1 : U32	DIP14_3	7408	B	Default	U32B
7	encoder_routing : PAGE1 : U33	DIP14_3	7404	C	Default	U33C
8	encoder_routing : PAGE1 : U33	DIP14_3	7404	D	Default	U33D
9	encoder_routing : PAGE1 : U33	DIP14_3	7404	B	Default	U33B
10	encoder_routing : PAGE1 : U33	DIP14_3	7404	A	Default	U33A
11	encoder_routing : PAGE1 : U34	DIP14_3	7432	B	Default	U34B
12	encoder_routing : PAGE1 : U34	DIP14_3	7432	C	Default	U34C
13	encoder_routing : PAGE1 : U34	DIP14_3	7432	D	Default	U34D
14	encoder_routing : PAGE1 : U34	DIP14_3	7432	A	Default	U34A
15	encoder_routing : PAGE1 : U35	DIP14_3	7432	C	Default	U35C
16	encoder_routing : PAGE1 : U35	DIP14_3	7432	D	Default	U35D
17	encoder_routing : PAGE1 : U35	DIP14_3	7432	B	Default	U35B
18	encoder_routing : PAGE1 : U35	DIP14_3	7432	A	Default	U35A

Fig. 2.8.4 – Setting the PCB Footprint parameter value

During annotation and annotation system setup, the program automatically assigns names to all conductors that are used later for tracing, resulting in successful completion of the operation (Fig. 2.8.5).

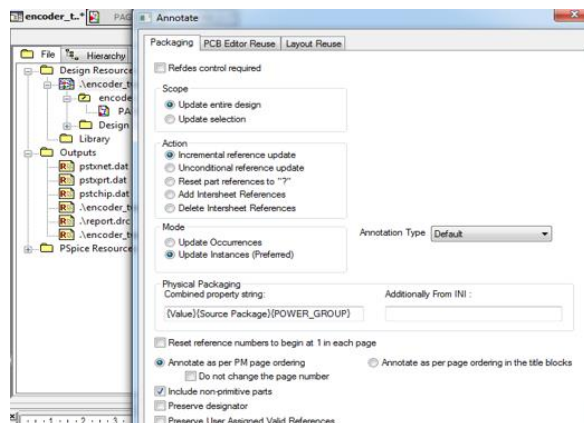


Fig. 2.8.5 – Generating annotation for device diagram

To perform a check of electrical and physical rules, the parameters of the check procedure are set (Fig. 2.8.6).

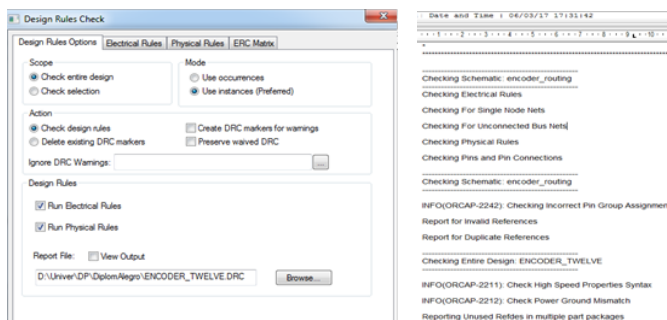


Fig. 2.8.6 – Checking rules

Now we are building a list of conductors and connections (Fig. 2.8.7), after which we will be able to select the Allegro PCB SI XL program for further work.

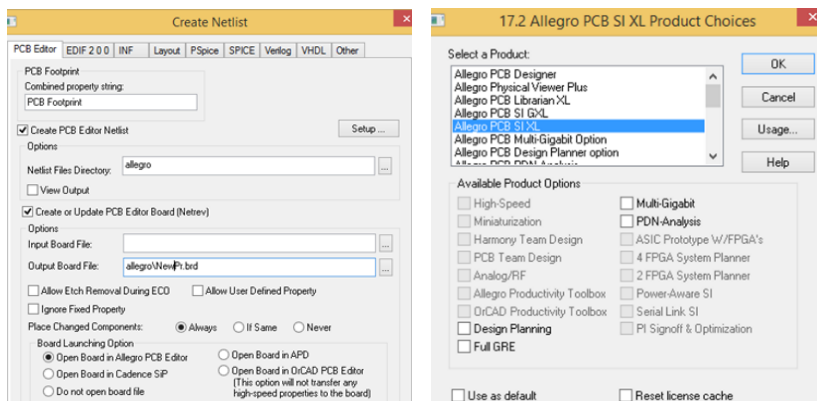


Fig. 2.8.7 – Compiling a list of project connections and selecting a program

To place elements on the printed circuit board, select the *Place* → *Manually interface* section. After that, a window appears in which we select each element string in turn and install it on the board (Fig. 2.8.8).

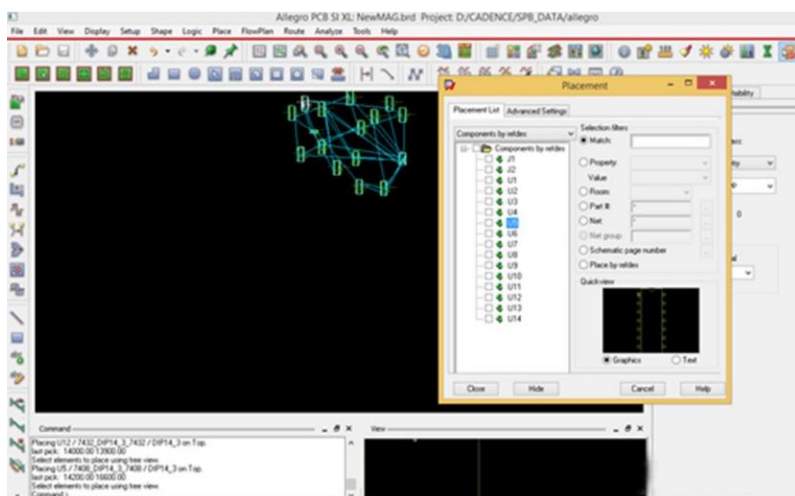


Fig. 2.8.8 – Placement of elements on the board

Now we set the conductor thickness to 1.2 milliinches, the material to be copper. We choose fiberglass as the dielectric material (Fig. 2.8.9).

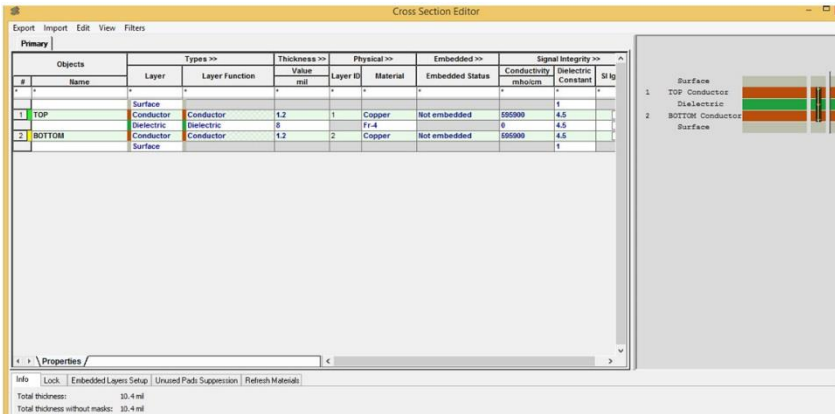


Fig. 2.8.9 – Specifying the thickness and type of material

Now we set the minimum allowable distance between the conductors (Fig. 2.8.10).

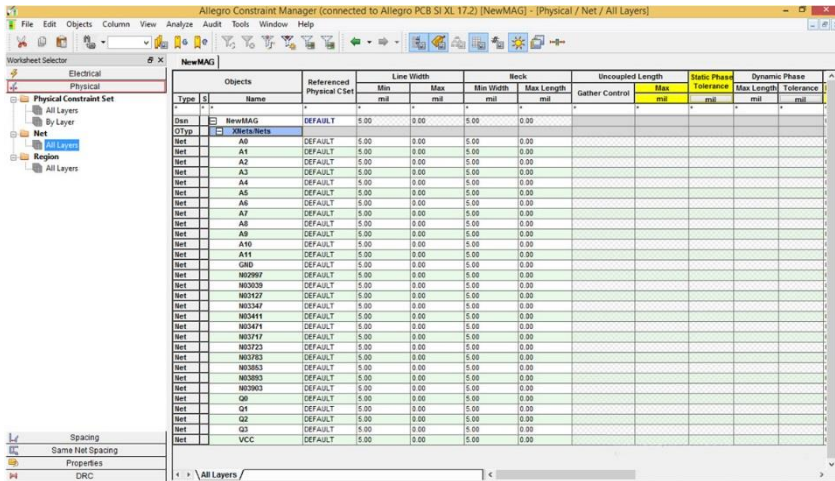


Fig. 2.8.10 – Setting the minimum distance between conductors

We perform automatic tracing using the *Route → Route Automatic* command. The tracing process is shown in Fig. 2.8.11, and its result is shown in Fig. 2.8.12.

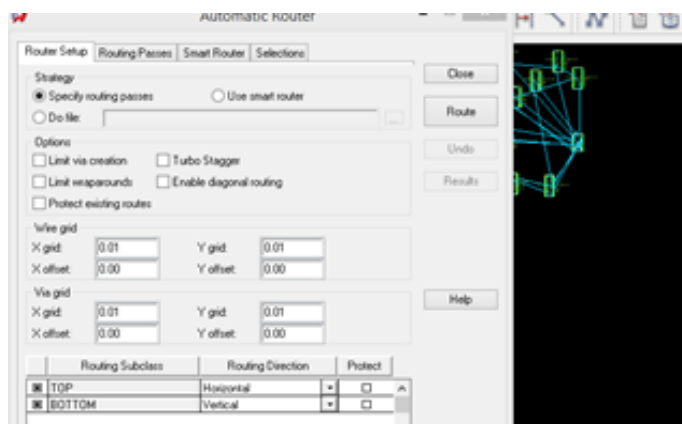


Fig. 2.8.11 – Running a trace

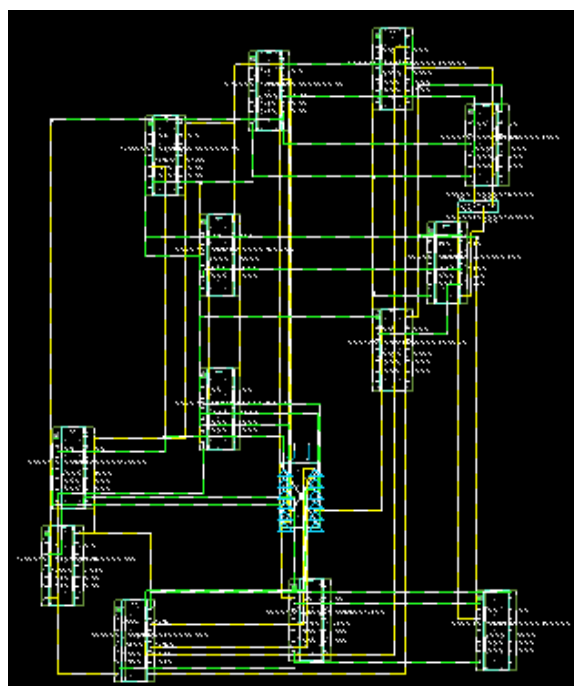


Fig. 2.8.12 – Trace result

We specify the required voltage values for VCC and GND (*Logic → Identify DC Nets*) – we set the voltages to 5 V and 0 V (Fig. 2.8.13).

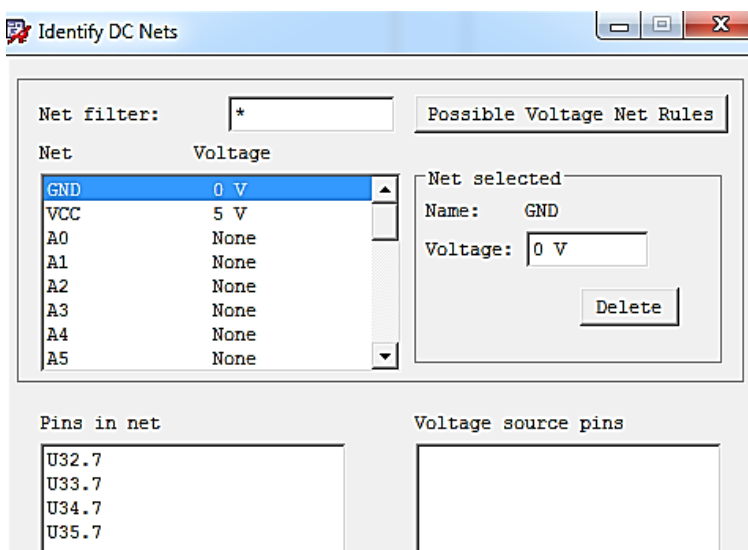


Fig. 2.8.13 – Setting voltages

IBIS-models are used to analyze signal integrity and crosstalk in digital printed circuit boards. The IBIS format is a format for the external description of an electronic device without taking into account its internal structure and operating characteristics.

The parameters of IBIS-models are obtained on the basis of the current-voltage characteristics for different logic states of the DC pins, parasitic case parameters and transient characteristics on a resistive load. We create an IBIS model for each element of the scheme: *Analyze → Model Assignment* (Fig. 2.8.14).

After completing the settings and performing the tracing, the conductors that will act as *slave* and *master were selected*.

Master conductor is the conductor that is the signal source, in this study it is conductor N03723 (Fig. 2.8.15). The *slave conductor* is the conductor that is the signal receiver (in this study it is conductor N03039).

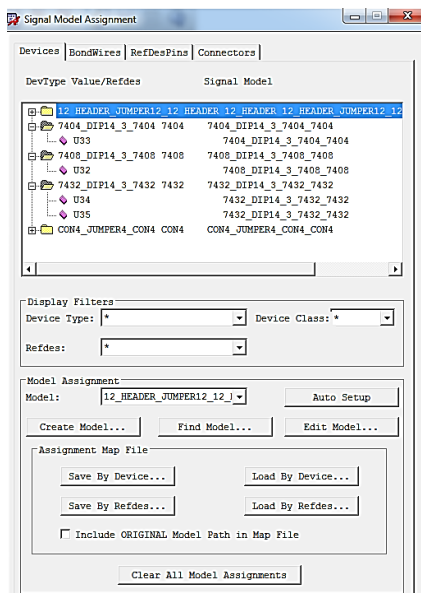


Fig. 2.8.14 – Creating IBIS models for board elements

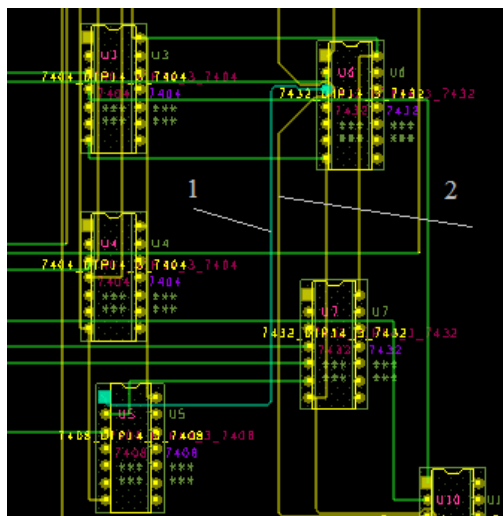


Fig. 2.8.15 – Selection of conductors for research (1 – slave conductor, 2 – master conductor)

Study of the device operation with different conductor lengths

Let's study the change in the value of the crosstalk voltage between two parallel conductors under the condition of changing the length of their adjacent parts.

First, we will measure the conductors and set their lengths (4344 milliinches). We will build a crosstalk report for the *slave conductor* for the given length using the *Analys → Probe interface section*, to do this, in the window, select the conductor with the number corresponding to the *slave conductor number* (N03039), and select the *Create report* item.

In the new window, set the following checkboxes: *Crosstalk Detailed*; simulation speed *Slow*, *Highlighted Net Only* (currently highlighted conductor); *Each Neighbor* – the neighboring conductor is the *master*; *Pulse – supply of pulsed pseudo-random signals to the input of the master conductor* (Fig. 2.8.16).

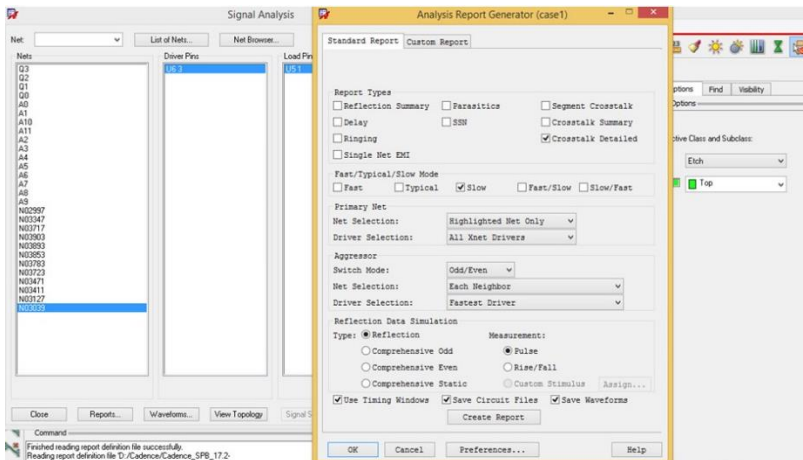


Fig. 2.8.16 – Selecting parameters for creating a crosstalk report

In the received report, the value of the crosstalk voltage is located in the *Single Neighbor Crosstalk* section. The resulting values of the interference voltage (Fig. 2.8.19) are entered in Table 2.8.

```

*****
Single Neighbor Crosstalk at Receivers (mV) (Slow FTSMode)
*****
Victim XNet  Victim Drvr  Victim Rcvr  HSOddXtalk  HSEvenXtalk  LSOddXtalk  LSEvenXtalk  Aggr XNet  Aggr Drvr
*****
NEWDRP N03039  NEWDRP U6 3  NEWDRP U5 1  824.4      879.5      762.4      931.1      NEWDRP N03723  NEWDRP U10 3
*****

```

Fig. 2.8.17 – Crosstalk voltage values for a length of the adjacent part of the conductors of 4344 milliinches

Table 2.8 – Simulated crosstalk situations

Cross obstacle	Conditions of occurrence crosstalk		Value, mV
	Slave logic level	Master transition	
HSOddXtalk	HI	HI – LO	824.4
HSEvenXtalk	HI	LO – HI	879.5
LSOddXtalk	LO	LO – HI	726.4
LSEvenXtalk	LO	HI – LO	931.1

The maximum voltage value in this report is LSEvenXtalk, which is 931.1 mV when the *slave signal logic level* is zero (0 V) and *the master* switches from one to zero (from 5 V to 0 V).

Next, we perform similar actions provided that the length of the adjacent part of the conductors is reduced to 2020, 1620, 1050, 630, 320, 300 milliinches. Reports on these studies are shown in Fig. 2.8.18 – 2.8.23.

The values of the crosstalk voltages obtained for the length of the adjacent part of the conductors of 2020 milliinches are given in Table 2.9.

```

*****
Single Neighbor Crosstalk at Receivers (mV) (Slow FTSMode)
*****
Victim XNet  Victim Drvr  Victim Rcvr  HSOddXtalk  HSEvenXtalk  LSOddXtalk  LSEvenXtalk  Aggr XNet  Aggr Drvr
*****
NEWDRP N03039  NEWDRP U6 3  NEWDRP U5 1  729.6      744.4      668.8      847.3      NEWDRP N03723  NEWDRP U10 3
*****

```

Fig. 2.8.18 – Crosstalk voltage values for a length of the adjacent part of the conductors of 2020 milliinches

The maximum voltage value in this report is LSEvenXtalk (847.3 mV) when the *slave signal logic level* is zero (0 V) and *the master* switches from one to zero (from 5 V to 0 V).

Table 2.9 – Simulated crosstalk situations

Cross obstacle	Conditions of occurrence crosstalk		Value, mV
	Slave logic level	Transition master	
HSOddXtalk	HI	HI – LO	729.6
HSEvenXtalk	HI	LO – HI	744.4
LSOddXtalk	LO	LO – HI	668.8
LSEvenXtalk	LO	HI – LO	847.3

```

*****
Neighbor Crosstalk at Receivers (mV) (Fast/Slow FTSMode)
*****
XNet   Victim Drvr  Victim Rovr  HSOddXtalk  HSEvenXtalk  LSOddXtalk  LSEvenXtalk  Aggr XNet    Aggr Drvr
-----
N03039 NEWDR U6 3   NEWDR U5 1   414.4       309.5        360.3       362.9        NEWDR N03723 NEWDR U10 3
-----

```

Fig. 2.8.19 – Values of crosstalk voltages at length adjacent part of the conductors 1620 milliinches

The values of crosstalk voltages for a length of the adjacent part of the conductors of 1620 milliinches are given in Table 2.10.

Table 2.10 – Simulated crosstalk situations

Cross obstacle	Conditions of occurrence crosstalk		Value, mV
	Slave logic level	Master transition	
HSOddXtalk	HI	HI – LO	414.4
HSEvenXtalk	HI	LO – HI	309.5
LSOddXtalk	LO	LO – HI	360.3
LSEvenXtalk	LO	HI – LO	362.9

The maximum voltage value in this report is HSOddXtalk (414.4 mV) when the *slave signal logic level* is one (5 V) and the *master* switches from one to zero (from 5 V to 0 V).

```

*****
Single Neighbor Crosstalk at Receivers (mV) (Slow FTSMode)
*****
Victim XNet    Victim Drvr    Victim Rvrv    HSOddXtalk    HSEvenXtalk    LSOddXtalk    LSEvenXtalk    Aggr XNet    Aggr Drvr
-----
NEWDRP N03039  NEWDRP U6 3  NEWDRP U5 1  341.8        305.8          343.7         345.6         NEWDRP N03723  NEWDRP U10 3
*****
*****
Driver I/O Characteristics (Slow FTSMode) RiseSlew/FallSlew in (mV/ns)
*****
*****
Driver          Device          IOModel          Volmax          Vohmin          RiseSlew          FallSlew          JTemp          DiffPairMatch
-----
NEWDRP U6 3     7432_D1P14_3_7432  CDSDefaultOutput_2p8V  NA              NA              3333           3333           NA              NA
*****

```

Fig. 2.8.20 – Values of crosstalk voltages at length adjacent part of the conductors 1050 milliinches

The values of crosstalk voltages for a length of the adjacent part of the conductors of 1050 milliinches are given in Table 2.11.

Table 2.11 – Simulated crosstalk situations

Cross obstacle	Conditions of occurrence crosstalk		Value, mV
	Slave logic level	Transition master	
HSOddXtalk	HI	HI – LO	341.8
HSEvenXtalk	HI	LO – HI	305.8
LSOddXtalk	LO	LO – HI	343.7
LSEvenXtalk	LO	HI – LO	345.6

The maximum voltage value in this report is LSEvenXtalk (345.6 mV) when the *slave signal logic level* is zero (0 V) and the *master* switches from one to zero (from 5 V to 0 V).

```

*****
Single Neighbor Crosstalk at Receivers (mV) (Slow FTSMode)
*****
Victim XNet    Victim Drvr    Victim Rvrv    HSOddXtalk    HSEvenXtalk    LSOddXtalk    LSEvenXtalk    Aggr XNet    Aggr Drvr
-----
NEWDRP N03039  NEWDRP U6 3  NEWDRP U5 1  72.95        51.46          56.11         80.78         NEWDRP N03723  NEWDRP U10 3
*****

```

Fig. 2.8.21 – Values of crosstalk voltages at length adjacent part of the conductors 630 milliinches

The values of crosstalk voltages for a length of the adjacent part of the conductors of 630 milliinches are given in Table 2.12.

Table 2.12 – Simulated crosstalk situations

Cross obstacle	Conditions for crosstalk		Value, mV
	Slave logic level	Transition master	
HSOddXtalk	HI	HI – LO	72.95
HSEvenXtalk	HI	LO – HI	51.46
LSOddXtalk	LO	LO – HI	56.11
LSEvenXtalk	LO	HI – LO	80.78

The maximum voltage value in this report is LSEvenXtalk (80.78 mV) when the *slave signal logic level* is zero (0 V) and *the master* switches from one to zero (from 5 V to 0 V).

```

.....
Single Neighbor Crosstalk at Receivers (mV) (Slow FTSMode)
.....
Victim XNet  Victim Drvr  Victim Rcvr  HSOddXtalk  HSEvenXtalk  LSOddXtalk  LSEvenXtalk  Aggr XNet  Aggr Drvr
.....
NEWDPR N03039  NEWDPR U6 3  NEWDPR U5 1  55.33      42.16      48.23      69.47      NEWDPR N03723  NEWDPR U10 3
.....

```

Fig. 2.8.22 – Values of crosstalk voltages at length adjacent part of the conductors 320 milliinches

The values of crosstalk voltages for a length of the adjacent part of the conductors of 320 milliinches are given in Table 2.13.

Table 2.13 – Simulated crosstalk situations

Cross obstacle	Conditions for crosstalk		Value, mV
	Slave logic level	Transition master	
HSOddXtalk	HI	HI – LO	55.33
HSEvenXtalk	HI	LO – HI	42.16
LSOddXtalk	LO	LO – HI	48.23
LSEvenXtalk	LO	HI – LO	69.47

The maximum voltage value in this report is LSEvenXtalk (69.47 mV) when the *slave signal logic level* is zero (0 V) and *the master* switches from one to zero (from 5 V to 0 V).

***** Single Neighbor Crosstalk at Receivers (mV) (Slow FTSMode) *****									
Victim XNet	Victim Drvr	Victim Rcvr	HSOddXtalk	HSEvenXtalk	LSOddXtalk	LSEvenXtalk	Aggr XNet	Aggr Drvr	

Fig. 2.8.23 – Values of crosstalk voltages at length adjacent part of the conductors less than 300 milliinches

When reducing the length of the adjacent part of the conductors to a value less than 300 milliinches, it is possible to see the absence of cross-talk voltages. After the studies performed, it can be concluded that with the reduction of the length of the adjacent part of the conductors, the value of the cross-talk voltage decreases. And when setting the length not exceeding 300 milliinches, the corresponding cross-talk report field was empty, so it can be said that cross-talk does not occur or does not affect the functioning of the device. In Fig. 2.8.24 shows a graph of crosstalk voltages for a length of the adjacent part of the conductors of 4344 milliinches. This graph shows a pulse signal applied to the *master conductor* N3723 (U11 9) and a graph of the signal on the *slave conductor* N03039 (U5 1), the amplitude of the interference in this case reaches 1 V.

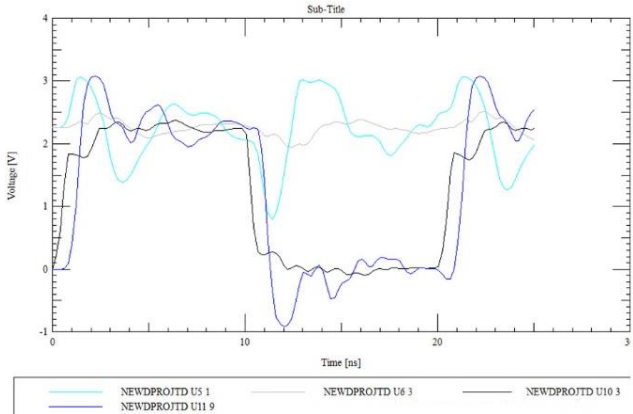


Fig. 2.8.24 – Crosstalk voltage graph at length adjacent part of the conductors 4344 milliinches

Fig. 2.8.25 shows a graph of crosstalk voltages when the length of the adjacent part of the conductors is reduced to 2020 milliinches. In this case, the crosstalk amplitude value has decreased to 750 mV.

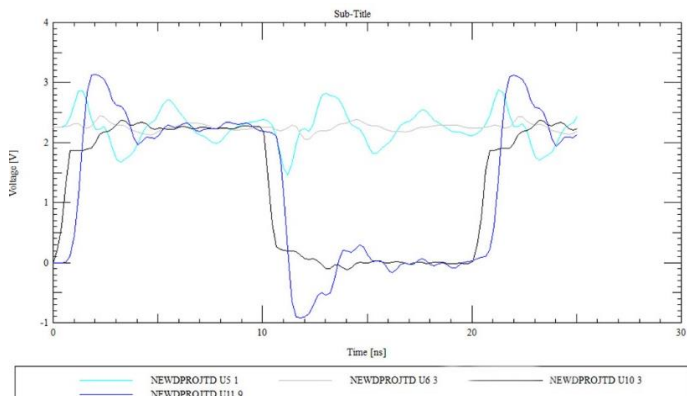


Fig. 2.8.25 – Crosstalk voltage graph at length adjacent part of the conductors 2020 milli-inches

Fig. 2.8.26 shows a graph of crosstalk voltages with a length of the adjacent part of the conductors of 1620 milli-inches. A pulsed pseudo-random signal (U11 9) is applied to the *master conductor* (N03723). In this case, we obtain a signal distortion on the *slave conductor* (U5 1). The crosstalk amplitude value has decreased to 550 mV.

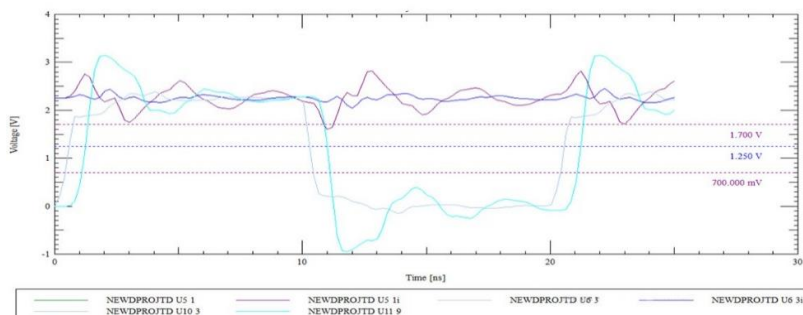


Fig. 2.8.26 – Crosstalk voltage graph at length adjacent part of the conductors 1620 milli-inches

Fig. 2.8.27 shows a graph of crosstalk voltages with a length of the adjacent part of the conductors of 360 milli-inches. A pulse signal (U11 9) is applied to the *master conductor* N03723. The signal on the *slave conductor*

N03039 (U5 1) has a distorted shape, the amplitude of the interference has decreased to 100 mV. The frequency of the pulse signal applied in all cases is $f = 50\text{ MHz}$, and the period $T = 20\text{ ns}$.

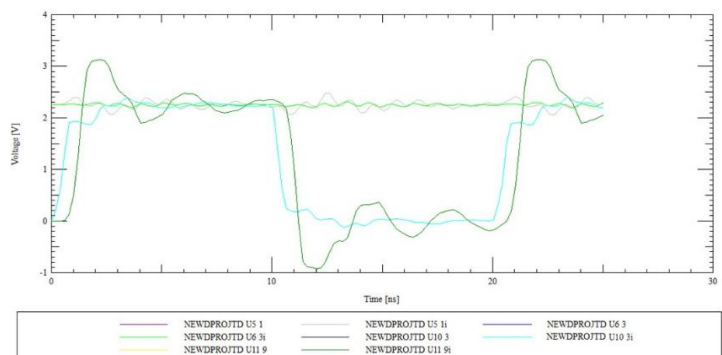


Fig. 2.8.28 – Crosstalk voltage graph at length adjacent part of the conductors 360 milliinches

Study of the device operation with different conductor thicknesses

The study is performed with a length of the adjacent part of the conductors of 1460 milliinches. The construction of crosstalk reports is performed similarly to the previous studies.

Fig. 2.8.29 shows the values of crosstalk voltages for conductors with a thickness of 1.2 milliinches (material – silver).

```

*****
Single Neighbor Crosstalk at Receivers (mV) (Slow FTSMode)
*****
Victim XNet      Victim Drvr      Victim Rcvr      HSOddXtalk      HSEvenXtalk      LSOddXtalk      LSEvenXtalk      Aggr XNet      Aggr Drvr
-----
NEWDPROJTD N03039 NEWDPROJTD U6 3 NEWDPROJTD U5 1 410.9      440.3      355.8      496.5      NEWDPROJTD N03723 NEWDPROJTD U10 3
-----
*****
Driver I/O Characteristics (Slow FTSMode) RiseSlew/FallSlew in (mV/ns)

```

Fig. 2.8.29 – Values of crosstalk voltages at thickness 1.2 milliinches conductors

According to Fig. 2.8.30, the maximum value of the crosstalk voltage is 496.5 mV, in LSEvenXtalk. The next step was to increase the conductor thickness to 5 milliinches in the Setup → Cross-Section window and generate a

cross-talk report. In this case, the maximum cross-talk voltage value is 707.9 mV at HSOddXtalk (Fig. 2.8.31).

```
=====
Single Neighbor Crosstalk at Receiver (mV) (Slow FTMode)
=====
HENDPROJTD H03039 HENDPROJTD U6 3 HENDPROJTD U5 1 707.9 539.3 620.1 576.6 HENDPROJTD H03723 HENDPROJTD U10 3
=====
Driver I/O Characteristics (Slow FTMode) RiseSlew/FallSlew in (mV/ns)
=====
Deve Device IOModel Volmax Vohmin RiseSlew FallSlew JTemp DiffPairMate
=====
HENDPROJTD U6 3 7432_DEP14_3_7432 CDSDefaultOutput_2p5v HA HA 3333 3333 HA HA
=====
```

Fig. 2.8.30 – Values of crosstalk voltages at thickness 5 milliinches conductors

Then the thickness was similarly changed to 6 milliinches and the crosstalk values were obtained (Fig. 2.8.31). The maximum voltage value in this case is 770.8mV, at HSOddXtalk.

```
=====
Single Neighbor Crosstalk at Receiver (mV) (Slow FTMode)
=====
HENDPROJTD H03039 HENDPROJTD U6 3 HENDPROJTD U5 1 770.8 544.3 679.6 580.1 HENDPROJTD H03723 HENDPROJTD U10 3
=====
Driver I/O Characteristics (Slow FTMode) RiseSlew/FallSlew in (mV/ns)
=====
Deve Device IOModel Volmax Vohmin RiseSlew FallSlew JTemp DiffPairMate
=====
HENDPROJTD U6 3 7432_DEP14_3_7432 CDSDefaultOutput_2p5v HA HA 3333 3333 HA HA
=====
```

Fig. 2.8.31 – Values of crosstalk voltages at thickness 6 milliinches conductors

The results of the research are given in Table 2.14, from which we can conclude that with a decrease in the thickness of the conductors under study, crosstalk decreases.

Table 2.14 – Values of crosstalk voltages for different conductor thicknesses

Thickness	HSOdd Xtalk, mV	HSEven Xtalk, mV	LSOdd Xtalk, mV	LSEven Xtalk, mV
1.2 milliinches	410.9	440.3	355.8	496.5
5 milliinches	707.9	539.3	620.1	576.6
6 milliinches	770.8	544.3	679.6	580.1

Study of the device operation with different conductor materials

Data on the materials used for the study and their electrical conductivity are given in Table 2.15. To set the conductor material, go to *Setup* → *Cross-Section* and set the parameters of the conductors and dielectrics in the window.

We will perform the study with a length of the adjacent part of the conductors of 550 milliinches, a thickness of the conductors of 5 milliinches, and a distance between the conductors of 1.2 milliinches. After appropriate settings, reports were obtained that contain data on the magnitude of the crosstalk and the maximum values of the obtained voltages (Table 2.15).

Table 2.15 – Values of crosstalk voltages for different conductor materials

Material	Electrical conductivity value, S/m	Maximum crosstalk voltage, mV	Crosstalk type
Silver	62500000	141.6	LSEvenXtalk
Platinum	10000000	141.3	LSEvenXtalk
Copper	59500000	143	LSEvenXtalk
Aluminum	35000000	142.7	LSEvenXtalk
Steel	14000000	128.4	LSEvenXtalk
Gold	43000000	142.8	LSEvenXtalk
Nickel	14000000	141.9	LSEvenXtalk

After analyzing the obtained data, we can conclude that changing the conductor material has an insignificant effect on changing the crosstalk voltage.

Study of the device operation at different distances between conductors

Let's study the crosstalk voltage values at conductor spacings of 5 and 6 milliinches. First, we set the minimum distance between conductors during tracing (5 milliinches), then we create a crosstalk voltage report for *slave conductor* N03039 and obtain the crosstalk voltage values (Fig. 2.8.32).

```

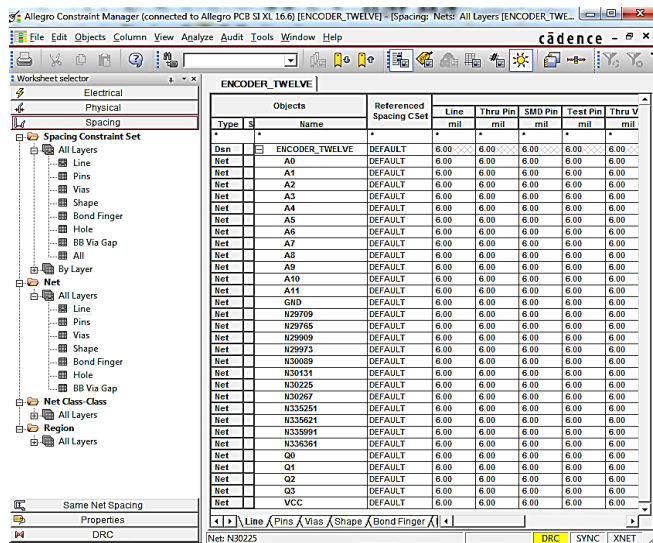
*****
Single Neighbor Crosstalk at Receivers (mV) (Slow FTSMode)
*****
Victim XNet  Victim Drvr  Victim Rcvr  HSOddXtalk  HSEvenXtalk  LSOddXtalk  LSEvenXtalk  Aggr XNet  Aggr Drvr
-----
NEWMAG N03347  NEWMAG U2 4  NEWMAG U9 9  587.3      436.5      481        517.2      NEWMAG N03893  NEWMAG U14 11
-----

```

Fig. 2.8.32 – Crosstalk voltage values with a conductor spacing of 5 milliinches

We increase the distance between the conductors to 6 milliinches (Fig. 2.8.33) and perform automatic tracing (Fig. 2.8.34). We create a report of

crosstalk voltages (Fig. 2.8.35), from which it is clear that at a distance between the conductors of 6 milliinches, the value of crosstalk voltages is absent.



Type	Name	Referenced Spacing C Set	Line mil	Thru Pin mil	SMD Pin mil	Test Pin mil	Thru V mil
Den	ENCODER_TWELVE	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	A0	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	A1	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	A2	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	A3	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	A4	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	A5	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	A6	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	A7	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	A8	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	A9	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	A10	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	A11	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	GND	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	N29799	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	N29765	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	N29909	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	N29973	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	N30409	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	N30131	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	N30225	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	N30267	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	N335251	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	N335621	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	N335991	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	N336361	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	Q0	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	Q1	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	Q2	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	Q3	DEFAULT	6.00	6.00	6.00	6.00	6.00
Net	VCC	DEFAULT	6.00	6.00	6.00	6.00	6.00

Fig. 2.8.33 – Increasing the distance between conductors to 6 milliinches

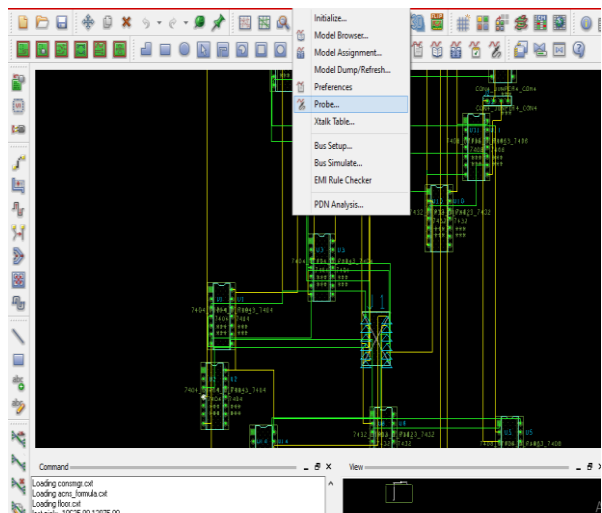


Fig. 2.8.34 – Trace result

Single Neighbor Crosstalk at Receivers (mV) (Slow FTSMode)									

Victim XNet	Victim Drvr	Victim Rcvr	HSOddXtalk	HSEvenXtalk	LSOddXtalk	LSEvenXtalk	Aggr XNet	Aggr Drvr	

Fig. 2.8.35 – Crosstalk voltage values with a conductor spacing of 6 milliinches

Study of the device operation with different numbers of layers between conductors

We conduct research to detect crosstalk between conductors when they are located in different layers of the printed circuit board – from being located in one common layer to being placed in layers that are on opposite levels.

To perform this study, select *Setup* → *Cross-Section* → *Add Layer*, we set 5 layers of conductors and 4 layers of dielectrics placed between them (Fig. 2.8.36).

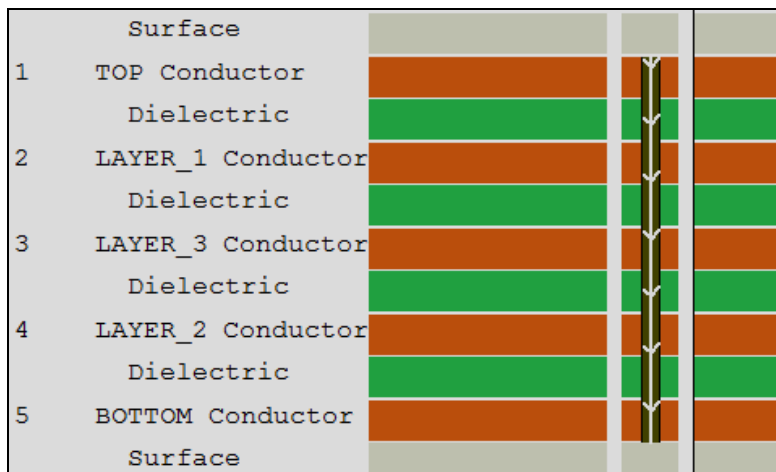


Fig. 2.8.36 – Installation of 5 conductive and 4 dielectric layers

Research on the detection of the magnitude of crosstalk is carried out using the established parameters of the thickness of the conductor layers of 1.2 milliinches and 5 milliinches, the thickness of the dielectric layers of 8 milliinches and 16 milliinches. The length of the adjacent part of the conductors is 450 milliinches. The values of the crosstalk voltages obtained during the simulation are given in Tables 2.16 – 2.17. After analyzing the

obtained data, we can conclude that with an increase in the thickness of the dielectric layer, the value of the crosstalk voltage decreases, but not as much as when conductors are placed on layers of different levels.

Table 2.16 – Crosstalk voltage values for conductor layers of 1.2 milliinches when placed in different layers of the board

Dielectric thickness, milliinches	Conductors placed in 1 layer	Conductors placed in 1 and 2 layers	Conductors placed in 1 and 3 layers	Conductors placed in 1 and 4 layers	Conductors placed in 1 and 5 layers
8	129.3 mV	116.5 mV	94.28 mV	76.06 mV	69.89 mV
16	145.9 mV	113.2 mV	83.79 mV	68.72 mV	62.89 mV

Table 2.17 – Crosstalk voltage values for conductor layers of 5 milliinches when placed in different layers of the board

Dielectric thickness, milliinches	Conductors placed in 1 layer	Conductors placed in 1 and 2 layers	Conductors placed in 1 and 3 layers	Conductors placed in 1 and 4 layers	Conductors placed in 1 and 5 layers
8	170.4 mV	138 mV	103 mV	77.01 mV	73.26 mV
16	180.3 mV	123.8 mV	88 mV	65.74 mV	59.32 mV

It has also been found that between conductors with higher thickness values, crosstalk of greater magnitude is formed when they are in the same layer. However, when the conductors are placed in different layers, the interference is reduced.

The crosstalk voltage value is lower for conductors with a greater thickness (5 milliinches) if they are located in layers 1 and 5, 1 and 4 with a dielectric layer thickness of 16 milliinches, when the total distance between the layers at 1 and 4 is 58 milliinches.

In Fig. 2.8.37 shows the simulation results: a pulse signal (shown in black) is fed to the input of the *master conductor*; the signal of the *slave conductor* should have a constant logic level of unity in the absence of interference in this simulation in the case of being in one layer (shown in blue),

as well as in the case of 1 and 5 layers with 16 milliinches of dielectric between the layers (shown in gray). Thus, we obtain a decrease in the amplitude of the signal distortion on the *slave conductor*.

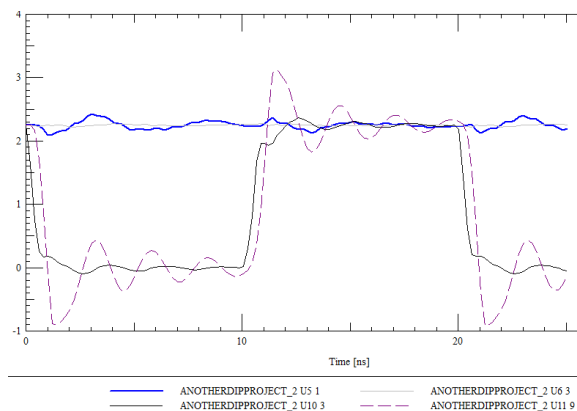


Fig. 2.8.37 – Reducing the amplitude of signal distortion when placing conductors in different layers and increasing the distance between them

Study of the device operation with different numbers of adjacent conductors

Let us conduct a study that determines the impact of increasing the number of incorrectly placed conductors in one layer. For this, modeling was performed under the condition of possible interaction between 2, 5 and 10 conductors (Fig. 2.8.38), namely, conditions were created for the occurrence of crosstalk between conductors for further analysis of their magnitude.

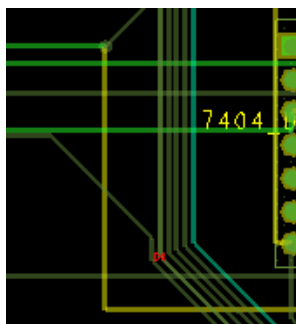


Fig. 2.8.38 – Increasing the number of adjacent conductors

Under the condition of interaction between two conductors located in the first layer, reports of crosstalk voltage values were generated (Fig. 2.8.39), which are listed in Table 2.18 (the maximum value is 730.3 mV in HSOddXtalk). After changing the number of conductors, data on the magnitude of crosstalk were obtained for 5 and 10 conductors, a report on the presence and magnitude of crosstalk for 5 adjacent conductors is given in Fig. 2.8.40, and for 10 conductors – in Fig. 2.8.41.

```

.....
ming Windows Neighbor Group Crosstalk at Receivers (mV) (Slow FTSMode)
.....
ctim XNet      Victim Drvr      Victim Rcvr      HSOddXtalk      HSEvenXtalk      LSOddXtalk      LSEvenXtalk      Group
-----
OTHERDIPPROJECT_2 N03039 ANOTHERDIPPROJECT_2 U6 3 ANOTHERDIPPROJECT_2 U5 1 730.3      585      685.9      577.2      Group all_tw
.....

```

Fig. 2.8.39 – Crosstalk voltage values with 2 adjacent conductors

```

.....
ming Windows Neighbor Group Crosstalk at Receivers (mV) (Slow FTSMode)
.....
ctim XNet      Victim Drvr      Victim Rcvr      HSOddXtalk      HSEvenXtalk      LSOddXtalk      LSEvenXtalk      Group
-----
OTHERDIPPROJECT_2 N03039 ANOTHERDIPPROJECT_2 U6 3 ANOTHERDIPPROJECT_2 U5 1 1129      632.9      1022      697.9      Group all_tw
.....

```

Fig. 2.8.40 – Crosstalk voltage values with 5 adjacent conductors

```

.....
ming Windows Neighbor Group Crosstalk at Receivers (mV) (Slow FTSMode)
.....
ctim XNet      Victim Drvr      Victim Rcvr      HSOddXtalk      HSEvenXtalk      LSOddXtalk      LSEvenXtalk      Group
-----
OTHERDIPPROJECT_2 N03039 ANOTHERDIPPROJECT_2 U6 3 ANOTHERDIPPROJECT_2 U5 1 1590      630.3      1453      584      Group all_
.....

```

Fig. 2.8.41 – Crosstalk voltage values with 10 adjacent conductors

Table 2.18 – Values of crosstalk voltages depending on the number of conductors

Number	HSOddXtalk	HSEvenXtalk	LSOddXtalk	LSEvenXtalk
2	730.3	585	685.9	577.2
5	1129	632	1022	697.9
10	1590	630.3	1453	584

Graphs that demonstrate the dependence of the change in the crosstalk voltage value on the change in the number of conductors are shown in Fig. 2.8.42 (2 conductors) and Fig. 2.8.43 (10 conductors).

These graphs show pulse signals and the change in the amplitude of the interference voltage on the *slave conductors*.

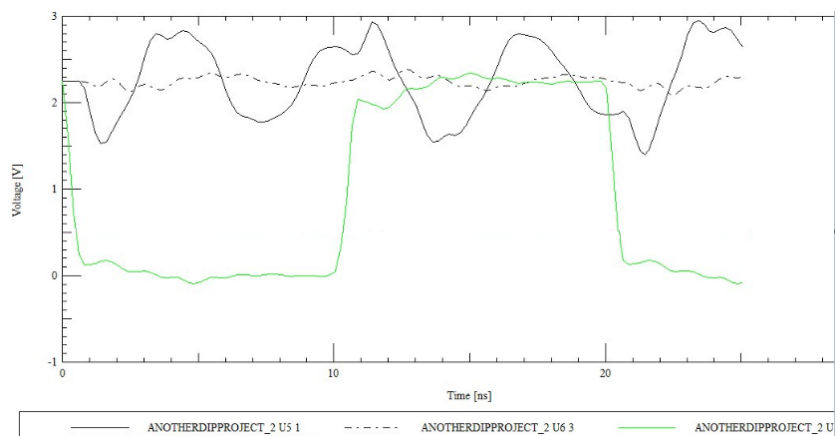


Fig. 2.8.42 – Signal change when using 2 conductors

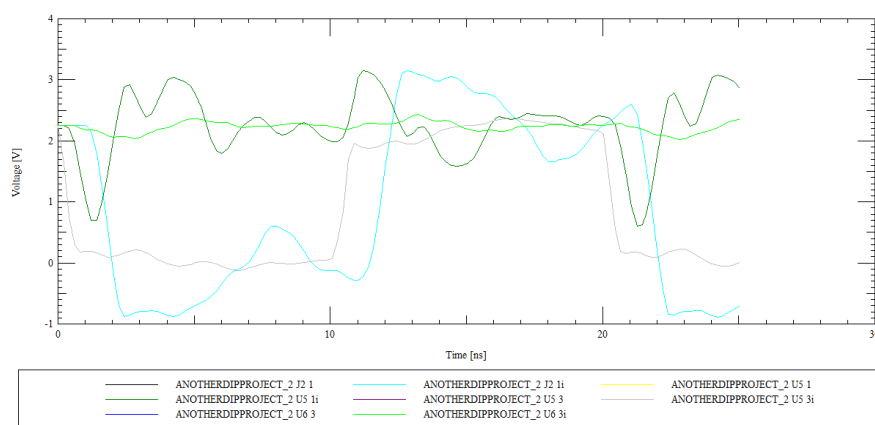


Fig. 2.8.43 – Signal change when using 10 conductors

Tasks for practical work

Conduct a study of the device developed in practical work № 1 to evaluate its operation taking into account electromagnetic compatibility.

CHAPTER 3

USING ORCAD FOR MODELING COMPLEX DEVICES

Laboratory work 1

Developing a functional diagram of the device.
Break down the diagram into hierarchical levels.
Modeling elements of the first hierarchical level

The purpose of the work: to acquire and consolidate practical skills in modeling logic elements in OrCAD.

Theoretical information

The functional diagram of a digital device can be broken down into hierarchical levels, the simplest of which is the level of logic elements, which serve as the basis for designing more complex devices.

That is why studying the parameters of the element base is important for the further design and correct functioning of the developed device.

OrCAD has several series of logic elements, which differ not only in the set of logic elements, but also in the internal parameters of this logic.

Since the passage of the input signal through the element requires a certain amount of time, the output signal level is set through a time interval, which is one of the most important parameters of logic and is called the element delay.

As a rule, we are talking about a few nanoseconds, which, nevertheless, when creating complex devices with a large number of logic elements, can accumulate and lead to malfunctions in the developed device. The delays of elements of different series can vary significantly, so it is not recommended to combine elements of different series.

Tasks for laboratory work: develop a functional diagram of the device according to Table 3.1.

Perform modeling of the lower hierarchical level elements, obtain timing diagrams, and evaluate the delay of each element.

Table 3.1

№	Individual task	Microcircuit series
1	Scheme for adding and subtracting 6-digit binary numbers	7400
2	High-speed parallel 4-bit multiplier	74HC
3	Matrix multiplier for 4-bit binary numbers	74L
4	Scheme for converting 8-bit binary numbers into binary-decimal code	7400
5	Scheme for adding and subtracting 8-bit binary numbers	74AC
6	Matrix multiplier for binary 8-bit numbers	74AS
7	Addition-subtraction circuit for 8-bit binary numbers	74S
8	Comparison circuit for 12-bit codes	74F
9	Schematic diagram implementing 15 logical functions from four arguments	74S
10	Scheme for converting binary-decimal numbers to the binary number system	7400
11	Schematic diagram of a multiplier for 3-digit binary numbers	74HC
12	Schematic diagram for implementing logical addition, subtraction, XOR and inversion operations	74AS
13	Schematic diagram of binary number division	74AS
14	Scheme for raising a number to a power	7400
15	Scheme for converting a complementary code into a forward and reverse code based on an arithmetic-logic device	74HC

Example of task execution

Let's develop a device for multiplying two 8-bit numbers with analysis of the lower bits of the second multiplier and shifting the first multiplier towards the higher bits. Series – 7400.

To implement the multiplication algorithm (Fig. 3.1.1), the following are required:

- 1) 16-bit register for the partial sum;
- 2) 8-bit registers for multipliers;
- 3) 16-bit adder;

- 4) a pulse counter to determine the end of multiplication;
- 5) a flip-flop for storing the most significant digit of the multiplier.

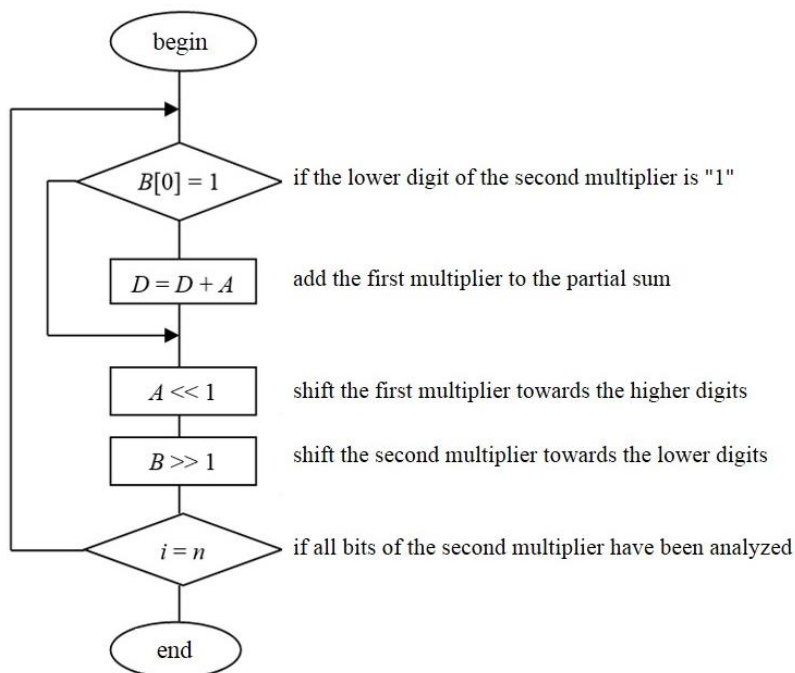


Fig. 3.1.1 – General multiplication algorithm

The functional diagram of the device under development is shown in Fig. 3.1.2. Let's divide the diagram into five hierarchical levels:

- *elements of the 1st level of the hierarchy:*
NOT, 2AND, 2AND-NOT, 3AND, 3AND-NOT, 2OR-NOT, 3OR-NOT, 2XOR.
- *elements of the 2nd level of the hierarchy:*
RS, D, JK, T flip-flop; adder; multiplexer; control device.
- *Elements of the third level of the hierarchy:*
4-bit shift register; 4-bit adder; 4-bit counter; 16-bit parallel register.
- *Elements of the 4th level of the hierarchy:*
16-bit shift register; 16-bit adder; 8-bit register.
- *Elements of the 5th level of the hierarchy:*

The element of the 5th level of the hierarchy is a device for multiplying two 8-bit numbers.

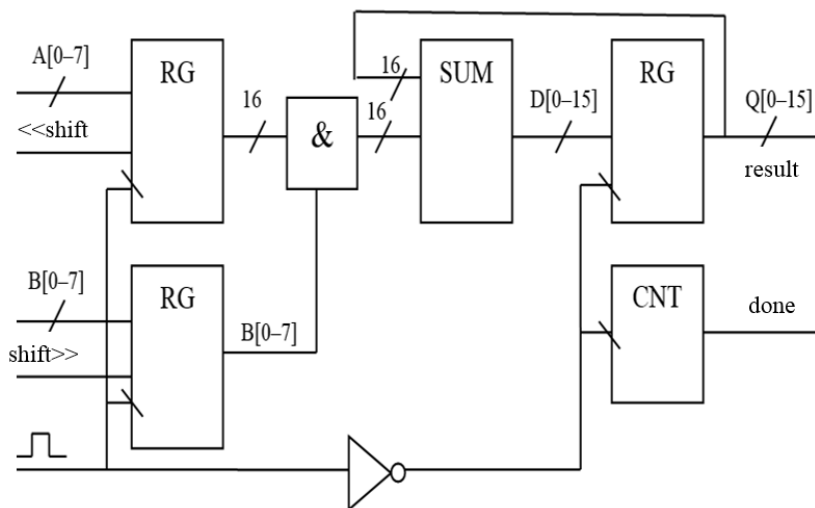


Fig. 3.1.2 – Functional diagram of the device

Now let's create a new project. To do this, launch OrCAD Capture, then select *File – New – Project* in the main menu. Enter the name and directory for the project being created. Select the element base according to the individual task (Table 3.1).

Then, using the min/max method of accounting for delays in logic elements, we will perform the simulation:

- feed possible combinations of digital signals to the inputs of the elements;
- obtain timing diagrams of input and output signals, and evaluate the delay of each element.

To perform these actions, first place the logic elements on the workspace. To do this, select *Place – Part* in the main menu, then add a library.

In our case, this is library 7400 and *SOURCE* – in the window, click the *Add Library* button (Fig. 3.1.3) and select the necessary libraries in the form of files with the extension *.olb, which can be found in the *CADENCE\SPB_16.5\tools\capture\library\pspice* directory.

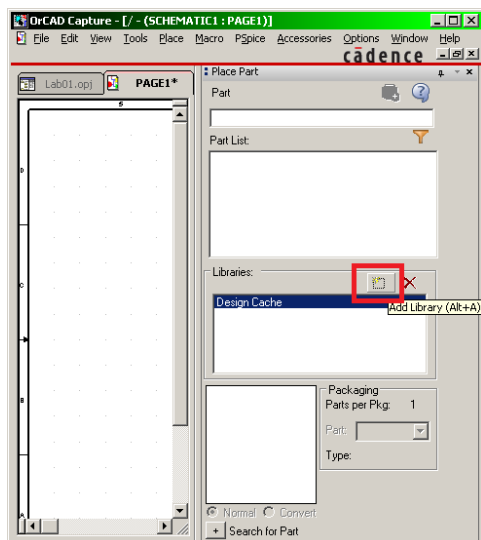


Fig. 3.1.3 – Adding a library

In Fig. 3.1.4 shows a research diagram consisting of logic elements and a signal generator.

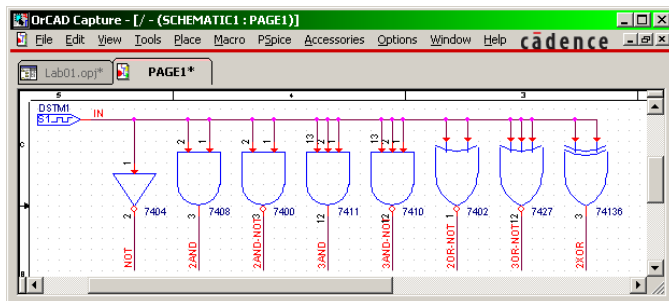


Fig. 3.1.4 – Research diagram

Now everything is ready for the main operation – simulation. To do this, you need to create a simulation profile: select *PSpice – New Simulation Profile* in the menu. In the window, enter the profile name and click *Create*.

A window for configuring the simulation profile appears, in which we enter a simulation time of 2 μ s (Fig. 3.1.5).

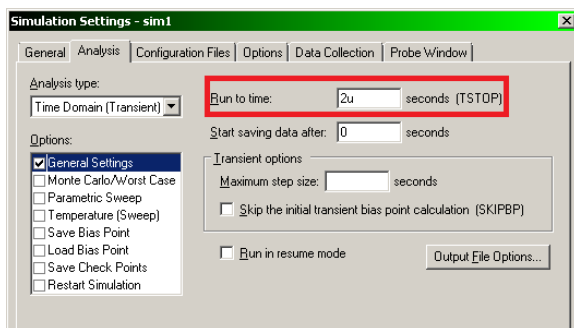


Fig. 3.1.5 – Setting the simulation time

In the *Options* tab of the *Gate-level Simulation* category, set the *Worst-case* (min/max) parameter as shown in Fig. 3.1.6.

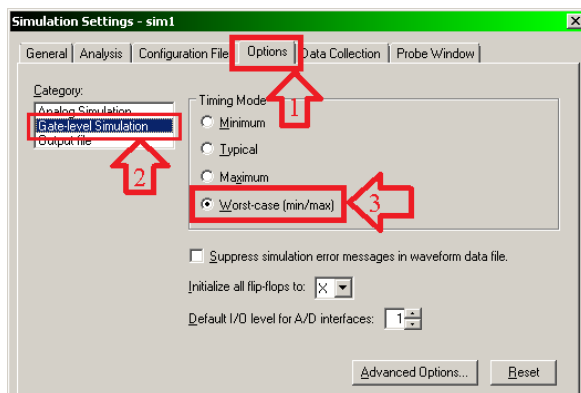


Fig. 3.1.6 – Connecting the min/max mode

Now, in the main menu, select *PSpice – Markers – Voltage Level* and set the "voltmeter" on the conductor (Fig. 3.1.7).

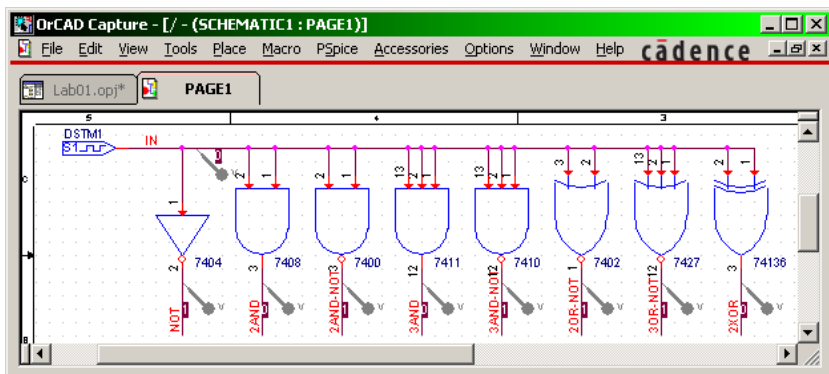


Fig. 3.1.7 – Voltmeter placement

To simulate the operation of the device, select *PSpice – Run* in the main menu. As a result, we obtain time diagrams (Fig. 3.1.8).

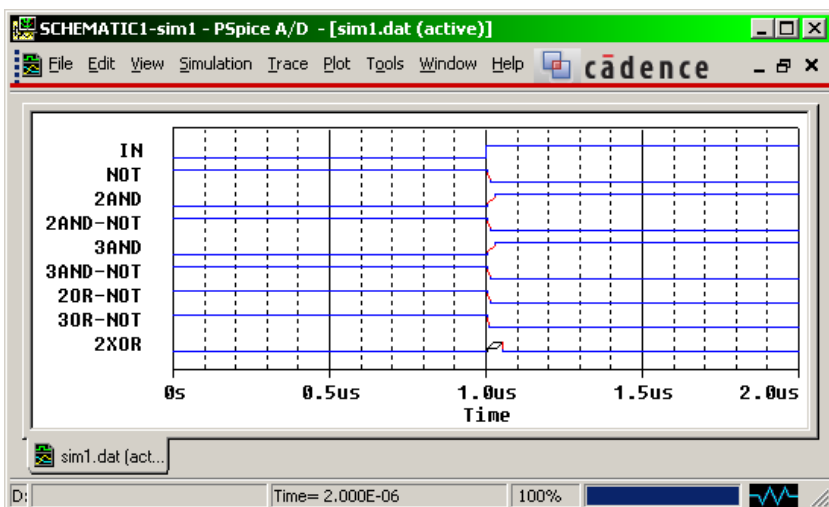


Fig. 3.1.8 – Time diagrams

To determine the delay and uncertainty period, we increase the scale of the time diagram (Fig. 3.1.9) and enter the data in Table 3.2.

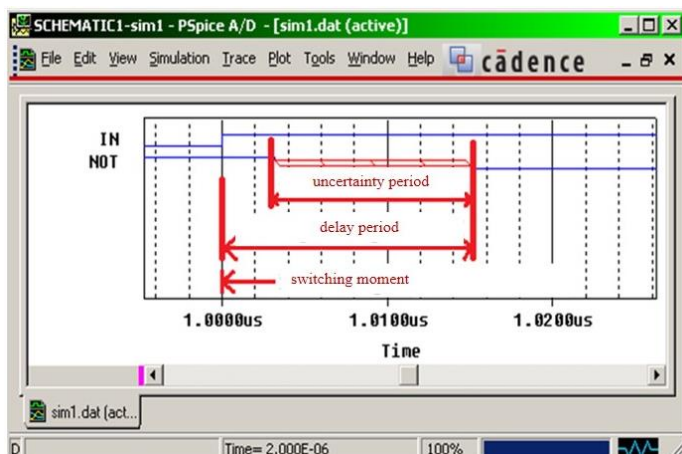


Fig. 3.1.9 – NOT element timing diagrams

Table 3.2

Element (library 7400)	Delay period T_s , ns	Indeterminacy period $T_{\text{uncertainty}}$, ns
NOT	15	11.8
2AND	27	20
2AND–NOT	15	12.2
3AND	27	20.2
3AND–NOT	15	12.2
2OR–NOT	15	11.8
3OR–NOT	11	8.2
2XOR	55	50.2

Laboratory work 2

Modeling elements of the second hierarchical level

The purpose of the work: develop a functional diagram of the device. Acquire and consolidate practical skills in designing and modeling second-level hierarchical elements in OrCAD.

Tasks for laboratory work: design and model second-level hierarchical elements.

Example of task completion

The elements of the second hierarchical level for the given device are: RS and D flip-flop; adders; multiplexers. These circuits are implemented on elements of the first hierarchical level.

Modeling of D flip-flop

In Fig. 3.2.1 shows the D flip-flop circuit.

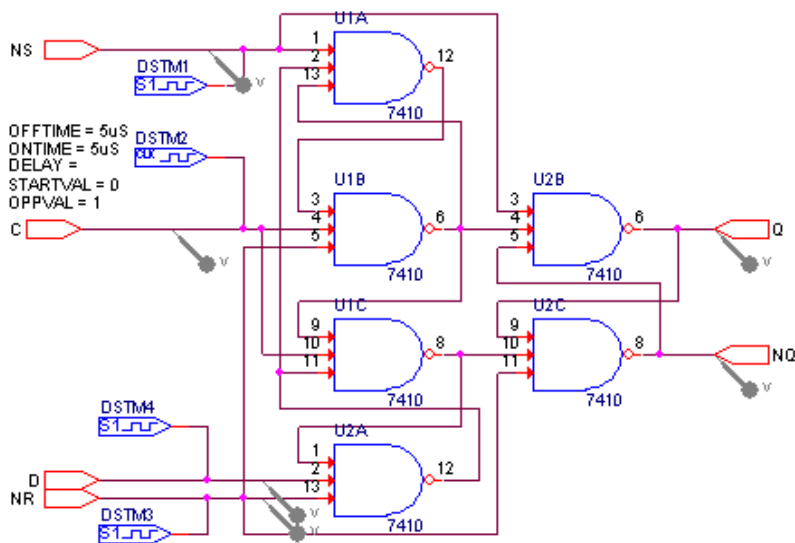


Fig. 3.2.1 – D flip-flop

We perform the simulation of the *D* flip-flop (Fig. 3.2.2). We determine the uncertainty time (42 ns) and the delay time (52 ns) for the *NQ* output when transitioning from "1" to "0" (Fig. 3.2.3).

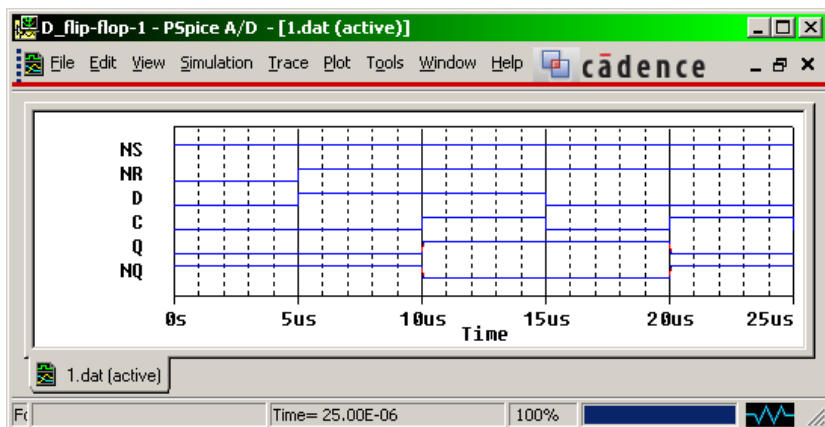


Fig. 3.2.2 – Timing diagrams of the *D* flip-flop

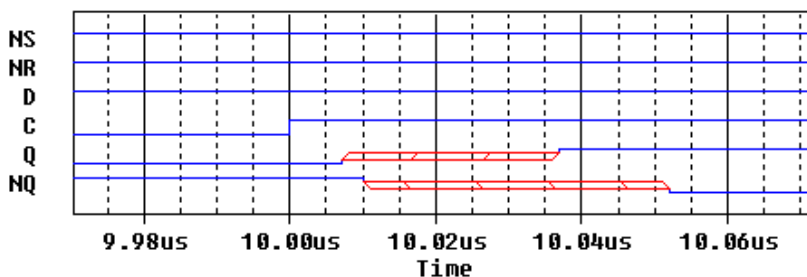


Fig. 3.2.3 – Timing diagrams

Modeling of RS flip-flop

Since the devices modeled in this work are finite and will be used in subsequent laboratory work, we will create a separate diagram for each device, and we will do all this in one project. To do this, in the file browser of this project (Fig. 3.2.4), right-click on the project name with the extension *.dsn and select *New Schematic*.

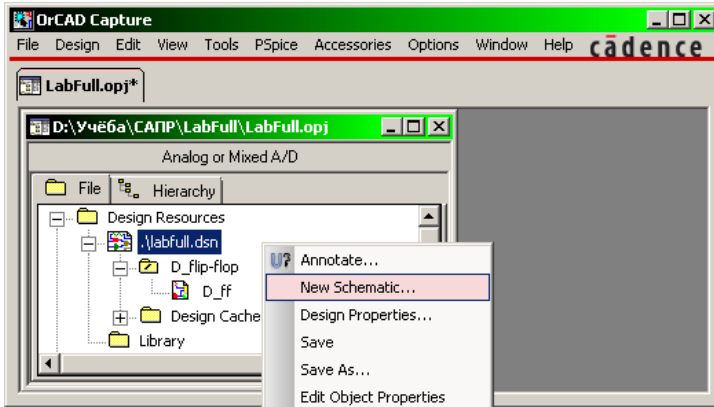


Fig. 3.2.4 – Creating a new schematic in the project

Now we create a sheet on which we will assemble the device schematic (RS flip-flop) – this is the *New Page* item.

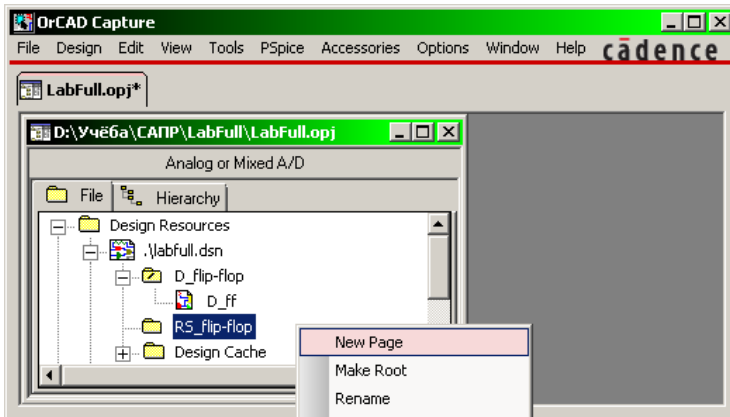


Fig 3.2.5 – Creating a new page

In Fig. 3.2.6 shows a two-stage synchronous RS flip-flop with an additional reset input. To simulate this circuit, it must be made the root of the project file hierarchy. To do this, open the RS flip-flop circuit menu and select *Make Root* (Fig. 3.2.7).

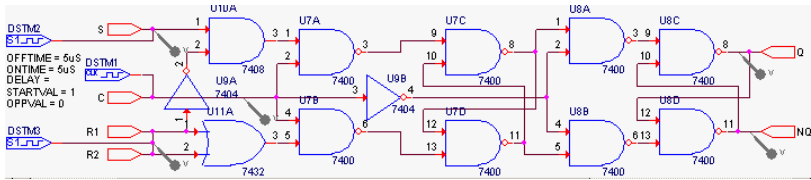


Fig. 3.2.6 – RS flip-flop circuit

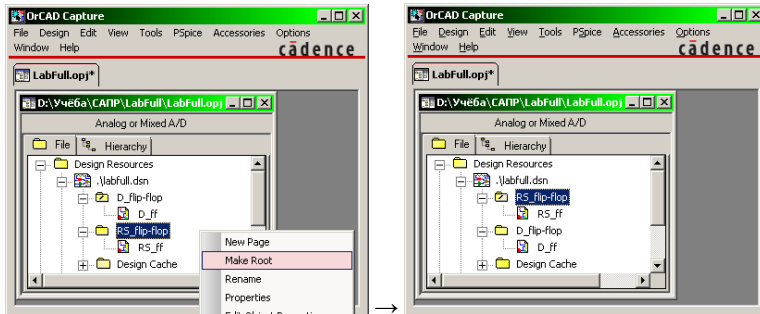


Fig. 3.2.7 – Assigning the root circuit

A simulation profile must be created for each circuit. We determine the uncertainty time (59.2 ns) and the delay time (74 ns) for the NQ output when transitioning from "0" to "1" (Fig. 3.2.8).

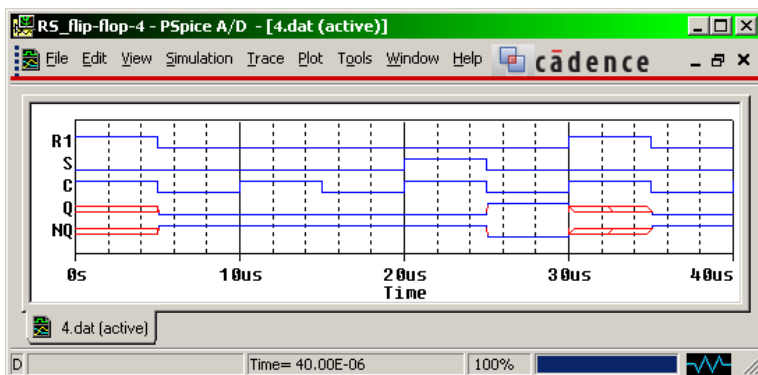


Fig. 3.2.8 – Timing diagram of the RS flip-flop

Simulation of *JK* flip-flop

In Fig. 3.2.9 shows the diagram of a *JK* flip-flop. Synchronization is performed on the falling edge of the pulse (Fig. 3.2.10).

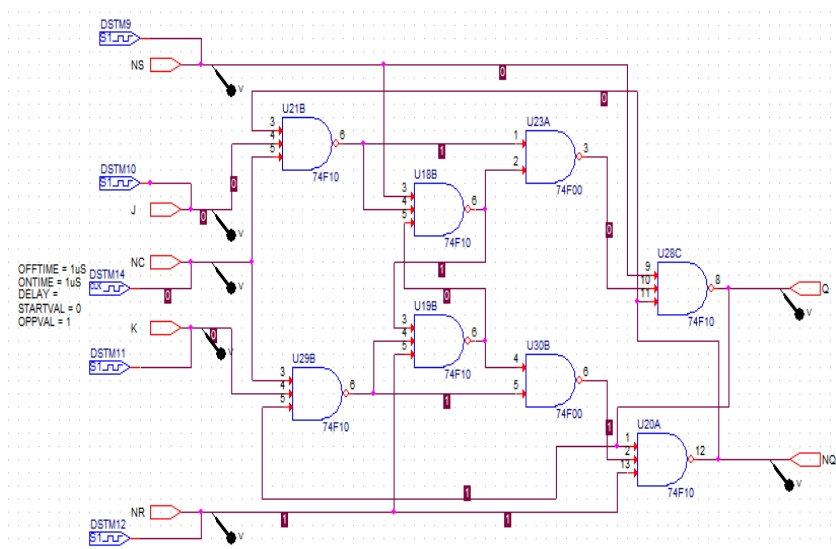


Fig. 3.2.9 – *JK* flip-flop

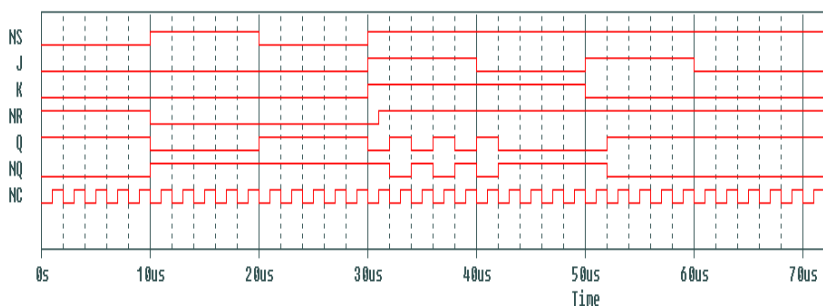


Fig. 3.2.10 – *JK* flip-flop timing diagram

Modeling the *JK* flip-flop in min/max mode showed unacceptable results, so the standard *Typical* mode was chosen. We determine the delay time (25 ns) for the *NQ* output during the transition from "0" to "1" (Fig. 3.2.11).

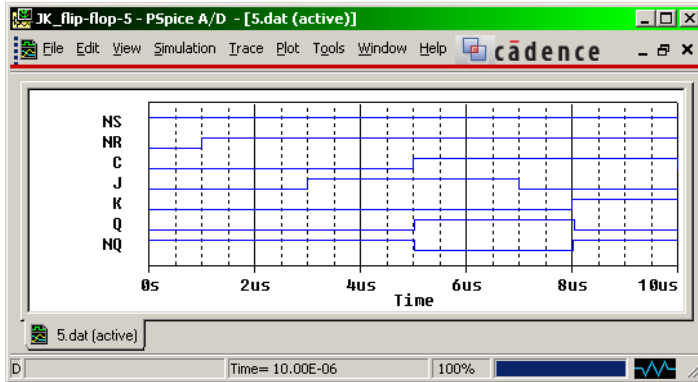


Fig. 3.2.11 – Timing diagram of the *JK* flip-flop

Simulation of the adder

In Fig. 3.2.12 shows a diagram of one of the options for constructing a complete single-bit adder.

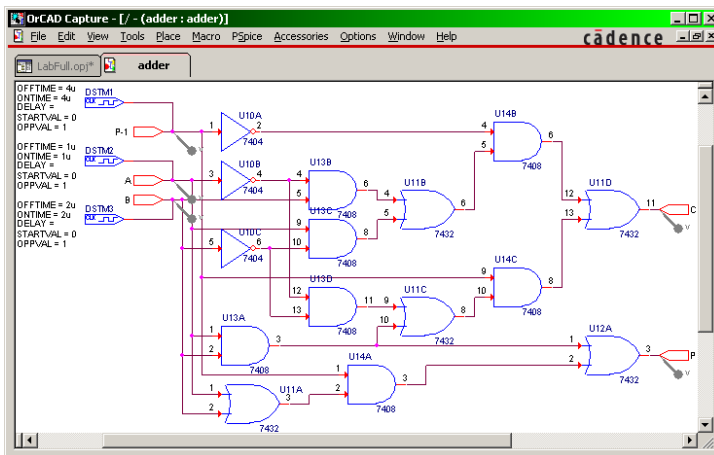


Fig. 3.2.12 – Adder

We determine the uncertainty time (95 ns) and the delay time (106 ns) for the *P* output during the transition from "0" to "1" (Fig. 3.2.13).

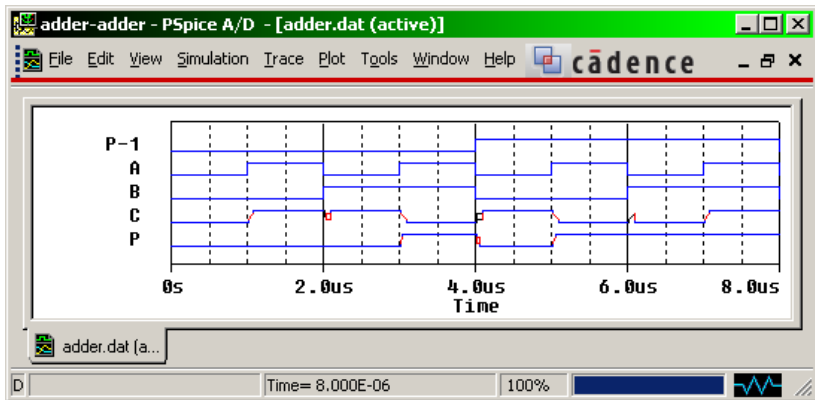


Fig. 3.2.13 – Timing diagram of the adder

Multiplexer simulation

In Fig. 3.2.14 shows a diagram of a 4×1 multiplexer. We determine the uncertainty time (42.2 ns) and delay time (57 ns) for output Y when transitioning from "0" to "1" (Fig. 3.2.15).

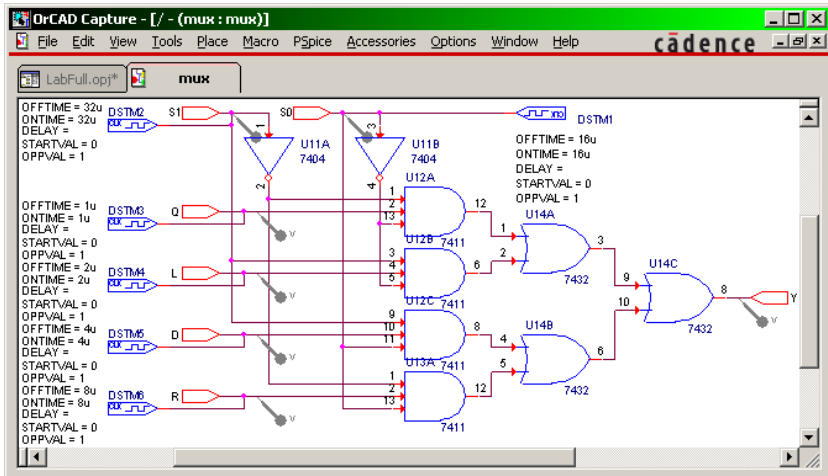


Fig. 3.2.14 – Multiplexer

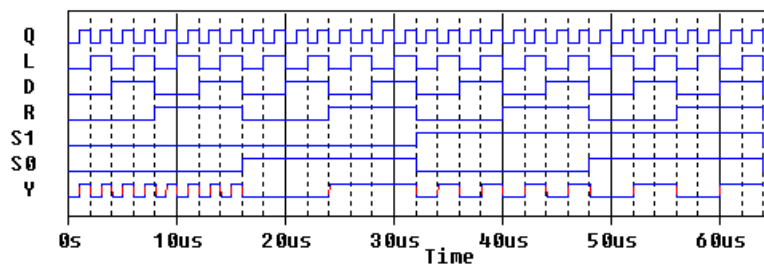


Fig. 3.2.15 – Multiplexer timing diagram

Modeling of a control device

In Fig. 3.2.16 shows a diagram of a control device that allows input data to pass through unchanged if "1" is applied to the control input, otherwise "0". We determine the period of uncertainty and delay (Fig. 3.2.17) and enter the data in Table 3.3.

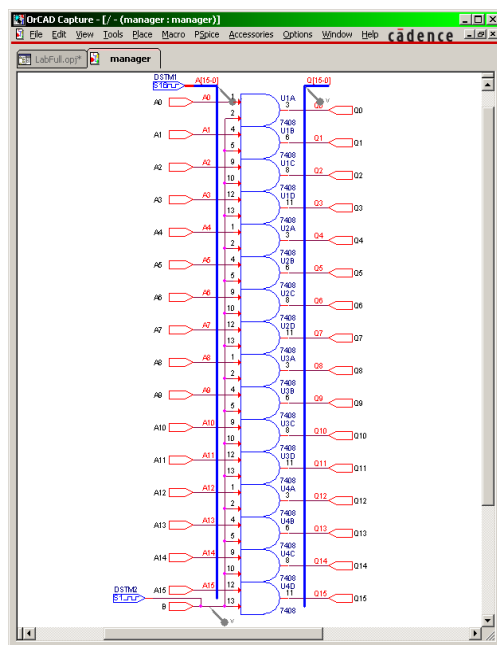


Fig. 3.2.16 – Control device

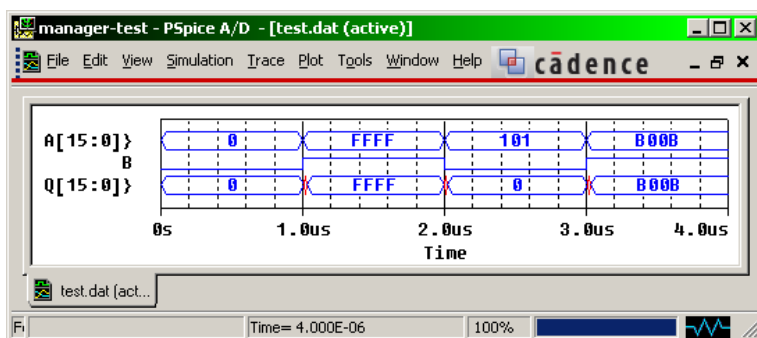


Fig. 3.2.17 – Timing diagram of the control device

Table 3.3

Device	Delay period T_s , ns	Indeterminacy period $T_{\text{uncertainty}}$, ns
<i>D</i> flip-flop	52	42
<i>RS</i> flip-flop	74	59,2
<i>JK</i> flip-flop	25	–
adder	106	95
multiplexer	57	42,2
control device	27	7

Laboratory work 3

Modeling elements of the third hierarchical level

The purpose of the work: develop a functional diagram of the device. Acquire and consolidate practical skills in designing and modeling third-level hierarchical elements in OrCAD.

Tasks for laboratory work: design and model third-level hierarchical elements.

Example of task completion

In our case, the elements of the third hierarchical level are a four-bit register, an adder, a counter, and a 16-bit register. In this and subsequent laboratory works, we will use the elements of the second hierarchical level that we created earlier.

Modeling a 4-bit register

Create a new schematic and sheet (Fig. 3.3.1). Place the hierarchical multiplexer block on the sheet. To do this, select *Place – Hierarchical Block* in the main menu. In the window, enter any name in the *Reference* field and the specific name of the schematic (*mux*) in the *Implementation name* field, as shown in Fig. 3.3.1.

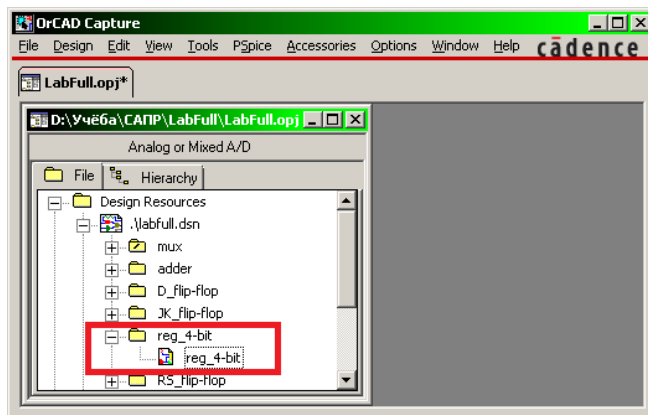


Fig. 3.3.1 – Creating a new schematic

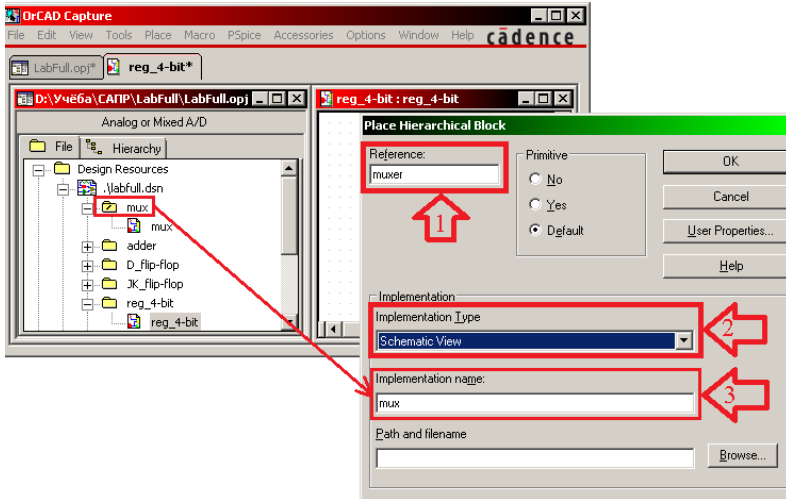


Fig. 3.3.2 – Process of creating a hierarchical block

After clicking **OK**, draw a multiplexer on the sheet in the form of a rectangle of any size. You can freely change the size of the blocks and the location of the ports (Fig. 3.3.3).

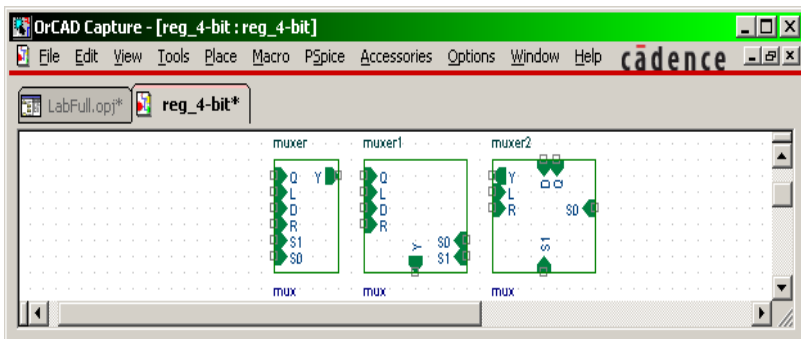


Fig. 3.3.3 – Hierarchical blocks on the worksheet

In Fig. 3.3.4 shows a 4-bit shift register with reverse synchronization.

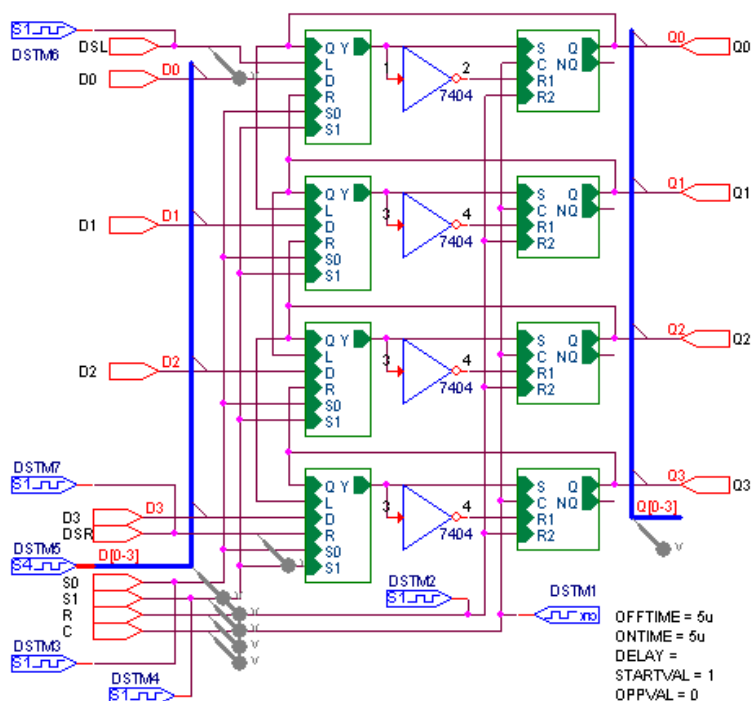


Fig. 3.3.4 – Register diagram

The element named *DSTM5* of type *STIM4* from the *SOURCE* library is analogous to *STIM1*, but is capable of "outputting" a 4-bit number (Fig. 3.3.5).

COMMAND1	COMMAND2	COMMAND3
0s 0000	15u 0011	40u 0010

Fig. 3.3.5 – Input signals in *STIM4*

For the 4-bit number output by *STIM4*, a bus is used, which is shown in the diagram as a wide blue conductor.

To connect single-bit conductors and buses, "connectors" are used, which can be selected in the main menu *Place – Bus Entry*.

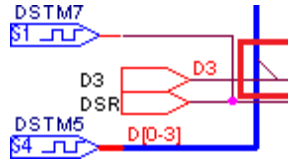


Fig. 3.3.6 – View of the bus connector

The names of the conductors and buses play an important role in this case. As we can see from Fig. 3.3.4, the names of the conductors are: $D0$, $D1$, $D2$, $D3$, and the name of the bus to which they are connected is $D[0-3]$.

The rule for constructing the name of a single-bit conductor is as follows:

name <bus_bit_index>;

and for the bus:

name [<high_order_index> – <small_order_index>].

To avoid errors, block ports cannot be left without conductors. In our case, the NQ port in the RS flip-flop must be connected to a conductor, otherwise there will be an error and it will be impossible to simulate the device. *Typical* mode was selected for simulation, since min/max mode causes failures. We will only use *Typical* mode in the future. We determine the delay time (37 ns) for output $Q[0-3]$ during the transition from "3" to "2" (Fig. 3.3.7). The switching moment is shown more accurately in Fig. 3.3.8.

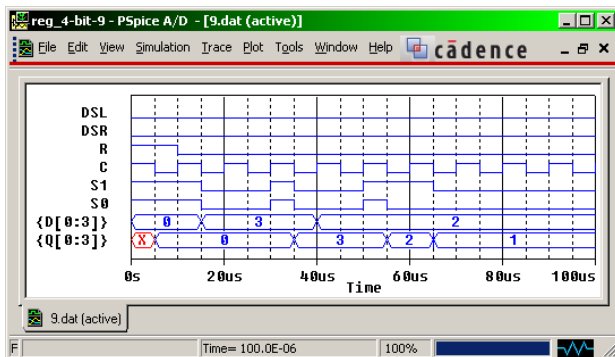


Fig. 3.3.7 – Timing diagram of the register operation

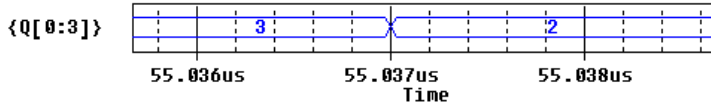


Fig. 3.3.8 – Switching moment at the register outputs

Simulation of a 4-bit adder

In Fig. 3.3.9 shows a 4-bit adder with serial carry.

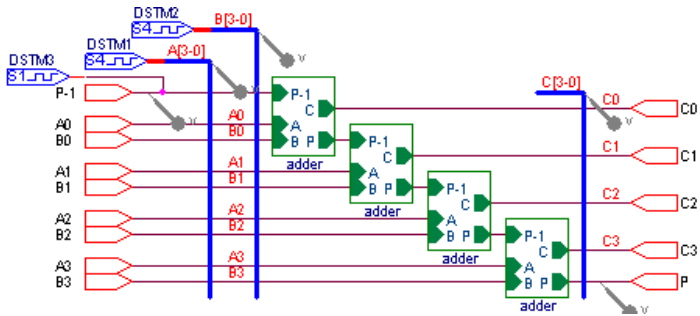


Fig. 3.3.9 – Adder circuit

Bus Entry does not need to be used to connect the bus and conductors if the bus and conductor intersect each other or if the bus and conductor comply with the naming rules. We determine the delay time (116.5 ns) of output $C[3:0]$ (Fig. 3.3.10).

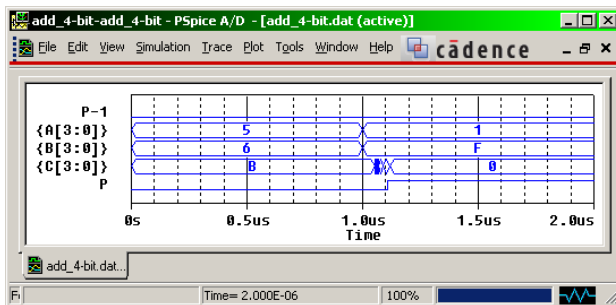


Fig. 3.3.10 – Timing diagram

Simulation of a 4-bit counter

In Fig. 3.3.11 shows a schema of a 4-bit counter with direct synchronization based on *D* flip-flop.

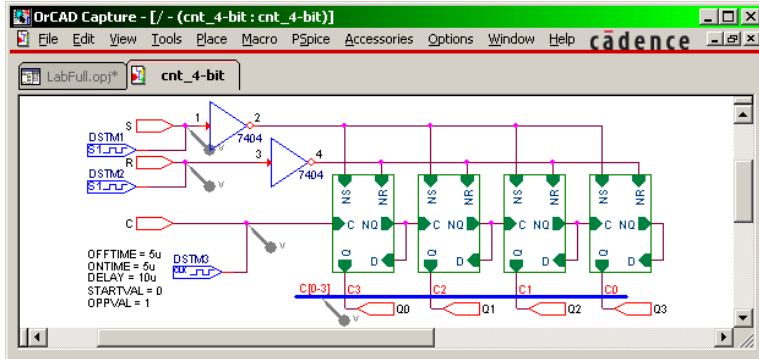


Fig. 3.3.11 – Counter circuit

We determine the delay time (79 ns) for output *C* [0-3] (Q0:Q3) (Fig. 3.3.12).

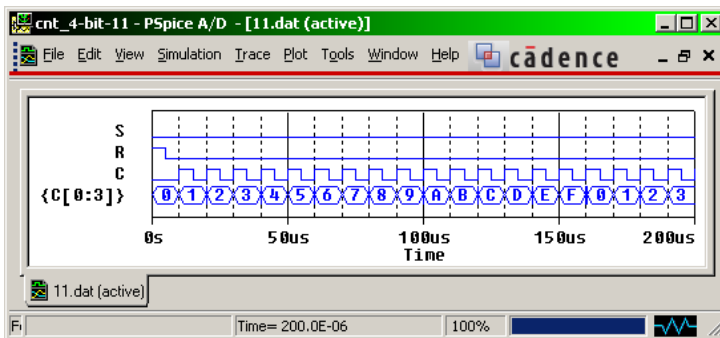


Fig. 3.3.12 – Timing diagram of the counter

Simulation of a 16-bit register

In Fig. 3.3.13 shows a diagram of a 16-bit register based on *D* flip-flop.

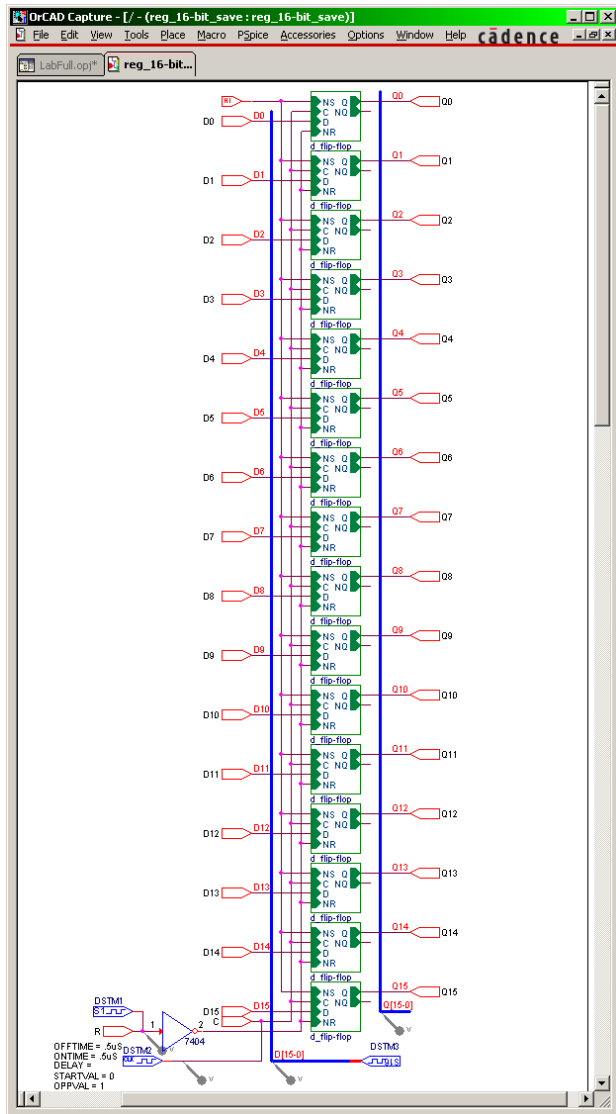


Fig. 3.3.13 – Register

We determine the delay period (Fig. 3.3.14) and enter the data in Table 3.4.

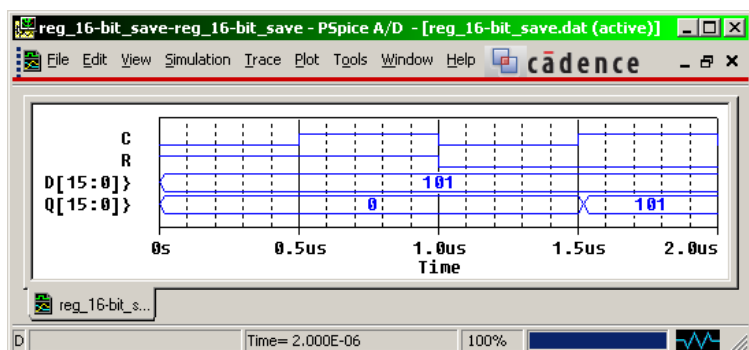


Fig. 3.3.14 – Timing diagram of the register

Table 3.4

Device	Delay period T_s , ns
4-bit shift register	3
adder	116.5
counter	79
16-bit parallel register	18

Laboratory work 4

Modeling elements of the fourth hierarchical level

The purpose of the work: develop a functional diagram of the device. Acquire and consolidate practical skills in designing and modeling fourth-level hierarchical elements in OrCAD.

Tasks for laboratory work: design and model fourth-level hierarchical elements.

Example of task completion

The fourth-level hierarchical elements for the given device are: a 16-bit shift register; an 8-bit register; a 16-bit adder. These circuits are implemented on third-level hierarchical elements.

Modeling of a 16-bit shift register

Fig. 3.4.1 shows a diagram of a 16-bit register with shift, parallel write, and reverse synchronization.

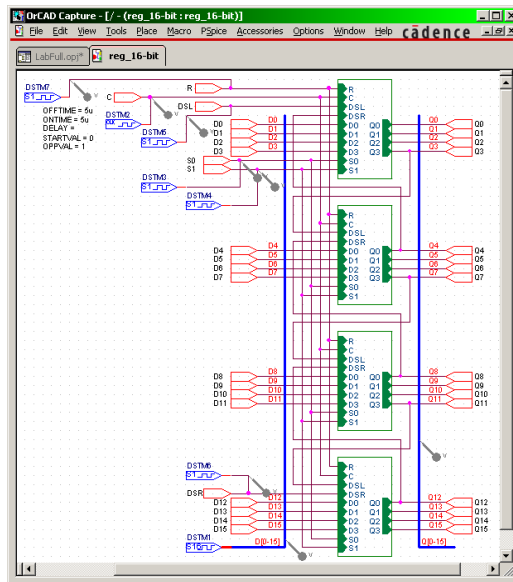


Fig. 3.4.1 – Register circuit

The element named *DSTM1* of type *STIM16* from the *SOURCE* library is analogous to *STIM4*, but is capable of "outputting" a 16-bit number. We determine the delay time (37 ns) for output Q [0-15] (Fig. 3.4.2).

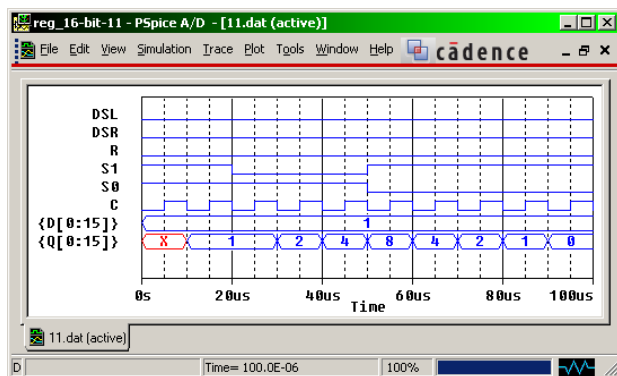


Fig. 3.4.2 – Timing diagram of the register

Modeling an 8-bit register

In Fig. 3.4.3 shows a diagram of an 8-bit register with shift, parallel write, and reverse synchronization. We determine the delay time (37 ns) for output Q [0-7] (Fig. 3.4.4).

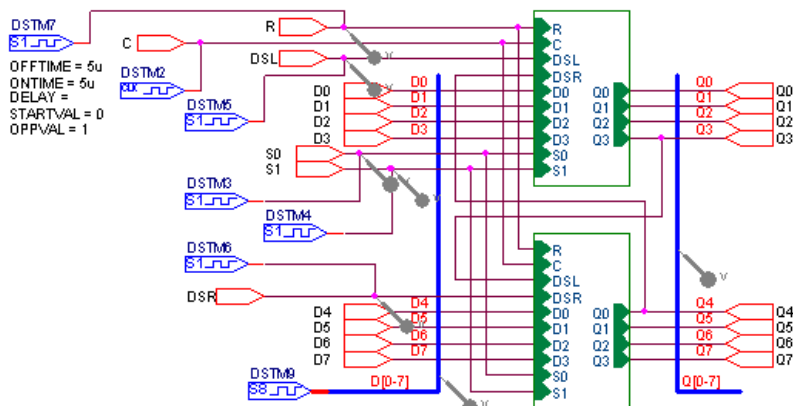


Fig. 3.4.3 – Schema of an 8-bit register

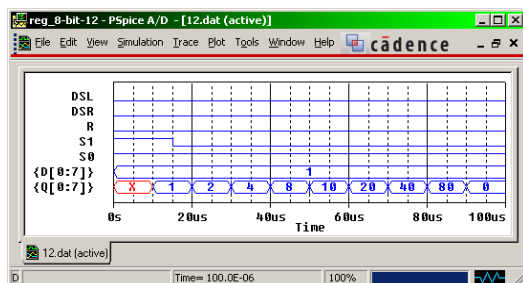


Fig. 3.4.4 – Timing diagram

Modeling of a 16-bit adder

In Fig. 3.4.5 shows the schema of 16-bit adder. We determine the delay time (Fig. 3.4.6) and enter this data in Table 3.5.

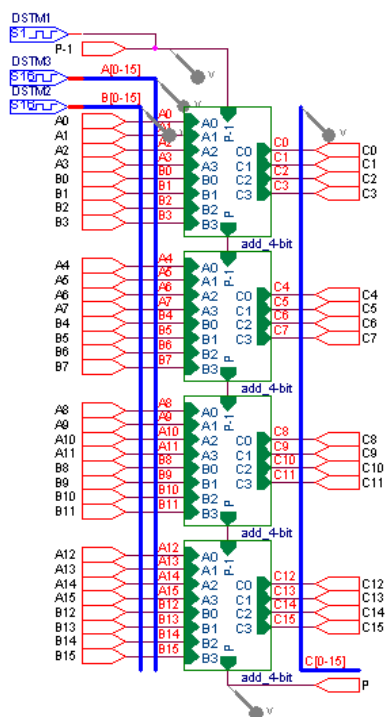


Fig. 3.4.5 – Adder circuit

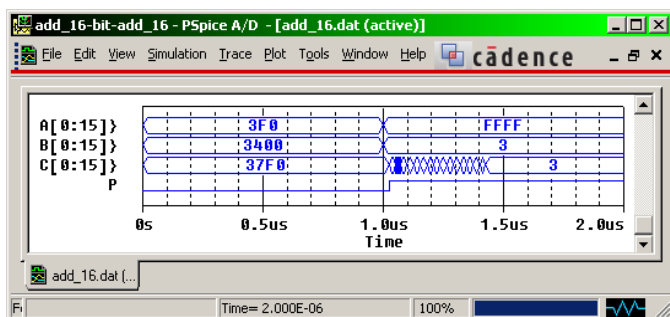


Fig. 3.4.6 – Timing diagram of the adder

Table 3.5

Device	Delay period T_s , ns
16-bit register	37
8-bit register	37
16-bit adder	422.5

Laboratory work 5

Modeling and analysis of the correct functioning of the developed device

The purpose of the work: perform simulation of the developed device circuit and analyze its functioning in OrCAD.

Tasks for laboratory work: perform modeling and analysis of the developed device functionality.

Example of task execution

The circuit of the developed device is based on all hierarchical levels that were considered in previous laboratory works (Fig. 3.5.1).

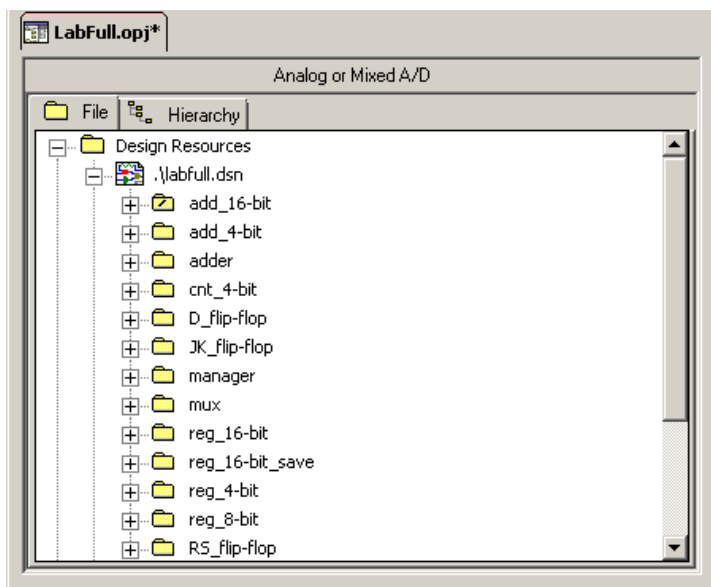


Fig. 3.5.1 – File structure type with hierarchy

The diagram is created based on the functional diagram (Fig. 3.1.2) given in laboratory work 1 (Fig. 3.5.2). The results of the device simulation are shown in Fig. 3.5.3.



To verify the correctness of the operation, we will perform the multiplication of two numbers: $9B_{16} \times D3_{16} = 7FC1_{16}$ (Table 3.6).

Table 3.6

Clock number	Action / state
1	A = 00000000 10011011 ₂ = $9B_{16}$ B = 1101001 1 ₂ = 1 D = 0 D = A + D = $9B_{16}$
2	A(<<) = 0000000 10011011 0 ₂ = 136_{16} B(>>) = 0110100 1 ₂ = 1 D = A + D = $1D1_{16}$
3	A(<<) = 000000 10011011 00 ₂ = $26C_{16}$ B(>>) = 0011010 0 ₂ = 0
4	A(<<) = 00000 10011011 000 ₂ = $4D8_{16}$ B(>>) = 0001101 0 ₂ = 0
5	A(<<) = 0000 10011011 0000 ₂ = $9B0_{16}$ B(>>) = 0000110 1 ₂ = 1 D = A + D = $B81_{16}$
6	A(<<) = 000 10011011 0000 ₂ = 1360_{16} B(>>) = 0000011 0 ₂ = 0
7	A(<<) = 00 10011011 00000 ₂ = $26C0_{16}$ B(>>) = 0000001 1 ₂ = 1 D = A + D = 3241_{16}
8	A(<<) = 0 10011011 000000 ₂ = $4D80_{16}$ B(>>) = 0000000 1 ₂ = 1 D = A + D = $7FC1_{16}$

The values of partial sums D coincide with the simulation results (Fig. 3.5.3). After completing the calculations, the device signals readiness with an *OK* signal.

Laboratory work 6

Testing the developed device for speed

The purpose of the work: to test the developed device for speed.

Tasks for laboratory work: to study the developed device for speed.
Determine the optimal frequency of input signals.

Example of task execution

We calculate the approximate value of the maximum permissible frequency of input pulses:

$$F_{\min} = 1 / T_{\min},$$

where $T_{\min}$ is the sum of the maximum delays of the device elements.

The maximum delay time of the adder is 422.5 ns, but since the carry operation is not used in the operation of this device, the maximum delay is equal to $t_d = 125$ ns.

Thus, we obtain:

$$T_{\min} = 2 \times t_d = 2 \times 125 = 250 \text{ ns},$$

$$F_{\max} = 1 / 250 \times 10^{-9} = 4 \text{ MHz}.$$

Let's check the obtained data by setting the switching time in the synchronous pulse generator to $t = 125$ ns (Fig. 3.6.1).

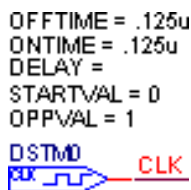


Fig. 3.6.1 – View of the generator and its parameters

For normal operation of the device, the switching time (from "1" to "0") of elements named *DSTM3* and *DSTM6* when switching the synchronizing pulse (*DSTM0*) at 5u was set to 20u ($A_S0 = B_S1 = 20u$).

For the device named *DSTM7* (*D_R*) under the same conditions, the switching time was 25u. By changing the parameters of the synchronization pulse generator (*DSTM0*), it is necessary to change the switching time for these devices.

The coefficient $k = 0.125\text{u} / 5\text{u} = 0.025$.

Switching time:

$$DSTM3 = DSTM6 = 20\text{u} \times k = 20\text{u} \times 0.025 = 0.5\text{u},$$

$$DSTM7 = 25\text{u} \times k = 25\text{u} \times 0.025 = 0.625\text{u}.$$

The simulation results are shown in Fig. 3.6.2, which shows that the result of the calculations (7FC1) at the *Q* [15:0] output coincides with the *OK* signal. This means that the device is performing its function.

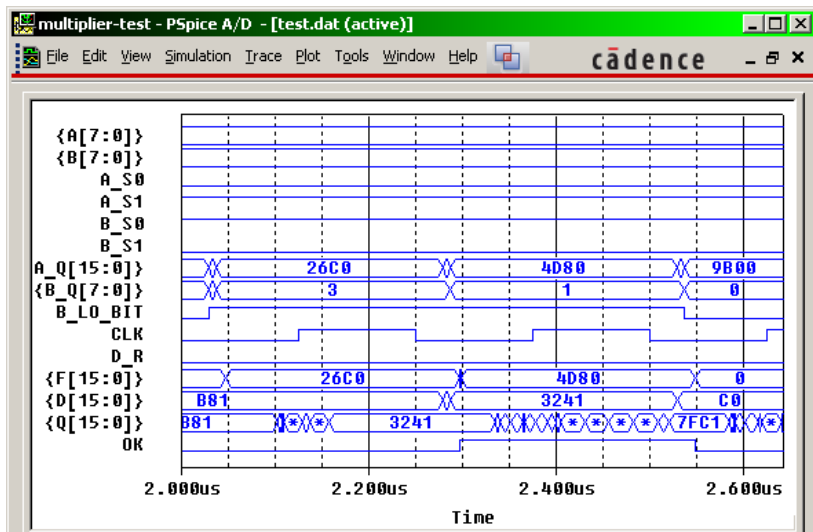


Fig. 3.6.2 – Timing diagram

If we increase the frequency to 5 MHz, we get the timing diagram shown in Fig. 3.6.3, which shows that although we observe the calculation results (7FC1) at the *Q* [15:0] output, the time of its appearance does not coincide with the *OK* signal.

Therefore, the multiplication device does not perform its function, so to increase the speed of the device, its elements must be replaced with faster ones.

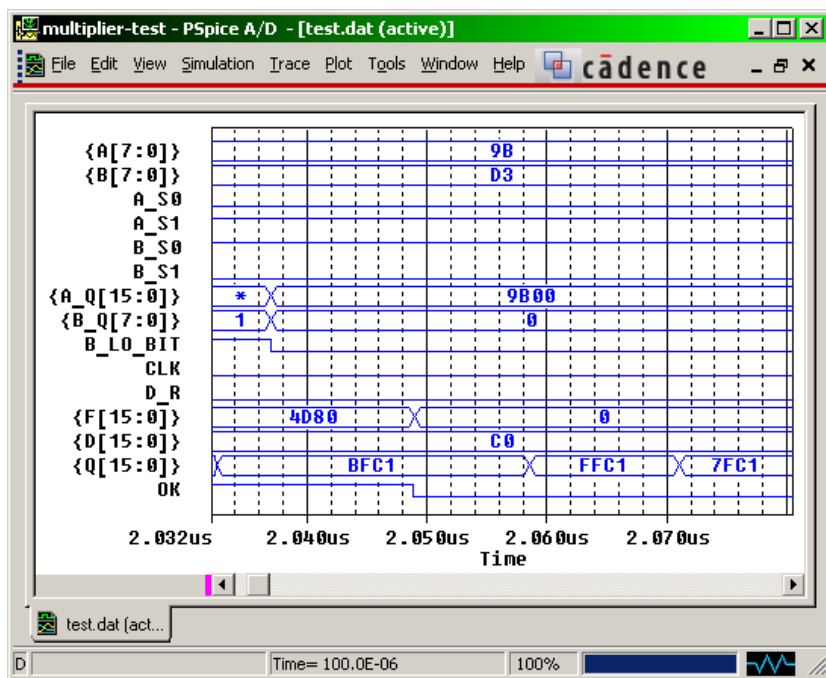


Fig. 3.6.3 – Simulation results when increasing the frequency input signals

Laboratory work 7

Assessment of the error in performing the specified operations on the developed device

The purpose of the work: to conduct research on the developed device to evaluate the error in performing specified operations.

Tasks for laboratory work: to evaluate the error in performing specified operations on the developed device and on a device that performs similar operations on analog blocks.

Example of task execution

To evaluate the error in performing specified operations, we assemble the circuit shown in Fig. 3.7.1. Two analog signals are fed through an analog-to-digital converter (ADC) to the device inputs. The outputs of the device are connected to a digital-to-analog converter (DAC). In parallel, these analog signals are multiplied by an analog multiplication block. This allows us to see the error in the execution of the specified operation.

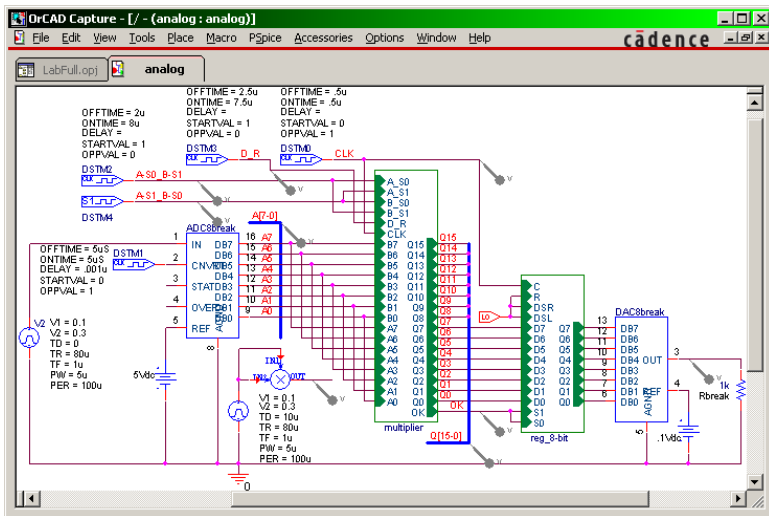


Fig. 3.7.1 – Multiplication device diagram

The DAC and ADC are located in the *BREAKOUT* library and are called *DAC* and *ADC*. The analog multiplication unit is located in the *ABM* library and is called *MULT*. The reference voltage of the DAC at the output of the digital multiplier is calculated using the formula:

$$V(REF) = \frac{V(OUT) \cdot 2^m}{DB},$$

where m is the number of binary digits, DB is the digital code at the input, and $V(OUT)$ is the required voltage at the output.

For the circuit in Fig. 3.55, we have:

$V(OUT) = 5 \times 5 = 25$ (V) – this is the voltage that occurs when two signals with a voltage of 5 V are multiplied:

$$V(REF) = \frac{25 \cdot 255}{254} \approx 25,1 \text{ (V)}.$$

In Fig. 3.7.2, the stepped signal is the result of digital multiplication, and the smooth line next to it is the result of analog multiplication. The error is the difference between the graphs of the results of analog and digital multiplication.

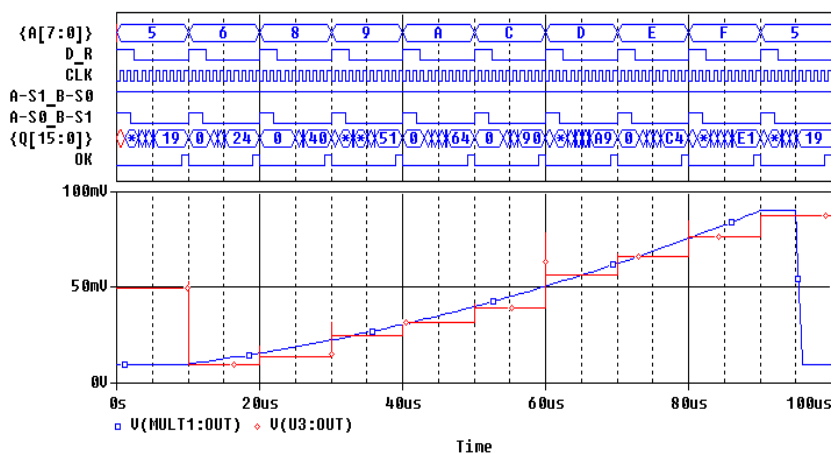


Fig. 3.7.2 – Simulation results

Laboratory work 8

Using Simulation Manager to control simulation processes

The purpose of the work: to examine the capabilities of *Simulation Manager*.

Tasks for laboratory work: simulate two device circuits in turn, controlling the voltage values, and record the results in a file for the specified mode according to Table 3.7.

Table 3.7 – Individual tasks

№	Schematic № 1		Schematic № 2	
	Device	Mode	Device	Mode
1	5AND-NOT	Transient	RS flip-flop	AC sweep
2	3-bit adder	DC sweep	6-bit adder	Transient
3	D flip-flop	AC sweep	encoder 8×3	DC sweep
4	6OR	Transient	D flip-flop	DC sweep
5	JK flip-flop	AC sweep	7AND-NOT	Transient
6	multiplexer 5× 1	DC sweep	4-bit register	AC sweep
7	D flip-flop	Transient	6OR-NOT	Transient
8	multiplexer 6× 1	DC sweep	2-bit adder	Transient
9	RS flip-flop	AC sweep	3OR-NOT	DC sweep
10	5OR-NOT	Transient	D flip-flop	AC sweep
11	2-bit adder	Transient	multiplexer 3×1	AC sweep
12	4OR-NOT	DC sweep	JK flip-flop	Transient
13	JK flip-flop	AC sweep	RS flip-flop	DC sweep
14	2-bit adder	Transient	6-output decoder	DC sweep
15	RS flip-flop	AC sweep	8OR-NOT	Transient

Example of task execution

The *PSpice* program includes the *Simulation Manager* program, which provides a familiar and easy-to-use interface for managing multiple simulation processes. The *PSpice Simulation Manager* dialog box is shown in Fig. 3.8.1.

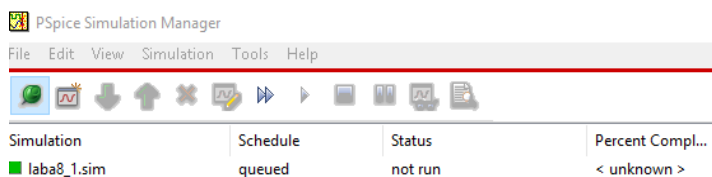


Fig. 3.8.1 – *Simulation Manager* window

The *Simulation Manager* program allows you to perform the following operations:

- change the order of simulations in the queue;
- add or delete simulations;
- monitor the progress of simulations;
- start, stop or interrupt simulations.

To execute the queue in *Simulation Manager* in the *Capture* schematic editor on the *Probe Window* tab of the *Simulation Setting* dialog box (Fig. 3.8.2), uncheck the *Display Probe Window* box (ignore automatic *Probe* launch).

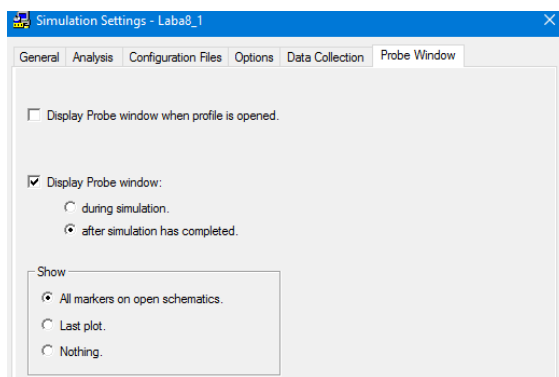


Fig. 3.8.2 – *Probe Window* tab

To see the simulation results, you need to load it manually. The *Simulation Manager* program is launched either from *PSpice* or each time a new simulation is started, after which it remains active throughout the simulation.

You can also launch the *Simulation Manager* program from the Windows *Start* menu. The colored icons in the *Schedule* column to the left of the file name show the current status of the simulation. Table 3.8 provides explanations for the icons that show the status of the simulation.

Table 3.8 – Explanation of icons

	Simulation is stopped in an unfinished state and cannot be continued
	Simulation is currently running
	Simulation is temporarily paused and can be resumed
	Simulation stopped in an unfinished state and cannot be continued

Table 3.9 provides explanations for the comments in the *Schedule* column, and Table 3.10 provides explanations for the comments in the *Status* column.

Table 3.11 describes the functional capabilities of *Simulation Manager* for each version of *PSpice*.

Table 3.9 – Explanations of comments in the *Schedule* column

Comments	Explanation
<i>queued</i>	Simulation in queue, will be executed in the order in which it is listed in the queue
<i>running</i>	Simulation is running and occurring at this time, reflected in the <i>status</i> column information
<i>on hold</i>	Simulation is temporarily suspended
<i>stopped</i>	Simulation is complete or has been stopped due to an error

Table 3.10 – Explanation of comments in the *Status* column

Status	Explanation
<i>Not run</i>	Simulation has not yet been run (default setting)
< <i>Status</i> >	Basic information from the < <i>status</i> > column about the progress of the analysis, which is displayed for simulations that are currently running
<i>paused</i>	Simulation is temporarily paused (manually or automatically)
<i>Complete–no errors</i>	Simulation completed without errors
<i>errors</i>	Simulation completed partially and stopped because errors were detected
<%>	Percentage of the modeling process completed (increases as modeling progresses)

Table 3.11 – *Simulation Manager* functional capabilities

<i>PSpice</i> versions	Functionality
<i>PSpice Demo, PSpice A/D Basics</i>	Only one simulation can be run or paused (execution order: one after another)
<i>PSpice, PSpice A/D</i>	One simulation can be executed and multiple simulations can be paused (execution queue: one after another)

Let's look at how *Simulation Manager* identifies errors during the simulation process.

Emergency simulation – if simulation is impossible for any reason, *Simulation Manager* will stop the development of changes. Then, after a certain period of time, *Simulation Manager* will stop the simulation and automatically start the next task in the queue.

Temporary simulation suspension – if a temporary simulation suspension occurs automatically and requires manual intervention to continue, *Simulation Manager* will automatically start the next task in the queue.

Pause in simulation – if simulation is stopped automatically, *Simulation Manager* will automatically start the next task in the queue.

In *Simulation Manager*, you can set any number of simulation series to run one after another. At the same time, you can run another task in *PSPice*. Each new simulation will be assigned the *queued* category. This simulation will be executed after all other tasks in the queue have been completed. If a task is added, you can change its position in the queue: start, stop, pause, or make other changes to its status.

To add a simulation to the queue:

1. Select *File – Add Simulation* or click the *Add Simulation* button on the toolbar.
2. In the *Add Simulation* window, double-click on a file with the extension *.SIM or *.CIR.

In *Simulation Manager*, you can easily manage a series of simulations in the queue. To start a simulation from *Simulation Manager*:

1. Select a simulation from the list.
2. In the *Simulation* menu, select *Run* or click the *Run Selected* button on the toolbar.

To stop a simulation in *Simulation Manager*:

1. Select the simulation that is currently running.
2. Select *Stop* from the *Simulation* menu or click the *Stop Selected* button on the toolbar.

To temporarily pause a simulation from *Simulation Manager*:

1. Select the simulation that is currently running.
2. Select *Pause* from the *Simulation* menu or click the *Pause Selected* button on the toolbar.

If you need to display the simulation results for tasks selected from the list in *Simulation Manager*, you need to run the simulation in *PSPice*.

To start simulation in *PSPice*:

1. Select the simulation you want to run in *PSPice*.
2. In the *View* menu, select *Simulation Results*.

The *PSPice* program is activated and the results of the selected simulation

are displayed in *Probe*. If a simulation is currently running, you can observe the change in signals in the graph window. Each time a new simulation is added while the previous one is running, *Simulation Manager* must decide how to handle the new task. By default, the new simulation is added to the end of the queue, and the task currently being simulated continues to run.

You can change the default settings so that *Simulation Manager* starts immediately with each new simulation and stops or pauses any currently running tasks.

Options can be selected according to the explanations below (the *Options* dialog box can be displayed each time new simulations are added, or it can be hidden). For the default settings for *Simulation Manager*, select *Options* from the *Tools* menu – a dialog box will appear (Fig. 3.8.3).

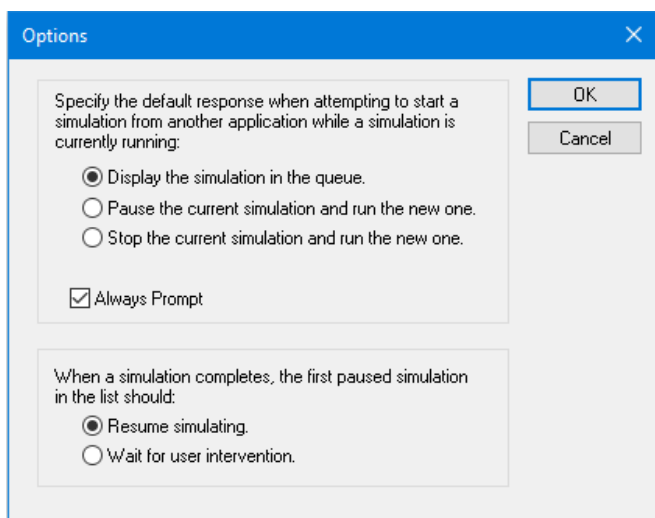


Fig. 3.8.3 – *Options* dialog box

If necessary, the *Options* dialog box will appear each time a simulation is added – for this, the *Always Prompt* checkbox must be selected (default setting). A description of some of the *Simulation Manager* commands is given in Table 3.12.

Table 3.12 – Description of *Simulation Manager* commands

Command	Purpose
File menu	
<i>Add Simulation</i>	Open simulation files (.SIM) or circuit files (.CIR)
View Menu	
<i>Simulation Results</i>	Displaying simulation results in <i>PSpice</i>
<i>Output File</i>	Open the output file for the selected simulation and display it in <i>PSpice</i>
Simulation Menu	
<i>Run Queued Items</i>	Execute all simulations in the queue
<i>Queue Selected</i>	Change all selected simulations
<i>Reset Queue</i>	Reset all simulations
<i>Edit Settings</i>	Open the profile for the selected simulation (to change the analysis settings)
Tools menu	
<i>Options</i>	Allows you to change the default settings

Saving a project, schematic or library

The *Capture* schematic editor provides functionality that allows you to easily save and archive projects at every stage of development. The *Save* command saves all open documents that refer to the project. When projects are saved together with schematics and libraries, the corresponding properties are saved, so there is no need to change positional designations.

The *Save As* command saves files depending on the selection in the *Project Manager*:

- if one or more schematics (or libraries) are selected, you can save each file with changes;
- if no schematics or libraries are selected in *Project Manager*, you can quickly save the project.

When a project is saved, *Capture* automatically creates a backup file with the extension *.DBK. When a library is saved, *Capture* automatically creates a backup file with the extension *.OBK. If a schematic page or symbol is saved,

no backup file is created. In *Capture*, a schematic file (*.DSN) is always accompanied by a project file (*.OPJ).

Archiving with *Capture* Schematic Editor

You can save the project (*.OPJ) and all associated files in different directories, as well as create a zip-file. You can also archive external projects, global *PSpice* model libraries, and global files included in the *PSpice* project or component reference data. You can use the *Archive Project* command in the *File* menu to archive a project. This command allows you to save all files related to the project in the directory specified for archiving and in a single *.zip file.

To archive a project:

1. Make sure that the project to be archived is active and that the schematic pages of this project are not open.
2. Select *Archive Project* from the *File* menu. The *Archive Project* dialog box will appear (Fig. 3.8.4).

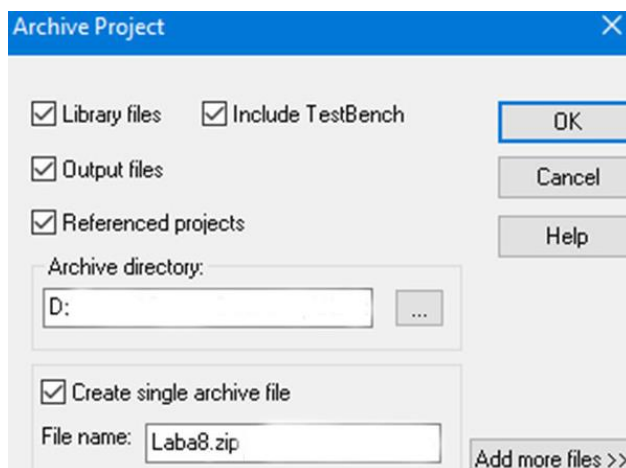


Fig. 3.8.4 – *Archive Project* dialog box

3. Select the files you want to include in the project. If no options are selected (*Library files*, *Output files*, or *Referenced projects*), *Capture* will

archive only the project files (*.OPJ) and schematics (*.DSN) by default. The *Include TestBench* option generates a report of all files and simulations. For *PSpice* projects, simulation profiles and local files (*.LIB, *.STL, *.INC) will always be archived together with the project. The *Output files* option also archives output files for *PSpice* projects, such as *.DAT and *.OUT.

4. Select the directory where you want to save the files. The *Select Directory* dialog box will appear (you can also enter the path to the archive directory in the *Archive directory* text field).

5. Select the directory where you want to archive the project and click *OK*.

6. Click *OK* in the *Archive Project* dialog box. *Capture* archives the project with the selected files in the specified directory and displays any errors in the *Session Log*. When you create a new archive directory, the current directory does not change.

The settings specified in the *Archive Project* dialog box are saved in the *CAPTURE.INI* file (these settings will be used when the next archiving session starts).

Other output options

Capture uses a special symbol, *WATCH1*, which allows you to monitor voltage values for three circuit modes: *DC sweep*, *AC sweep*, or *Transient*. The results are displayed in *PSpice*.

To display voltage values in the *PSpice* window:

1. Place and connect the *WATCH1* symbol (from the *PSpice* SPECIAL.OLB library) to the analog circuit (Fig. 3..8.5).

2. Double-click on the *WATCH1* symbol to display the *Parts* spreadsheet.

3. In the *ANALYSIS* properties column, enter DC, AC, or TRAN (*transient*) for the type of analysis whose results you want to observe.

4. In the *LQ* and *HI* property columns, enter values corresponding to the lower and upper voltage limits for this point in the circuit. If the results move beyond certain limits, *PSpice* temporarily stops the simulation so that the behavior can be evaluated.

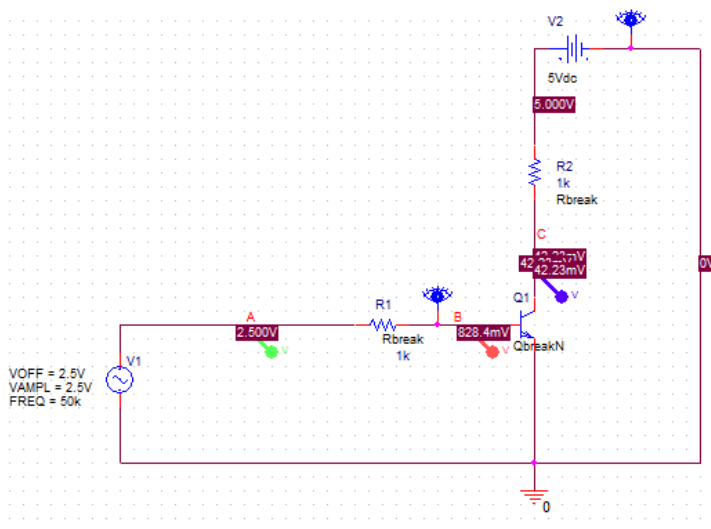


Fig. 3.8.5 – Research circuit

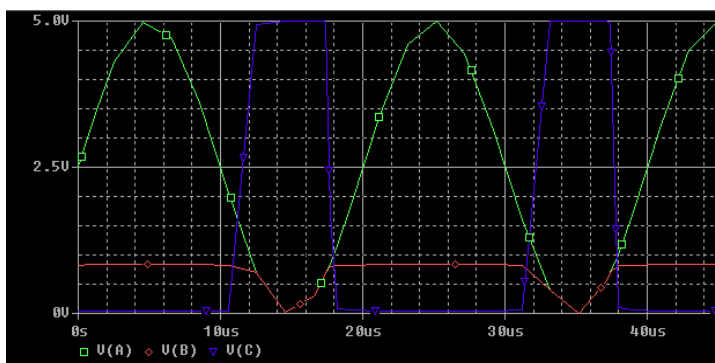


Fig. 3.8.6 – Simulation results

Writing additional results to the PSpice output file

Capture has additional symbols that allow you to save additional simulation results to the *PSpice* output file for printing graphs and tables.

To display the *PSpice* output file after simulation, select *Examine Output* from the *Simulation* menu. You can generate voltage and current graphs for any type of analysis: DC sweep, AC sweep or Transient.

To generate voltage or current graphs in the output file, do the following:

1. Place and connect any symbols from Table 3.13 (from *PSpice* – SPECIAL.OLB library).

Table 3.13 – Symbols for displaying graphs

Symbol	The graph displays
	Voltage at the section of the circuit to which the symbol is connected
	Differential voltage between the two circles to which the symbol is connected
	Current

2. Double-click on the symbol to display the *Parts* spreadsheet.
3. Select the type of analysis you want to display: DC, AC or TRAN.
4. In the *Analysis type* column to be displayed (DC, AC or TRAN), enter any value (Y, YES or "1").
5. If AC analysis type was selected, define the output format:
 - 5.1 Select one of the following output formats: MAG (amplitude), PHASE, REAL, IMAG (imaginary) or DB.
 - 5.2 Enter any value (Y, YES or "1").
 - 5.3 Repeat steps 5.1 – 5.2 for the other selected AC analysis output formats. If the format is removed, PSpice will select MAG by default.
6. Repeat steps 2 – 4 for other types of analysis that need to be displayed.

To generate a digital state change table in the output file, place the PRINTDGTLCG symbol (from the *PSpice* SPECIAL.OLB library) and connect it to the circuit.

Capture has a VECTOR symbol that allows you to save the results of digital simulation in a vector file (when the circuit with the VECTOR symbol changes state, the data is written to the vector file).

To create a test vector file for the circuit, do the following:

1. Connect the VECTOR symbol to the conductor (bus) at the output of the digital symbol sample.

2. Double-click on VECTOR to display the *Parts* spreadsheet.
3. Set the symbol properties as described in Table 3.14.
4. Repeat steps 1–3 for the numerical test vectors to be created.

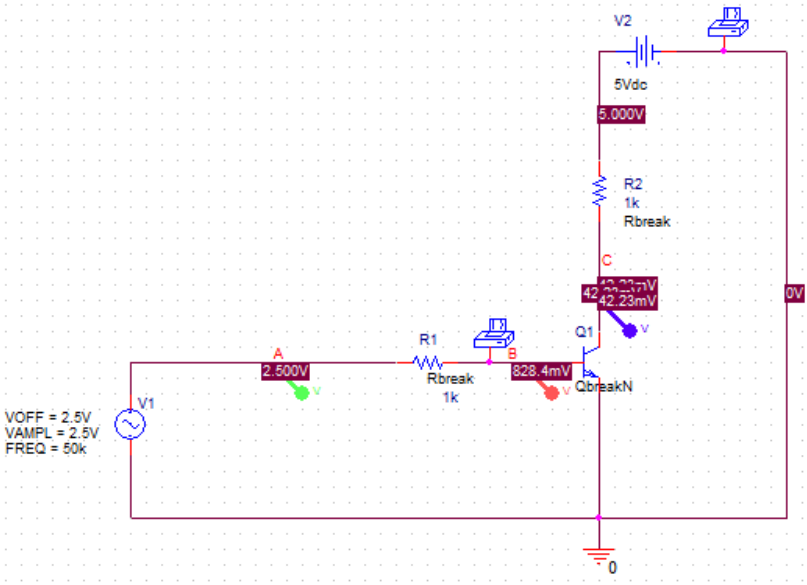


Fig. 3.8.7 Setting the VECTOR symbol

Table 3.14 – VECTOR font

Properties	Definition
POS	Column position in the file (valid range of values 1...25)
FILE	Name of the vector file
RADIX	Base of the number system
BIT	When the VECTOR symbol is attached to a conductor, the position of the bit among six or eight single digits
SIGNAMES	Names of signals that appear in the file header

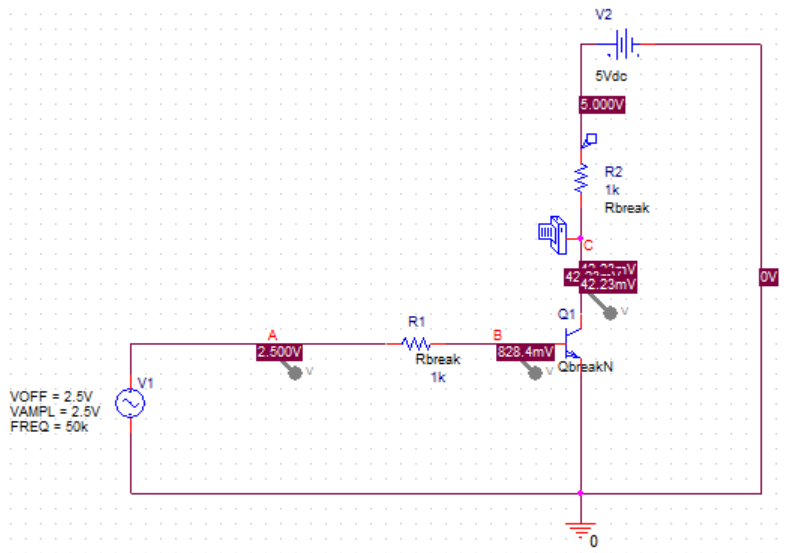


Fig. 3.8.8 Different settings for the VECTOR symbol

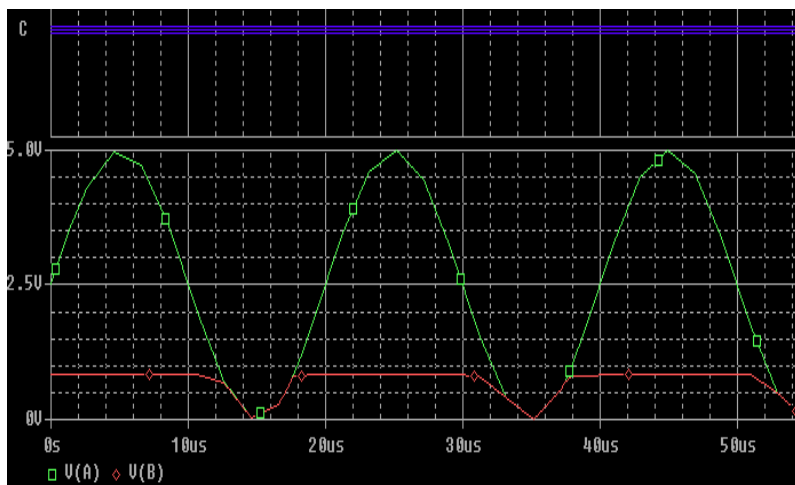


Fig. 3.8.9 – Simulation results

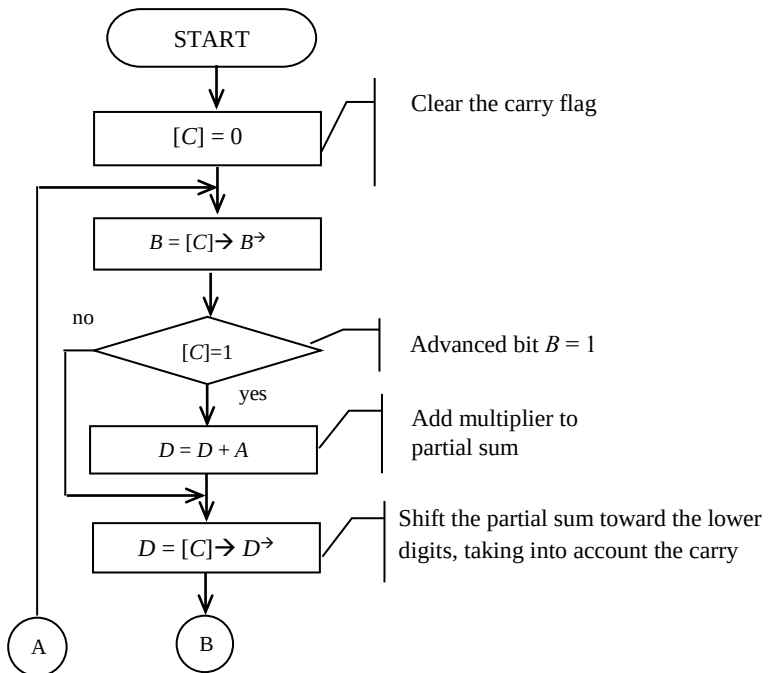
REFERENCES

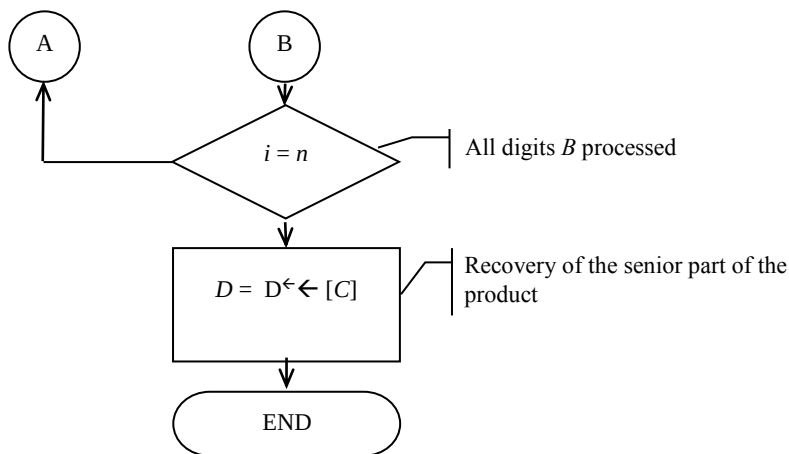
1. Технологія автоматизованого проектування комп'ютерних систем та їх складових. Навчальний посібник для студентів спеціальності «Комп'ютерна інженерія» / С.Ю. Леонов, Г.В. Гейко. – Харків: НТУ «ХПІ», 2024. – 256 с.
2. Methodological instructions for laboratory works in the «Computer systems» academic discipline for full-time and part-time students majoring in «Computer Engineering» / S. Leonov, H. Heiko. – Kharkiv: NTU «KhPI», 2025. – 141 p.
3. Автоматизоване проектування пристроїв в комп'ютерних системах: навчальний посібник / С.Ю. Леонов, Т.В. Гладких. – Харків: НТУ «ХПІ», 2024. – 256 с.
4. Цифрова схемотехніка. Моделювання та аналіз. Видання друге, виправлене та доповнене. [Електронний ресурс]: навчальний посібник для студентів спеціальності 171 «Електроніка» / В.В. Макаренко, В.М. Співак. – Київ: КПІ ім. Ігоря Сікорського, 2021. – 490 с.
5. Мікропроцесори та цифрова електроніка. [Електронний ресурс]: навчальний посібник для студентів спеціальності 141 «Електроенергетика, електротехніка та електромеханіка» / К.К. Победаш, В.А. Святненко. – Київ: КПІ ім. Ігоря Сікорського, 2021. – 120 с.
6. Ларін В.Ю., Харченко В.П. Автоматизація схемотехнічного проектування: підручник / Київ: НАУ, 2017. – 190 с.
7. Програмні засоби проектування: навч. посіб. для студ. спеціальності 6.050803 «Акустотехніка» спеціалізації «Аудіо-, відео- та кілотехніка» та 171 «Електроніка» спеціалізації «Електронні та інформаційні системи і технології телебачення, кінематографії та звукотехніки» / В.В. Макаренко. – Київ: КПІ ім. Ігоря Сікорського, 2018. – 244 с.
8. OrCAD Capture. User's Guide / Cadence Design Systems, 2000. – 374 p.
9. OrCAD Flow Tutorial / Cadence Design Systems, 2004. – 106 p.

APPENDIX

Multiplication algorithm 1

The multiplication command is implemented by step-by-step addition of the shifting multiplier value and the partial sum, provided that the corresponding digit of the multiplier has a non-zero bit. This algorithm is characterized by the fact that registers of the same size are used for the multiplier and the partial sum. The result (product) is stored in a pair corresponding to the partial sum and the multiplier, which is gradually filled with zeros, with the higher part stored in the partial sum register and the lower part in the multiplier register. Consider an algorithm in which A and B are multipliers and D is a partial sum.





Example

$A = 6_{10} = 0110_2$ (multiplier); $B = 3_{10} = 0011_2$ (multiplier);

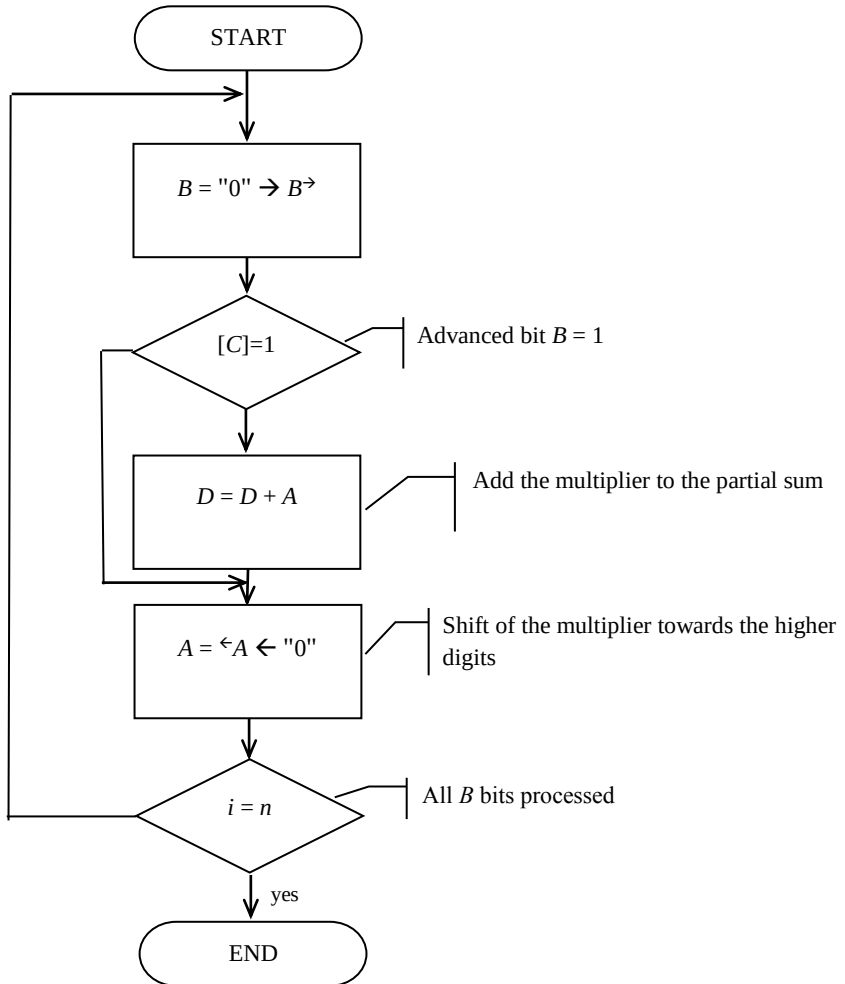
$D = 0000_2$ (partial sum)

Clock number	Action		Flag [C]
1	Shift B to the right	$B = [0]001 1$	1
2	$[C] = 1$, then add the multiplier to the partial sum	$D = D + A$ $\begin{array}{r} 0000 \\ + 0110 \\ \hline 0 0110 \end{array}$	0
3	Shift D to the right	$D = [0]011 0$	0
4	Shift B to the right	$B = [0]000 1$	1
5	$[C] = 1$, then add the multiplier to the partial sum	$D = D + A$ $\begin{array}{r} 0011 \\ + 0110 \\ \hline 0 1001 \end{array}$	0
6	Shift D to the right	$D = [0]100 1$	1
7	Shift B to the right	$B = [1]000 0$	0
8	Shift D to the right	$D = [0]010 0$	0
9	Shift B to the right	$B = [0]100 0$	0
10	Shift D to the right	$D = [0]001 0$	0
11	Shift B to the right	$B = [0]010 0$	0
12	Shift D to the right	$D = [0]000 1$	1
13	Shift D to the left	$D = 0 000[1]$	0

Result in $D:B : 0001:0010$. We see that $00010010_2 = 18_{10}$.

Multiplication algorithm 2

The algorithm is characterized by the fact that double capacity registers are used for the first factor and partial sum, and single capacity registers are used for the second factor, which is equal to N . The result is stored in the partial sum register. Consider an algorithm in which A and B are factors, and D is the partial sum.



Example

$A = 6_{10} = 00000110_2$ (multiplier); $B = 3_{10} = 0011_2$ (multiplier);

$D = 00000000_2$ (partial sum)

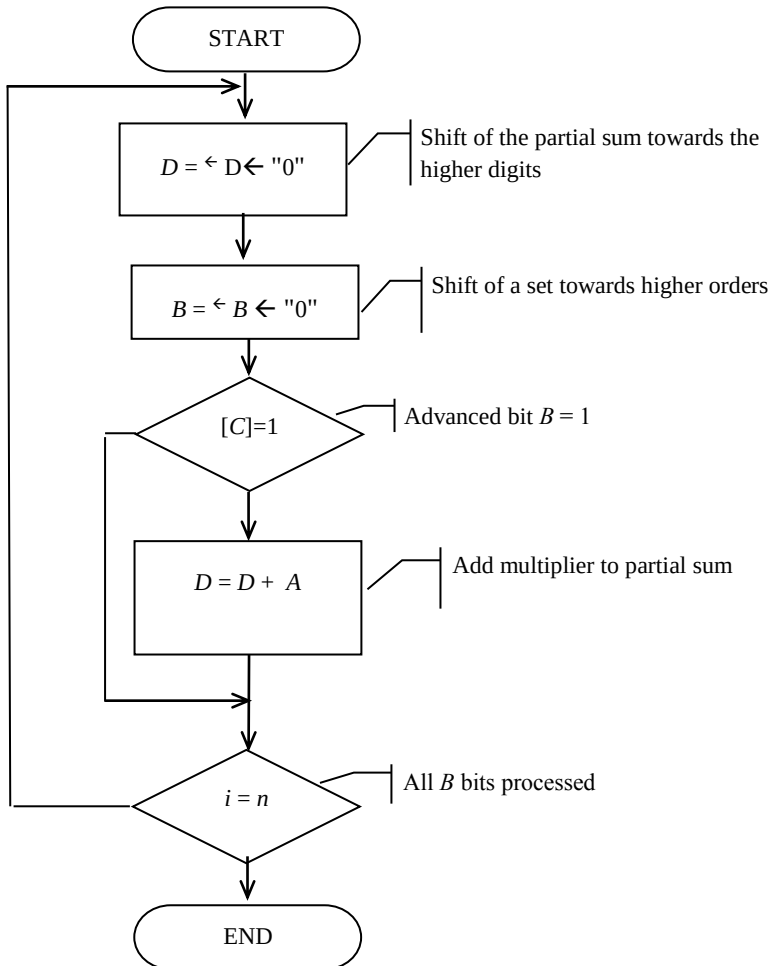
Clock number	Action		Flag [C]
1	Shift B to the right	$B = 0001 1$	1
2	$[C] = 1$, then add the multiplier to the partial sum	$D = D + A$ $\begin{array}{r} 00000000 \\ + 00000110 \\ \hline 0 0000110 \end{array}$	0
3	Shift A to the left	$A = 0 00001100$	0
4	Shift B to the right	$= 0000 1$	1
5	$[C] = 1$, then add the multiplier to the partial sum	$D = D + A$ $\begin{array}{r} 00000110 \\ + 00001100 \\ \hline 0 00010010 \end{array}$	0
6	Shift A to the left	$A = 0 00011000$	0
7	Shift B to the right	$= 0000 0$	0
8	Shift A to the left	$A = 0 00110000$	0
9	Shift B to the right	$= 0000 0$	0
10	Shift A to the left	$A = 0 01100000$	0

Result in D : 00010010. We see that $00010010_2 = 18_{10}$.

Multiplication algorithm 3

The algorithm is characterized by the fact that a double capacity registers is used for the partial sum, and a single capacity registers equal to N is used for the multipliers.

The result is stored in the partial sum register. Consider an algorithm in which A and B are multipliers, and D is the partial sum.



Example

$A = 6_{10} = 0110_2$ (multiplier); $B = 3_{10} = 0011_2$ (multiplier);

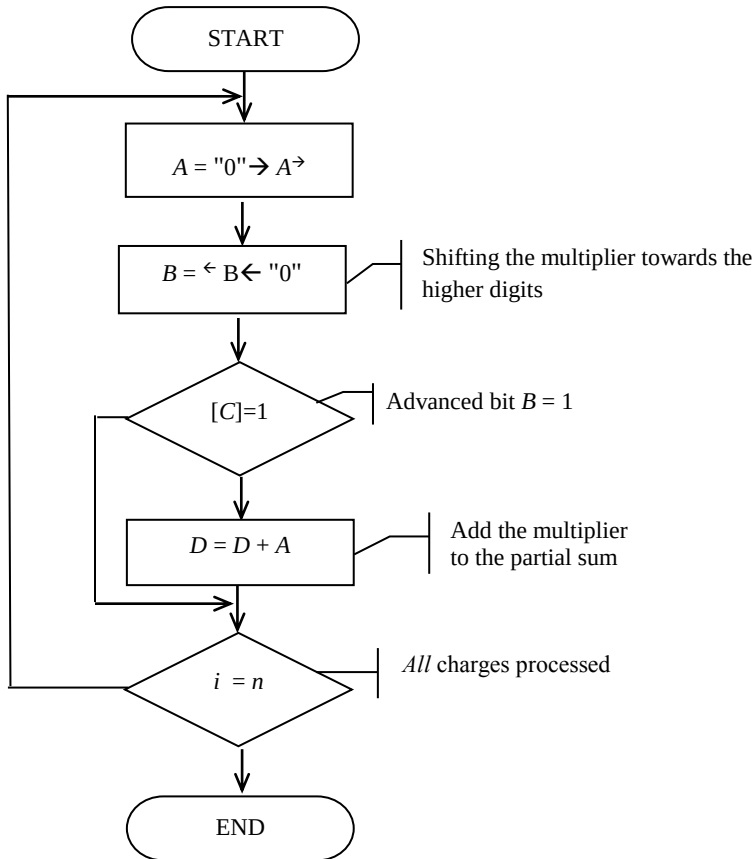
$D = 00000000_2$ (partial sum).

Clock number	Action		Flag [C]
1	Shift D to the left	$D = 0 00000000$	0
2	Shift B to the left	$B = 0 0110$	0
3	Shift D to the left	$D = 0 00000000$	0
4	Shift B to the left	$B = 0 1100$	0
5	Shift D to the left	$D = 0 00000000$	0
6	Shift B to the left	$B = 1 1000$	1
7	[C] = 1, then add the multiplier to the partial sum	$D = D + A$ $\begin{array}{r} 00000000 \\ + 00000110 \\ \hline 0 00000110 \end{array}$	0
8	Shift D to the left	$D = 0 00001100$	0
9	Shift B to the left	$B = 1 0000$	1
10	[C] = 1, next add the multiplier to the partial sum	$D = D + A$ $\begin{array}{r} 00001100 \\ + 00000110 \\ \hline 0 00010010 \end{array}$	0

Result in D : 00010010. We see that $00010010_2 = 18_{10}$.

Multiplication algorithm 4

The algorithm is characterized by the fact that double capacity registers are used for the partial sum and the register of the first multiplier, and single capacity registers equal to N are used for the second multiplier. In addition, the value of the first multiplier is written to the upper part of the register in order to shift it towards the lower digits, gradually decreasing its value. The analysis of the multiplier's digits is performed starting with the most significant ones. The result is stored in the partial sum register. Consider an algorithm in which A and B are multipliers, and D is the partial sum.



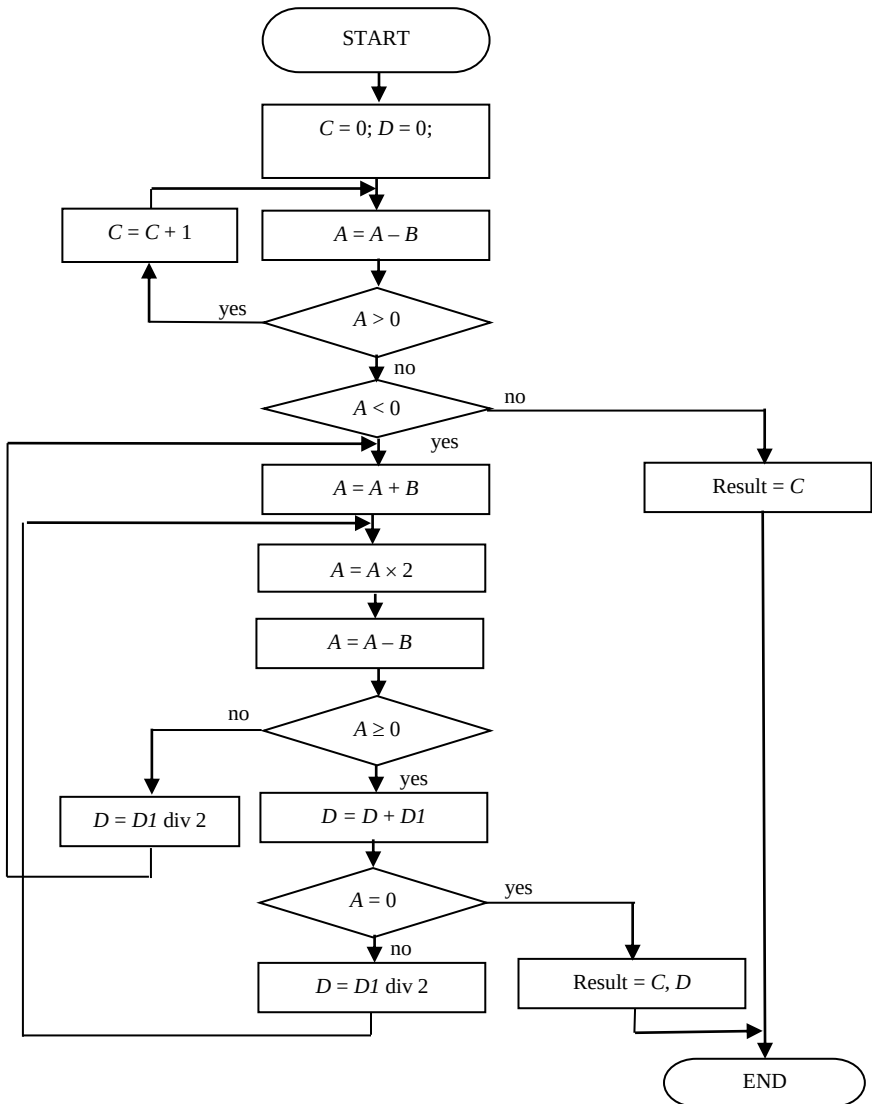
Example

$A = 01100000_2$; $B = 0011_2$; $D = 00000000_2$ (partial sum).

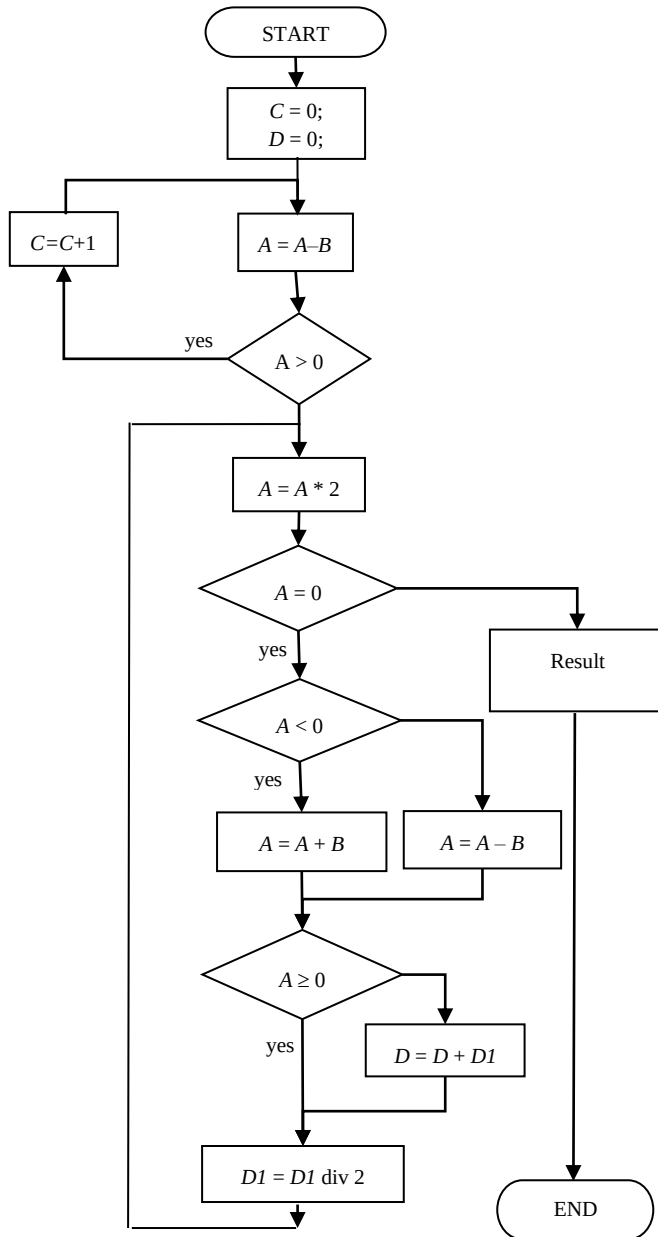
Clock number	Action		Flag [C]
1	Shift A to the right	$D = 00110000 0$	0
2	Shift B to the left	$B = 0 0110$	0
3	Shift A to the right	$D = 00011000 0$	0
4	Shift B to the left	$B = 0 1100$	0
5	Shift A to the right	$= 00001100 0$	0
6	Shift B to the left	$B = 1 1000$	1
7	[C] = 1, then add the multiplier to the partial sum	$D = D + A$ $\begin{array}{r} 00000000 \\ + 00001100 \\ \hline 0 00001100 \end{array}$	0
8	Shift A to the right	$D = 00000110 0$	0
9	Shift B to the left	$B = 1 0000$	1
10	[C] = 1, then add the multiplier to the partial sum	$D = D + A$ $\begin{array}{r} 00001100 \\ + 00000110 \\ \hline 0 00010010 \end{array}$	0

Result in D : 00010010. We see that $00010010_2 = 18_{10}$.

Division algorithm (with remainder recovery)



Division algorithm (without remainder recovery)



CONTENTS

Introduction.....	3
Chapter 1. Basics of CAD construction.....	4
1.1 Methodology for building design systems.....	4
1.2 Block-hierarchical approach to design.....	10
1.3 Technological route of ICs design.....	15
1.4 Simulation modeling.....	24
1.5 "System on a chip".....	28
1.6 Designing digital devices on PLD.....	31
1.7 Theoretical information about electromagnetic compatibility...	37
Chapter 2. Using ORCAD for device modeling.....	44
Practical work 1. Ascending and descending design	44
Practical work 2. Creating your own element based on basic components.....	61
Practical work 3. Programming a macro element.....	73
Practical work 4. Using the PSpice program of the ORCAD system for analog modeling of digital blocks.....	89
Practical work 5. Using the Stimulus Editor program for testing the device.....	103
Practical work 6. Using ORCAD to model digital blocks based on the Monte Carlo method.....	129
Practical work 7. PCB tracing.....	139
Practical work 8. Research on the operation of devices taking into account electromagnetic compatibility.....	156
Chapter 3. Using ORCAD for modeling complex devices.....	180
Laboratory work 1. Developing a functional diagram of the device. Break down the diagram into hierarchical levels. Modeling elements of the first hierarchical level.....	180
Laboratory work 2. Modeling elements of the second hierarchical level.....	188
Laboratory work 3. Modeling elements of the third hierarchical level.....	197
Laboratory work 4. Modeling elements of the fourth hierarchical level.....	205

Laboratory work 5. Modeling and analysis of the correct functioning of the developed device.....	209
Laboratory work 6. Testing the developed device for speed.....	212
Laboratory work 7. Assessment of the error in performing the specified operations on the developed device.....	215
Laboratory work 8. Using Simulation Manager to control simulation processes.....	217
References	230
APPENDIX	231

