NATIONAL UNIVERSITY OF LIFE AND
ENVIRONMENTAL SCIENCES OF UKRAINE

Faculty of Information Technologies

Department of computer systems, networks and cybersecurity

Kulinich O.M., Kasatkin D.Yu., Lakhno V.A., Nikitenko Y.V., Kharchuk N.S.

DESIGN, IMPLEMENTATION AND MAINTENANCE OF COMPLEX SYSTEMS OF INFORMATION PROTECTION

Проектування систем обробки та захисту інформації.

Навчальний посібник (англійською мовою)

Recommended for publication by the Academic Council of the Faculty of Information Technologies (protocol No. from 2024)

Kyiv – 2024

UDC 621.3.049.77.001.63

P79

Recommended for publication by the Academic Council of NULES of Ukraine (protocol No. from 2024)

Reviewers:

Anna Korchenko, Doctor of Technical Sciences, Professor of the Department of Information Technology Security, National Aviation University;

Olena Kryvoruchko, Doctor of Technical Sciences, Professor, Head of the Department of Software Engineering and Cyber Security, State University of Trade and Economics;

Mykhailo Shvidenko, Doctor of Economics, Professor, Head of the Department of Information Systems and Technologies, National University of Bioresources and Nature Management of Ukraine;

Svitlana Amelina, Ph.D., professor, head of the department of foreign philology and translation, National University of Bioresources and Nature Management of Ukraine.

Design, implementation and maintenance of complex systems of information protection. Textbook./ Kulinich O.M., Kasatkin D.Yu., Lakhno V.A., Nikitenko Y.V., Kharchuk N.S. – K.: Publishing House «Comprint», 2024, - 438 p.

The textbook presents a modern elementary base of data security tools - the main types of memory devices and integrated circuits with a programmable structure. The textbook is intended for students studying disciplines related to the creation and operation of digital methods of data processing. It is intended for students in specialties 122 "Computer Science" and 125 "Cyber Security".

The edition was edited by the author.

УДК 621.3.049.77.001.63

П79

Рекомендовано до друку Вченою радою університету (протокол № від жовтня 2024 р.)

Рецензенти:

Корченко Анна Олександрівна, доктор технічних наук, професор кафедри безпеки інформаційних технологій Національного авіаційного університету;

Криворучко Олена Володимирівна, доктор технічних наук, професор, завідувач кафедри інженерії програмного забезпечення та кібербезпеки, Державний торговельно-економічний університет;

Швиденко Михайло Зіновійович, к.е.н., професор, завідувач кафедри інформаційних систем і технологій, Національний університет біоресурсів і природокористування України;

Амеліна Світлана Миколаївна, д.п.н., професор, завідувачка кафедри іноземної філології і перекладу, Національний університет біоресурсів і природокористування України.

Проектування, впровадження та супровід комплексних систем захисту інформації. Навчальний посібник (англійською мовою - Design, implementation and maintenance of complex systems of information protection) / Кулініч О.М., Касаткін Д.Ю., Лахно В.А., Нікітенко Є.В., Харчук Н.С. – К.: Видавництво «Компрінт», 2024, – 438 с.

В навчально посібнику представлена сучасна елементна база засобів захисту інформації – основні типи запамятовуючих пристроїв та інтегральні схеми зі структурою, що програмується. Посібник призначений для студентів, що вивчають дисципліни повязані зі створенням та експлуатацією цифрових засобів обробки інформації. Призначені для студентів спеціальності 122 «Комп'ютерні науки» та 125 «Кібербезпека».

INTRODUCTION

Attempts to formalize the work of designers and impose a strict program of actions on them are generally harmful to creativity. When we try to liken the work of information machines to the actions of our brain - this is reasonable. But it would be a sad mistake to make our activity similar to the functioning of a computer. However, the identification of general laws of the design and construction process, the selection of common stages and procedures, the development of various methods of solving tasks at these stages is the matter necessary both for stimulating the creative activity of experienced designers and for training young specialists.

The design process is multi-stage and iterative, which involves returning and revising previously made decisions. It is worth adding the capabilities of a modern base of elements to this process, the distribution of which is completed by obtaining typical functions corresponding to individual microcircuits or elements of functional libraries programmed by VIS/HVIS.

Standard VIS/HVIS differ in the level of integration, as the high design cost reaches hundreds of millions of dollars, which is acceptable in this case because it is spread over a large number of manufactured chips.

In addition to standard parts the system also includes special parts specific to this solution. This applies to block control schemes, ensuring their interaction, etc. The implementation of a non-standard part of the system is historically associated with the use of small and medium-sized integrated circuits. The use of MIC and CIC is accompanied by a rapid increase in the number of integrated circuit cases, installation complications and a decrease in the reliability and speed of the system.

In addition, there is a long period of time between the completion of the circuit design and the delivery of the first prototypes, and any error or change in the circuit is very expensive.

The resulting contradiction found a solution in large - scale integrated circuits and ultra-large scale integrated circuits with a programmable and reprogrammable structure. The first representatives of this direction were programmable logic matrices, programmable logic matrix and basic matrix crystals. The user organizes his own logic in them, even in a single sample, using simple and cheap hardware compared to programmers.

Programmable logic arrays are based on technologies similar to those used in programmable memory devices, but these technologies are used for somewhat unusual purposes.

A programmable matrix is a crystal containing a large number of elements that perform basic logic functions (mainly gates and triggers). The user can organize these functions in one or another logic scheme according to his requirements and arbitrarily "split" not only the connections between the logic elements of the "AND/OR" buffer, but also determine the purpose of the crystal terminals and the chip body.

Currently, there are many versions of powerful software designed for compatible PCs. Using them the actual programming of the system is carried out according to the user's description of logic functions and equations using computer simulation, during which the correctness of the prepared project is checked.

At present you can write something like a "program", and then create a specialized integrated circuit based on it. This approach provides a number of advantages: increasing the size and simplicity of printed circuit boards, accelerating the development of new designs, increasing reliability, reducing the number of required standard types of digital microcircuits, and improving personal skills.

CHAPTER I. GENERAL CONCEPTS REGARDING DESIGN

1.1. Basic concepts and definitions

The design of new types and models of machines, devices, apparatus, instruments and other products is a complex and long-term process, which includes the development of initial data, drawings, technical documentation necessary for the manufacture of prototypes and subsequent production and operation of design objects.

Designing is a set of works aimed at obtaining the specification of a new or modernized technical object, sufficient for the realization or production of the object under the given conditions.

In the design process, there is a need to create a description necessary for the construction of an object that does not yet exist. Descriptions obtained during the designing are definitive or indirect. Final descriptions are a collection of design and technological documentation in the form of drawings, specifications, programs for computers and automation systems, etc.

A designing process completely controlled by a human is called non-automated.

A designing where a person interacts with a computer is called computer-aided design.

A computer-aided design is an organizational and technical system consisting of a complex of design automation tools that interacts with units of the design organization and performs automated design.

Design object (OD) is a new (modernized) technical object that did not exist in nature (in this form) before the beginning of the design activity, created in the process and obtained as a result of design.

The life cycle usually means the stages of the "existence" of a technical object (system) from the moment of realizing the need for its production to the moment of its destruction as unnecessary.

The short list typically includes design, production,performing of technical specifications,operation (including repairability and modification) and finally recycling.

The complexity of technical objects from the point of view of their design will increase due to the need to take into account the specifics of all other stages of the life cycle already at the design stage. This component complicates both the design process and the object itself. The degree of difficulty is usually proportional to the initial difficulty of this object. The refusal to take into account in the design process those characteristics of OD that are relevant at other stages of the life cycle can (and usually does) lead to significant additional material costs.

As for technical objects (systems), the complexity is usually estimated by the number of elements that make up the system and by the number of connections that determine the way the elements interact. As an alternative estimate, the power of a set of possible (observable) states of the system and a set of dependencies that determine possible transitions from state to state are often used (usually the first estimate can be reduced to the second).

There are two components of project complexity: **subjective and objective**.

The first component determines the degree of human perception of system behavior. A human being has long learned to deal with this element simply abstracting from it.

The second one is related to the dimension of the description (representation) of the system, and in particular to the dimension of design tasks. This component can be represented by the complexity estimates given earlier and is less relevant for design automation because human and computer measures of complexity differ.

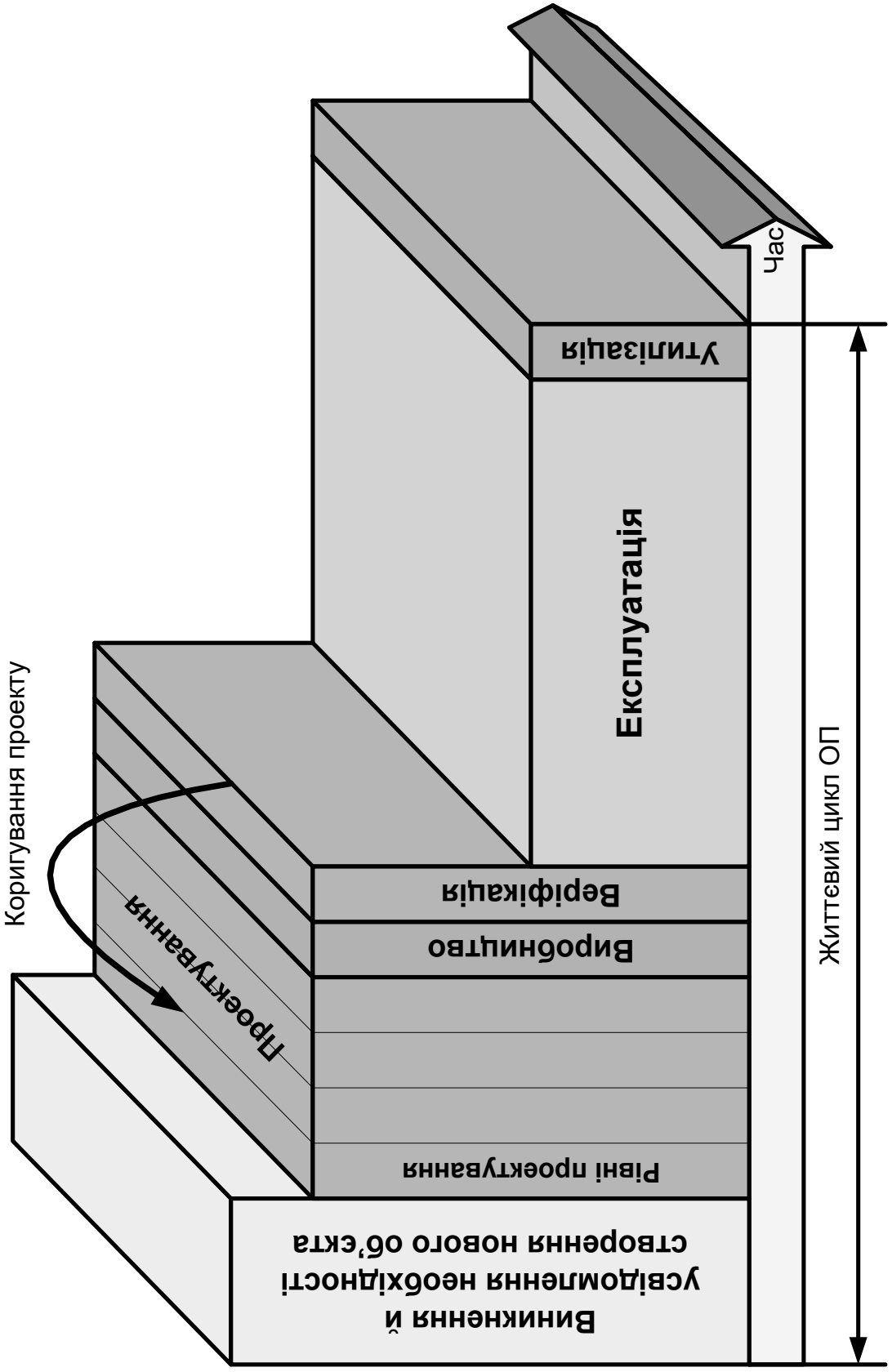


Figure 1.1 - Life cycle of a design object

The method of overcoming the "complexity complex" for the second component has also been known to mankind since ancient times and is based on the principle of "divide and conquer". The principle involves the division (decomposition) of the system into a number of less complex, interconnected subsystems, each of which has a smaller dimension and can be understood by a person if the abstractions corresponding to these subsystems are formulated.

In general, the influence of the "complexity complex" on the task of designing machines and computer systems is caused by:

- the complexity of the subject area itself;

- the need to take into account the features of other stages of the OD life cycle at the design stage;

- complexity of managing the design process;

- the complexity of describing the behavior of discrete systems;

- the necessity and complexity of verifying project solutions.

If the system is not discrete, that is, described by a continuous function, then small changes in the input parameters will always cause small changes in the output parameters. On the contrary, discrete systems by their nature have a finite number of possible states, although in large systems this number is very large according to the rules of combinatorics.

At the same time transitions between discrete states cannot be modeled using continuous functions. Any event external to the system can transfer it to a new state, and the transition from one state to another is not always deterministic. In adverse conditions an external event can disrupt the current state of the system, because its creators could not foresee all possible options. This is another (perhaps the main) reason why when decomposing discrete systems, even within the same level of abstraction, subsystems should have little dependence on each other, while ensuring

the integrity of the functions of the entire system. In discontinuous systems this behavior is unlikely, but in discrete systems any external event can affect any part of the system's internal state. This, of course, is the main reason for mandatory testing of discrete systems. But in reality, except for the most trivial cases, it is impossible to fully test these systems.

In the absence of both mathematical tools and intellectual capabilities to fully verify the behavior of large discrete systems, one can count only on an acceptable level of confidence in their correctness. This trust first of allmust be based on trust in project decisions, that is, it must be provided already in the design process.

Conventional tools testing comes down to testing individual programs on selective examples, this is not a way of demonstrating the correctness of programs in general, but rather an indicator of the possible absence of errors.

1.2. Classification of the design process

Ideas of complex technical objects in the process of their design are divided into aspects and hierarchical levels. Aspects characterize one or another group of related properties of an object. Typical aspects in descriptions of technical objects are: functional, constructive and technological.

The functional aspect reflects the physical and informational processes occurring in theobject during its operation.

The constructive aspect characterizes the structure, location in space and the shape of the component parts of the object.

The technological aspect determines the expediency, possibilities and methods of manufacturing the object under the given conditions.

The block-hierarchical approach to design consists in dividing the descriptions of the designed objects into hierarchical levels according to the degree of detailing the properties of objects.

Typical hierarchical levels of functional design are: functional-logical (functional and logical diagrams); diagrams (connection diagrams of nodes and individual blocks); component (design of elements and their location).

Design is also divided into stages,phases and procedures. There are stages:

- scientific research projects (SRP),
- research and development works (RD),
- prelimenarydesign,
- technical project,
- working project,
- test sample.

A design decision is a description of the object or its part sufficient to accept an application for the completion of the project or ways of its continuation.

A design procedure is a part of the designing that ends with obtaining a design decision.

A design path is a series of design procedures that lead to obtaining the necessary design solutions.

Analysis and synthesis procedures.

An analytical procedure is an object-oriented research.

The real task of analysis is formulated as the task of establishing correspondence between two different descriptions of the same object.

A synthesis procedure consists of creating descriptions of the designed object, which reflect the structure and parameters of the object.

The design of both individual objects and systems begins with the development of technical specifications (TS) for the design. The TS contains basic information about the project object, its operating conditions, as well as the customer's requirements for the designed product. A typical design scheme is shown in fig. 1.2.

The most important requirement for TS is its completeness.

Fulfillment of this requirement means timeliness and quality of project implementation. The next step is a preliminary design.

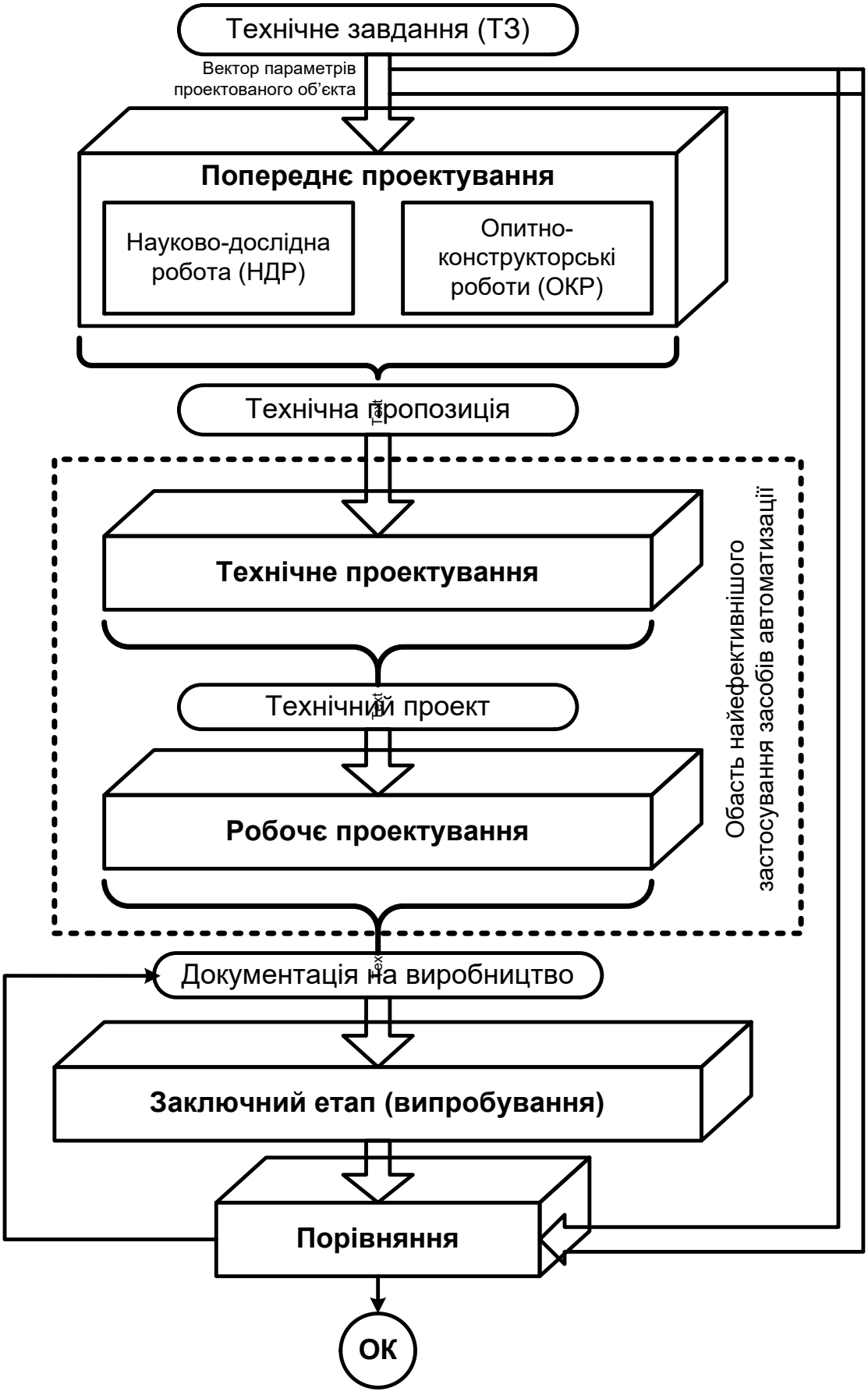


Figure 1.2 Typical design scheme

The initial project is a search for the main possibilities of building a system, studying new rules, structures and justifying the most general solutions.

The result of this stage is a technical proposal.

At the stage ofprelimenarydesign, a detailed study of the feasibility of building the system is carried out.

The design of the project is the result of this stage.

Technical design is the implementation of the presentation of all design and technological solutions which are in aggregate.

The result of this stage is a technical project.

Detailed design is the stage of detailed processing of all blocks, nodes and parts of the designed system, as well as technological processes of manufacturing parts and assembling them into nodes and blocks.

The last stage is the implementation of the prototype, the necessary changes based on the tests results are made to the design documentation.

The most time-consuming are the stages of technical and operational design in non-automated design. The introduction of automation at these stages leads to the most effective results.

In the process of designing a complex system certain ideas about the system, important properties with varying degrees of detail that reflect this system are formed. It is possible to distinguish constituent parts - levels of design in these representations.

One level usually includes representations that have a common physical basis and allow the same mathematical apparatus to be used to describe them. Design levels can be distinguished by the degree of detail with which the properties of the designed object are displayed. Then they are called horizontal (hierarchical) levels of design.

The block-hierarchical approach to design is divided into horizontal levels. Horizontal levels are characterized by the following features:

- when moving from a certain level K_1,where the system S is considered, to the adjacent, lower level K_2, the system S is divided into blocks and its individual blocks are considered instead of system S;
- consideration of each of the blocks at the K_2 level with a greater degree of detail than at the K_1 level leads to the tasks of approximately the same complexity from the point of view of human perception and the possibility of solving using available design tools;
- applying its concepts of system and element at each level of the hierarchy, that is, if the elements of the designed system S were considered to be blocks Sk , then at the adjacent, lower level K_2 the same blocks Sk are already considered the systems.

Design levels can also be distinguished from properties of objects that are taken into account by nature. In this case they are called vertical design levels.

When designing automation devices, the main vertical levels are functional (schematic), constructive and technological projects.

When designing automated complexes, algorithmic design (software) is added to these levels.

Functional design involves the development of structural, functional and principle schemes. The functional project determines the main features of the structure, principles of operation, the most important parameters and features of the created objects.

Algorithmic design is associated with the development of algorithms for the functioning of electronic computers and computer systems (CS), with the creation of their common system and application software.

Constructive design covers the implementation of the results of functional design, i.e. the choice of forms and materials of initial parts, the choice of standard sizes of unified parts, the spatial arrangement of components, which is ensured by taking into account the interaction between design elements.

The technological project covers the issue of implementation of the results of constructive design, that is, the issues of creating technological processes for the production are considered.

At the scientific research stage it is recommended to use special automation systems for conducting scientific research and experiments. Many elements of mathematics and CAD are used by these systems to support other stages of design.

A distinction is made between "top-down" and "bottom-up" design depending on the sequence of implementation of the design stages.

Bottom-up design (designing from the bottom up) is characterized by solving the tasks of the lower levels of the hierarchy before solving the tasks of higher levels.

Top-down design (top-down design) is the reverse sequence.

Currently, the design of complex devices, their nodes is carried out at various enterprises using various CAD programs, including typical ones, for example, CAD for the design of electronic and computing equipment, CAD for the design of electrical machines.

Functional design in CAD includes two large horizontal levels - system and functional-logical (fig. 1.3). Top-down design is typically used for tasks at these levels.

Structural diagrams of devices are designedat the system level. At this level the entire aggregation system is considered as a single whole, and the elements of

the system are such devices as processors, communication channels, various sensors, diligent devices, etc.

Functional and schematic diagrams of devices are developed at the functional and logical levels. There are sublevels here - register and logical:

- Block devices are developed at the register sublevel.
- At the logical level devices or their component blocks consist of separate logical elements.

At the schematic level basic electrical diagrams of devices are designed. The components here are elements of electronic circuits.

Individual components of the device are developed at the component level, they are considered as systems consisting of elements.

Functional design in CAD can be both top-down and bottom-up.

Higher, hierarchical levels of algorithmic design are used to create computer software. Two hierarchical levels are usually distinguished for complex software systems. At the highest of them, the software system is planned and algorithm diagrams are developed; program modules are elements of schemes. At the next level these modules are programmed in some algorithmic language. Top-down design is used here.

Architectural design is the choice of system architecture, that is, the definition of such structural and algorithmic features as data and command formats, the command system, rules of operation, the conditions for the occurrence and discipline of interrupt processing, etc.

Microprogramming is designed to develop microprograms for operations and procedures performed on a computer using hardware. This level is closely related to the functional and logical level of design.

The main tasks of the system and architectural levels of design are:

- determination of principles of system organization;
- choice of architecture, specification of system functions and their division into hardware and software functions;
- development of a design scheme, i.e. determination of the composition of devices and methods of their interaction;
- determination of requirements for the output parameters of the devices and compilation of technical specifications for the development of individual components of the system.

Technical conditions for the development of individual CAD devices include:

- a list of functions performed by the device;
- operating conditions of the device,
- requirements for its output parameters, data about the content and form of information that this device exchanges with other devices of the system.

In addition, at the stage of functional design of devices, the decision made at the stage of preliminary design regarding the nature of the base of the element is already known.

The following tasks are solved at the logical sub-level of the functional-logical design level:

- synthesis of functional and principle schemes of selected blocks;
- testing the operability of synthesized blocks taking into account delays and signal limitations of the selected base of elements or developing requirements for elements in the CAD system;
- synthesis of control and diagnostic tests;
- TC formulation for the circuit design level.

The main part of technical specifications at the level of circuit design is the requirements for the output parameters of electronic circuits: signal propagation delays, power dissipation, output voltage levels, resistance margin, etc. In addition,

the technical specification defines the operating conditions in the form of an indication of the permissible ranges of changes in external parameters (temperature, power supply voltage, etc.).

At the circuit design level the main design tasks are as follows:

- synthesis of the structure of the principle scheme;
- calculation of parameters of passive elements and determination of requirements for parameters of active elements;
- calculation of the probability of meeting thetechnical specifications requirements based on the output parameters;
- Formulation of technical specifications for the design of components.

The tasks of functional, structural and technological design are closely relatedat the component level. They are:

- selection of physical structure and calculation of parameters of semiconductor elements;
- selection of element topology and calculation of geometric dimensions;
- calculation of electrical parameters and characteristics of components;
- calculation of parameters of technological processes that provide the desired final result;
- calculation of the probability of meeting the requirements for the output parameters of elements and devices.

In top-down design the combination of hierarchical levels is manifested in the creation of technical specifications for the development of elements taking into account the requirements of the system.

In bottom-up design element development precedes system design, sotechnical specifications for elements are usually created based on expert opinion at the same level at which these elements are being developed. The connection between the levels is manifested primarily in the fact that when designing the

system, the properties of already designed elements are taken into account by using macromodels of elements.

Structural design involves hierarchical levels of design of racks, panels, typical replacement elements (TER). Top-down design is typical for solving project tasks.

Structural design includes solving tasks from groups: design of switchgear and installations; ensuring acceptable thermal conditions; design of electromechanical units of external devices; preparation of project documentation.

The main tasks of designing and assembling switches in CAD include placing the components on the substrate and tracking the electrical connections between the components. These tasks are listed below:

- design calculations of geometric dimensions of elements (this task is sometimes considered a functional design task);
- determining the relative arrangement of components on a structural element;
- the arragements of components on the structural element, taking into account the geometry of the device, schematic and technological limitations;
- call tracking;
- preparation of drawings depicting the general appearance of the device and determining the main overall dimensions.

The tasks of placement of elements and traceability of electrical connections are also solved in the CAD software of RSAD electronic devices. Thus, at the level of typical replaceable elements it is necessary to place the housings of microcircuits and lay the printed wires in one or several layers of the printed circuit board. In addition, the tasks of designing connectors and installations include the task of arranging elements into blocks.

Production of project documentation includes automatic design of project results of the above mentioned tasks in the required form (for example, in the form

of drawings, diagrams, tables, etc.). So, to obtain photo originals of printed circuit boards and photo templates of integrated circuits (ICs), software-controlled equipment - coordinate diagrams and photo-set devices are used now.

The tasks that are solved at each stage of block-hierarchical design are divided into tasks of synthesis and analysis (fig. 1, 3). Synthesis tasks are related to obtaining design options, analytical tasks are related to their evaluation.

Synthesis is the creation of a description of an object that corresponds to specified functions and complies with given constraints.

The task of synthesis is performed in the selected class of elementary objects that form an object implementing a given class of functions.

Output data - a description of the functions performed by the designed object; a list of quality parameters and their limits.

The result is some kind of structure that implements a given class of functions.

The structure of the object is understood as a set

$S = \{C,H\}$,

where C is a set of elements included in the object structure, and H is a set of associations between them.

Equal structures are those structures that perform the same functions ($F_1 = F_2$) , consist of the same elements ($\{C_1\} = \{C_2\}$) and are connected by identical bonds ($\{H_1\} = \{H_2\}$).).

Equivalent structures are the structures where $F_1 = F_2$, but $C_1 <> C_2$ and/or $H_1 <> H_2$.

A synthesis problem can have formal methods of solution - such a problem is solved algorithmically, otherwise it is algorithmically unsolvable. Algorithmically - unsolvable problems are solved manually or by heuristic methods (a complete list).

There are structural and parametric synthesis.

The purpose of structural synthesis is the obtaining structural diagrams of the object, which contain information about the composition of elements and the way of their connection. **Optimization** is the determination of the best values of the structure and (or) parameters in a certain sense.

Optimization associated exclusively with parametric synthesis, that is, with the calculation of optimal parameter values for a given object structure is called parametric optimization. The task of choosing the optimal structure is to optimize the structure.

The tasks of design analysis are the tasks of researching the model of the designed object.

The analysis consists in determining the functional and parametric description of the system based on the given structural description.

The purpose of solving the analytical problem is to study the properties of descriptions F, S and P, obtained at a certain stage of the convergence from the project solution tree. The purpose of such a study is to assess the quality of the obtained decision option or check F-descriptions for compliance with the given one.

Unlike the problem of synthesis, the problem of analysis is always solved algorithmically. The statement is correct, since a solution to the synthesis problem has already been obtained and at least the corresponding descriptions of F and S are known.

The problem of analysis is solved by modeling.

The most common methods of analysis are one-dimensional (studying an object at a given point of the behavior trajectory) and multidimensional (studying the properties of the object in the vicinity of a given point of the behavior trajectory).

The result of the analysis is the determination of adequacy.

Adequacy is an indicator of the conformity of the model to the analyzed object.

Formalization of the design task is a prerequisite for solving it on a computer. (fig. 1.3)

Formalized tasks are primarily tasks that have always been considered routine and do not require significant creative efforts of engineers. The procedures are:

compilation of design documentation (DD) in conditions in which the content of the DD has already been fully established, but does not yet have a form accepted for saving and further use (for example: the form of drawings, diagrams, schemes, algorithms, connections). , tables);

procedures for making electrical connections on printed circuit boards or making photoforms in printing.

In addition to routine tasks, formal tasks include most tasks related to the analysis of designed objects. Their formalization is achieved by developing the theory and methods of automated design before any modeling.

The first group of tasks are completely formalized tasks, most often they are performed on a computer, without human intervention in the decision-making process.

Partially formalized tasks

At the same time, there are many design tasks of creative nature for which formalization methods are unknown. These tasks are related to the selection of rules for the construction and organization of an object, the synthesis of schemes and structures in conditions where the selection of options is carried out from an

unlimited number of options, and the possibility of obtaining new, previously unknown solutions.

The second group of tasks are partially formalized tasks that are performed on a computer with the active participation of a person, that is, the work with a computer takes place in an interactive mode.

Informal problems, which make up the third group of problems are solved by an engineer without using a computer.

Currently, the development of synthesis methods and algorithms at different levels of hierarchical design is one of the directions of development of mathematical support for automatic design.

Modeling - in this context it defines the process of measuring on models the properties necessary to assess the quality of a design solution at a specific design level.

Models can be physical (various models, racks) and mathematical.

A mathematical model is a collection of mathematical objects (numbers, variables, vectors, sets, etc.) and connections between them.

The model reflects some properties of nature, but it is always different from nature. The model is the purest knowledge of the subject.

The following types of models are used in CAD:

- **Simulation model**. Applying some rules, the work of the OP is simulated;
- **Algorithmic model -** simulation model described with the help of algorithms and alphabet (set of rules and synthesis);
- **Analytical model** - a simulation model represented by a system of equations describing a given function of the object's functioning;
- **Functional model** - describes the functions performed by the system as a whole or the functions of individual elements and represents the physical or

informational processes occurring in the modeled object (usually these are systems of equations);

- **The structural model** reflects the structure, elements and connections between them and shows only the structural (in some cases geometric) properties of objects (usually graphs, matrices, etc.).

Let the project object (OP) be characterized by the expression

$$OP = \{F, S, P\},$$

where F, S and P are respectively functional, structural and parametric descriptions of the object.

The functional description displays the trajectory of the OP in the space of time states as a certain function, the arguments of which are control and passive actions of the external environment.Control actions can be both external and internal, that is, formed according to certain rules of operation hidden inside the object.

As a part of the formal definition, the project task is related to the decomposition of the original F-descriptions into certain subfunctions (components): $Fo = S(Fj1)$, $j = 1,2, ., n$, where S is the operator for determining such a composition of Fi_1, which provides a preliminary functional description (F_0). The operator S is called a structural description (S description) and defines the OP structure at this level of detail. The known objects called elements can correspond to some functional descriptions of the components (Fi) obtained by decomposition. An element can be a rather complex technical system. In this case it is important that during designing the F-description of the element does not need further decomposition and therefore it does not have S-descriptions.

Decomposition is again needed for the remaining Fi, and so on, until all descriptions of F match the elements.

The choice of decomposition option is usually determined by the quality of the obtained solution.

Let the quality be a set of OP properties called parameters P={ $p\ i$ }, i =1,2, ., k . At the same time, *pi* are calculated functions whose argument values are determined by the parametric descriptions of the terms of the next level ($P\ j$ +1), since the other levels have not yet been determined. Underlining the word "calculated" in italics means that the parameter value must be a scalar value. Otherwise, parameter values cannot be compared, that is, relational operations can be applied to them.

Abstraction is one of the most powerful intellectual ways of understanding complex phenomena available to humans.

The paradigm (starting point) of abstraction consists in determining the similarities of certain objects, situations or processes and deciding to emphasize the essential features, temporarily ignoring the differences.

Thus, the following stages can be distinguished during abstraction:

- Abstraction isthe selection of significant properties for consideration;
- Specification is the definition of the symbolic representation of abstract concepts:
- Transformation is the determination of a finite set of operations on a symbolic representation in order to predict the behavior of real objects in accordance with the abstraction;
- Axiomatization is a strict formulation of properties of objects and rules of transformation.

The model must be adapted or reassembled, and the parameters optimizedfor each new version of the design.

The design process is repeated. Iterations can span more than one project level. Thus, it is necessary to carry out the object analysis procedure many timesin the design process. Therefore, there is an obvious desire to reduce the complexity of each analysis option without compromising the quality of the final design. Under these conditions, it is advisable to use the simplest and most economical models

already at the initial stages of the design process, when high accuracy of the results is not required. At the final stage the most accurate models are used, a multifactorial analysis is carried out due to which reliable estimates of the object's operation are obtained.

The object synthesis procedure is a set of procedures for structure synthesis, model assembly, and parameter optimization.

The essence of this method is to use a cycle, which consists in sequential alternation of stages of generation of the next option and analyzing its quality (P-descriptions) as it approaches the desired result. Such an analysis allows you to determine the "direction" in which S-descriptions with a greater degree of agreement with the desired result can be obtained, and the opposite direction can be excluded from the consideration.

Procedures (algorithms) for solving the synthesis problem by heuristic methods, including the method of successive approximations are usually called iterative. Iterative algorithms sequentially create solution options and can be interrupted at any stage of the iteration if as a result of the quality analysis of the resulting option, it is determined that the option is acceptable. However, a constructive algorithm based on the method of successive approximations can be built if the signs of "cutting off" unpromising options are strong enough to give a single solution.

The optimal ones are those that correspond to the TS and are the best to achieve.

The task of CAD optimization comes down to transforming the physical representation of an object, its purpose and level of utility into a mathematical formulation of an extreme problem. The goal of optimization is expressed in optimization criteria.

Criteria are rules of superiority of compared options. The basis of the optimization criteria is the objective function F(x), where x is a set of controlled parameters. Vectors *x* with fixed values define one of the object variants and its characteristics.

The objective function should be such thatits valuescan be used to determine the degree of achievement of the goal. That is, the best option should be characterized by a large value of F(X), then the optimization consists in maximizing F (X) or vice versa, with the minimization of F(X) the best option should have smaller parameter values.

In addition to the objective function F(X) and the list of controlled parameters (X), the formulation of the optimization problem can take into account restrictions of the type of equality H(X) = 0 and inequality H(X)<> 0 . The direct restrictions *a i<= x i <= b i* , where *a i and b i* are the maximum possible values of *x i* , are a partial case of inequality restrictions.

The object is called strictly optimal if the values of all parameters are within the permissible range of parameter values.

The object is called quasi-optimal if some parameters of the vector (X) go beyond the limits, but the limits must be strictly defined.

In the presence of restrictions, the optimization task is called conditional optimization, otherwise it is called unconditional optimization.

The area where both direct restrictions and operability conditions are satisfied is called the operability region.

Thus, the final formulation of the design optimization problem is as follows: the extremization of the objective function F(X) in the domain XD determined by therestrictions H(X) = 0 and Φ(X)> 0.

The optimization problem in such a setup is a problem of mathematical programming. If the functions F(X), H(X), F(X) are linear, this is a linear programming problem. If at least one of them is nonlinear, we have a nonlinear programming problem.

The task of structure optimization comes down to the construction of the optimal structure S = (E, H).In this case, the optimalis understoodas a design option, the parameters of which correspond toall system, construction, technological, electrical and economic requirements of the TS.In this case, the criterion of optimality, which describes the quality of the designed structure acquires exceptional importance.

When designing according to private criteria, the objective function F(X) is considered the most important initial parameter of the designed object, all other parameters in the form of relevant operating conditions refer to restrictions. In this case, the optimal design problem is a one-criteria problem of mathematical programming: to extremize the value of the objective function F(X) if there is a system of restrictions on the parameters of the design object. The complexity of such a task is small.

Private criteria are chosen when there is a need to compare several equivalent solutions or there is a predetermined need to optimize one or more private criteria (without significant restrictions on other criteria).

If the characteristics of all necessary criteria can be specified in the function, then such a criterion is called generalized. The additive, multiplicative and minimax criteria are most often used as generalized criteria.

If the criterion does not take into account the probability dispersionof the parameteres, it is deterministic, otherwise it is statistical.

The objective function is created by summing the normalized values of the private criteriain additive criteria. Normalized criteria are the ratio of the actual value

of a private criterion to some normalizing value measured in the same units as the criterion itself (the result is a dimensionless value).

Several approaches to choosing a normalizing divisor are possible.

The first approach assumes that normative divisors are the maximum values of criteria achieved within existing design solutions.

The second approach involves taking the optimal value specified in the technical conditions as normalizing dividers.

The third approach suggests using the difference between the maximum and minimum value of the criterion in the compromise area as normalizing divisors.

Objective function: $F(x) = \Pi Fi(x)$

Disadvantages: formal perception does not follow from the objective role of a private criterion; mutual compensation of private criteria is possible.

A multiplicative criterion. When solving a problem, it is importantsometimes to take into account not the absolute value of the criterion, but its change.

Objective function: $F(x) = \Pi F_i(x)$

In case of inequality of the private criteria enter the significant factor C*i* and then the multiplicative criterion will take the form:

$$F(x)=\prod_{i=1}^{n} F_i^{C_i n}(x) \text{ Або } F(x)=\prod C_i F_i(x)$$

The advantage of the multiplicative criterion is that its use does not require the normalization of private criteria.

The disadvantage is that the criterion can compensate for excessive changes in some criteria as a result of changes in others.

Formally, the maximin principle is formulated as follows:

It is necessary to choose such a set $X0$ e X on which the maximum of the minimum values of the private criteria $F(x_0) = \max \min \{ c_i(x) \}$ is realized.

If the private criteria fi and (x) must be minimized, then the minimax rule applies

$F(x0) = min.\ max\ \{fi(x)\}$.

Additive criteria are selected when the absolute numerical values of the criteria with the selected vector X *have a significant value*.

If vectors are affected by changes in the absolute values of the private criteria X, then it is advisable to use the multiplicative criterion.

If the task is to achieve equality of normalized values of conflicting private criteria, then the optimization should be carried out according to the maximum criterion (minimax).

The evaluation method. Experts assign points in the range {1,10} to each parameter. Fractional values and the same estimates are also possible.

Then $H_{i,k} = h_{i,k} / ((i = 1, n)\ h_{i,k}$;

$c_i = (((k=1, L)\ H_{i,k}) / ((i=1, n)\ ((k=1, L)\ H_{i,k})$

The ranking method.Each of I experts ranks n criteria (in order of decreasing importance). Each parameter is assigned a rank of $n - 1$ based on this assessment. This value is called the transformed rank of the i-th criterion, then

$c_i = ((k=1, L)(r_{i,k} / ((i=1, n)((k=1, L)\ r_{i,k})$.

The properties of systems, system components and the external environment in which the object must function are distinguished among the properties of the object displayed in the descriptions at a certain level of the hierarchy.

Parameters are a quantitative expression of values that reflect the properties of systems as a whole, system components and the external environment in which the object must function.

The parameters are respectively outgoing, internal and external.

Let us denote the number of initial parameters - internal and external by *m, n, t,* and the vectors of these parameters by $Y = (y\,1, y\,2, .., y\,m)$, $X = (x_1, x_2, .., x\,n)$, $Q = (q_1, q_2, .., q\,t)$. It is obvious that the properties of the system depend on internal and external parameters, that is, there is a functional dependence

The system of relations $F = (y, x, t)$ is an example of a mathematical model (MM) of an object. The presence of such a MM facilitates the estimation of the initial parameters based on the known values of the vectors *Y and X*. However, the existence of a relationship does not mean that it is known to the programmer and can be accurately represented in such an unambiguous form in relation to the vectors *Y* and *X.* As a rule, to obtain a mathematical model is possible only for very simple objects.

The following features of the model parameters of the designed objects should be noted.

The internal parameters (elements parameters) in the models of the *k-th* level of the hierarchy become the output parameters in the models of the lower $(k+1)$-th level of the hierarchy. Thus, in the case of an electronic amplifier, the parameters of the transistor are internal during the design of the amplifier and simultaneously derived during the design of the transistor itself.

Initial parameters, that is, phase variables that appear in the model of one of the subsystems (in one aspect of the description), often turn out to be external parameters in the descriptions of other subsystems (other aspects). Thus, the maximum temperatures of the housings of electronic devices in the electrical models of the

amplifier refer to the external parameters, and in the thermal models of the same object - to the initial parameters.

Preliminary descriptions of designed objects are often KT for designing. These descriptions contain values called technical requirements and output parameters (output parameters standards). Technical requirements form a vector $TT = (TT_1, TT_2, ..., TT_n)$, where the values of TT are the limits of the ranges of changes in the initial parameters.

1.4. Automated design system

Automated design system (CAD) is a set of tools and methods of automated design.

CAD consists of several component parts, which are called consumables (Fig. 1.4).

CAD technical support is a set of interconnected and compatible technical tools designed to perform automated design tasks.

Technical support is divided into groups of activities:

- data processing software, which is represented by processors and memory devices, that is, computer devices in which data conversion and software control of calculations are carried out;
- data preparation and input;
- display and documentation, they are used to communicate the computer with the external environment and with the operator;
- archiving of project solutions presented by external media, databases;
- data transfer is used to organize communication between geographically dispersed computers and terminals (border points).

CAD software combines the programs of data processing systems on machine media and the software documentation necessary for the operation of the program.

Software is divided into general system, basic and applied (special). System software is intended for organizing the functioning of technical means, that is, for planning and managing the computing process, distribution of available resources and is represented by a computer and computer operating systems. System software is usually developed for many applications and does not reflect the specifics of CAD. Basic and application software is created for CAD purposes. Core software includes programs that enable the application programs to function correctly.

Application software implements mathematical support for the direct execution of project procedures. Application software typically takes the form of application program packages (APPs), each serving a specific step in the design process or a group of tasks of the same type within different steps.

CAD information support combines all types of data necessary for automated design.

These data can be presented in the form of certain documents on various media, containing background information on materials, component products, typical design solutions, parameters of elements, a summary of the current state of development in the form of intermediate and final design solutions, designs and parameters of the projected objects, etc. The main part of CAD information support is a data bank, which is a set of means of centralized collection and collective use of data in CAD. The data bank (BND) consists of a database and a database management system.

A database (DB) is the data itself which is stored in the computer's memory devices and organized according to the rules adopted in this database. A database management system (DBMS) is a set of software tools that ensure the functioning of a database. When using DBMS, data is stored in the database, their selection is

carried out at the request of users and application programs, data is protected against distortion, unauthorized access, etc.

CAD language support is represented by a set of languages used to describe automated design procedures and design solutions.

The main part of language support is the language of human-computer communication.

Methodological support for CAD consists of documents characterizing the composition, rules for choosing and functioning of computer-aided design tools.

A broader interpretation of the concept of methodological support is acceptable, in which methodological support is understood as a set of mathematical and linguistic support and named documents that implement the rules for using project resources.

Organizational support of CAD includes regulations, instructions, orders, staff schedules, qualification requirements and other documents regulating the organizational structure of the design organization's divisions and their interaction with the complex of computer-aided design tools.

The specialization of a certain part of CAD for the implementation of design tasks of one design stage leads to the seperation of this part as aCAD subsystem. Such specialization includes software, math, language support, and sometimes technical support. As a rule, automation devices are subsystems of functional and logical design and structural designin the CAD.

CAD systems include scheme design, component design and structural (topological) design.

The complexity of modern technical objects determines the emergence of subsystems in CAD that perform the functions of independent, automated design systems (CAD of electronic systems, CAD of electrical machines, etc.).

As a rule, each subsystem has its own input language and application package.

Modern CAD systems are created according to the following principles:

1) CAD- a man-machine system. The team of developers is an integral part of the design system, which performs design work interacting with computers.

2) Integrated automation of all design levels.

This makes it possible to implement such changes in the structure of project enterprises and forms of documentation that meet the goals of automation - reduction of material and time costs, improvement of design quality, maintenance of the number of engineering and technical workers at the current level despite the complexity of the designed objects.

3) Information coordination of subsystems and design programs is performed under the following conditions:

- programs are created to work with one database and do not require manual rearrangement of numerical tables that are entered for one and sent for another connected programs;
- assignment of initial information about the object or necessary design operations is performed in a single input language.

4) CAD accessibility.

The properties of accessibility of the system means the possibility of making changes to the system during its operation. Changes may consist of adding new or replacing old elements of software, information, and also technical and language support. Making changes should be as simple and accessible as possible for CAD users. The property of accessibility leads to an extension of the service life of the system and increases its versatility.

5) Compatibility of traditional and computer-aided design.

This principle is relevant in cases where computer-aided design is implemented at an already operating enterprise with an existing structure, departmental connections, forms and methods of using design documentation. It is in these conditions that the evolutionary path of CAD implementation is appropriate, in which changes dictated by the characteristics of automated design will not disrupt the normal functioning of the enterprise for a long time.

The following requirements are applied to CAD technical support:

- ease of use by design engineers, the possibility of prompt interaction between engineers and computers;
- sufficient performance and RAM capacity to solve the tasks at all design stages in an acceptable time;
- the possibility of simultaneous work with the technical means of the required number of users for the effective work of the entire team of programmers;
- openness of the complex of technical means to the expansion and modernization of the system in the course of improvement and development of technology;
- high reliability, reasonable cost,etc.

Fulfillment of the above-mentioned requirements is possible only under the condition of organizing technical support in the form of a PPO system that ensures operation in several modes. Such technical support is called a complex of technical CAD tools (TC complex).

The form of computer use, in which computing resources are concentrated in the data center and run exclusively in batch mode is not suitable for modern CAD.

The computer then becomes an efficient, a regularly used design tool, when the engineer is able to quickly access the machine and get the results of a solution just as quickly.

Therefore, a group of external information input-output devices should be developed in the TC complex.In this case, the effective interaction of an engineer with a computer will be ensured only when the form of input and output of information is convenient for a person and does not lead to the need to manually perform cumbersome and erroneous coding or decoding operations.

Depending on the nature of the tasks to be solved, convenient forms of presenting information can be tables, pictures, diagrams, text messages, etc.

Thus, the first of the requirements for technical CAD tools specified at the beginning of the section provides for the inclusion both a standard set of external computing devices and additional devices for the rapid input and output of information, including graphic form, as part of the TS complex. This set of external devices is installed in the premises of the design department and is called the designer automated work station (WS).

WS is a complex of special technical devices that with the support of software, information and language tools ensure the solution of specific project tasks.

The composition of the WS depends on the nature of the tasks performed in the project unit.

On the upper level there are one or several high-performance computers. These computers form a central computing complex (CCC) designed to solve complex design tasks that require a large amount of computer time and memory. At a lower level are minicomputers (terminal computers) which are part of ARM. The mini-computer in the ARM manages the operation of a complex of external devices, the exchange of information between the ARM and the CEC; solves relatively simple design problems in terms of machine time and memory consumption.

CAD mathematical support (MS) is a set of mathematical methods (MMet), mathematical models (MM) and design algorithms (ALP) which are necessary to perform design tasks and are presented in a given form.

Mathematical supports can be divided into two types: invariant MS; special MS.

Special MS is included in CAD specifically for the design of technical objects on which this CAD is oriented.

Invariant MS includes methods and algorithms loosely related to the characteristics of mathematical models and applied at many hierarchical levels.

Requirements for mathematical support:

1) uniformity;

2) algorithmic reliability;

3) validity of mathematical models and problems;

4) perfection;

5) efficiency

The versatility of MS is its applicability in a wide class of designed objects.

One of the differences between CAD calculation methods and manual calculation methods is a high degree of versatility. For example, the CAD circuit design subsystem uses transistor mathematical models valid for any operating field (active, saturated, cutoff, inverse active), as well as methods for obtaining and analyzing models for any analog or switching circuit. elements from the allowed list are used; models and algorithms are used in the subsystem of CAD structural design, information processing which allows to study stationary and non-stationary processes, with arbitrary rights of work in BC and with arbitrary input flows.

In order for CAD to be applicable to any or most objects designed at the enterprise, a high degree of MS versatility is required.

Algorithmic reliability is a property of MS component that provides correct results when used.

Algorithms for which it is impossible to clearly define the limits of applicability are called heuristic. Such algorithms can be used incorrectly, which will either lead to an incorrect solution or no solution will be found. In CAD such algorithms and methods are used in cases where abnormal results are obvious.

Quantitative assessment of algorithmic reliability is the probability of obtaining the correct result while maintaining certain restrictions on themethod use. If this probability is equal to or close to unity, then the method is said to be algorithmically reliable.

A convention of mathematical models and problems is the property of increasing the errors of the output parameters with small errors of the input parameters. To analyze and optimize objects with poorly defined mathematical models, it is necessary to use special methods with increased algorithmic reliability.

Accuracy is the degree of convergence of calculated results with the real ones. Accuracy is assessed by solving special test tasks designed to highlight the contribution of each element to the total error.

In most cases the solution to project tasks is characterized by:

- combined use of most MS components makes it difficult to define the contribution to the total error of each component;
- the vector nature of the results (for example, in the analysis there is a vector of initial parameters, in optimization - the coordinates of the extreme point), that is, the result of the decision is not a single parameter, but a multi-parameter value.

Efficiency - consumption of machine time and memory.

Ways to reduce machine operating time:

1. Parallelism of the computing process;

2. Application of macro modeling.

Invariant mathematics support includes:

1. Optimization methods and algorithms;

2. Multivariate analysis;

3. Logical combinatorial decision-making methods.

Optimization methods and algorithms.

Optimization methods differ in the final set of quality indicators.

Multivariate analysis

The main methods of multivariate analysis are sensitivity analysis and statistical analysis.

The purpose of sensitivity analysis is to determine sensitivity factors (influence factors).

The coefficient of absolute sensitivity determines the influence of the *i-th* internal parameter on the *j-th* external parameter: $a\,j\,,\ i = dy\,yj\,/\,d\,xi$;

The coefficient of relative sensitivity of the output parameter $y\,j$ to changes in the internal parameter x has the form: $b\,j\,,\ i = a\,j\,,\ i \times x\,ja\ nom\,/\,y\,j\ nom$.

The purpose of the statistical analysis is to obtain estimates of the spread of the initial parameters and the probability of meeting the specified operating conditions of the designed facility.

The results of the statistical analysis are as follows:

- histogram of outgoing parameters;

- estimates of mathematical expectations and mean square deviations of each of yj ;

- maximum possible deviations;

- estimate the correlation coefficients between $y\ j$ and $x\ j$ and the initial parameters.

Information on the spread of internal parameters and permissible ranges of changes or laws of distribution of external parameters are used as initial data.

Language support covers all languages used in CAD. The structure of language support can be represented as follows (fig. 1.5)

Language support is divided into the following components:

- programming languages (universal, machine);

- design languages (input languages);

- control languages;

- instrumental languages.

Input languages are divided into:

- task description language;

- object description languages.

Task description languages are used to describe the project path. They aresimilar to operating systems, but focused on CAD.

Object description languages are used to describe the function of parameters and structural description of an object, input, intermediate and result data.

Input language requirements are:

- versatility;

- convenience of perceiving the alphabet and language syntax;

- maximum brevity of the description;

- unambiguous interpretation of language elements and constructions;

- possibility of development and expansion.

Object description languages are divided into:

- procedural languages;

- automatic languages.

Procedural languages:

- languages of functional description;

- languages of structural description.

The functional description defines the dynamic functions of the object (the function develops over time) and can be expressed analytically, graphically, tabularly.

Since the model of structural description is a graph (oriented, multigraph, etc.), graph forms of representation are most often used for their task. Adjacency and incidence matrices can be used, but for a structural description of real objects these matrices will be essentially empty, leading to undue cost or high complexity, so these matrices are rarely used and the vast majority of languages use a list data structures.

Instrumental languages are a special category of software tools, with the help of which all other programs are created.

The tools category includes not only high-level language translators, but also assemblers, loaders, debuggersand other system programs. The tools are used to create both application software and new system programming tools, including high-level languagestranslators.

Language requirements can be formulated as follows:

1) All language constructions should be naturally and simply (elegantly) defined.

2) To solve a specific task it should be possible to use a combination of structures to avoid the need to introduce a new structure.

3) There should be a minimum number of non-obvious special purpose structures.

4) It is enough for a user to know only those set of constructions which are directly used in his program.

Task control languages are conditional languages from a specific set of commands that can be specified on the command line, create settings or invoke actions.

Task control languages provide task management features such as:

- adaptation of program components to solve a specific project task;
- interactive decision-making management;

CAD database configuration for a given design task.

GUI - a visual representation of the task control language.

CAD software includes:

- system software which is usually used by the operating system;
- thematic software that supports the design process itself.

Design process support tools belong to the class of complex software systems that use databases.

Existing and iterative CAD systems typically suffer from a number of significant shortcomings, including a focus on the physical architecture of CAD systems in a particular subject area.

Historically, CAD programs consist of two levels:

preprocessing level (ARM with GUI at the base)

- the final processing level(certainbdatabase with which some applications work, and the database and applications are closely related).

This program organization is not very flexible and poorly supported. Such programs are extremely difficult to load into a distributed environment.

To mitigate these difficulties, a logical application architecture with the following layers instead of a physical layer isused:

- document level (desktop and graphic programs)
- design principles, design decision-making principles and design project management
- data management

Provided that the cross-layer interfaces are standardized, logical layers can be created independently. This means that each of these layers can be implemented as an independent component and distributed. In this sense they talk about database server, application server and desktop applications.

CAD information support is a set of information necessary for the implementation of design tasks presented in a certain form.

The main part of IS consists of automated data banks, which consist of CAD databases (DB) and database management systems (DBMS).

A database is a structured collection of related data from a specific domain for various purposes that represents the states of objects, their properties, and relationships.

The IS includes regulatory and reference documents, tasks of state plans, forecasts of technology development, typical project solutions, systems of

classification and coding of technical and economic information, systems of documentation such as ESKD, ESTD, files and data blocks on machine media, planning, fund forecasting, typical solutions, algorithms and programs, etc

Information support is determined by CAD databases and is characterized by the characteristics of these databases.In this case, the database can be understood as a set of the following elements: their conceptual model, tools that ensure the immersion of data in a certain storage built on the basis of a conceptual model and the function of extracting the necessary data (DBMS), a set of tools and methods ensuring the physical representation of data on certain carriers.

Requirements for the CAD database:

1) information should be the central concept, applications should be built around the database;

2) possible exclusions of redundancy (information must be stored in one place and changed only once, conflicting copies are excluded);

3) methods of making changes to data must take into account the relativity of time.

For example, data changes and processing differ between transactional and analytical databases.

Transactional databases support the design process. Analytical onessupport those on the basis of which reports are created.

A database is a structured collection of related data from a specific domain for various purposes that represents the states of objects, their properties, and relationships.

All features of an object are attributes of the object. The information contained in each attribute is called the value of that attribute.

Any object that is characterized by a record that contains the object's identifier and its attributes and the record key used to find that object.A record address, an identifier, an indexed record addresscan be a key.

A database management system (DBMS) is a special program designed to record, systematize, search for information in a database and provide it to a consumer.

A set of databases and DBMS is called a data bank.

There are two representations of the database:

- logical;
- physical

A logical representation is a representation of application developers using ready-made databases. It reflects the composition of information and the connections between its elements stored in the database, and does not take into account the issue of placing and storing information on physical media.

The physical representation of a database reflects how the information is displayed on the computer medium (how and where the information is stored and used).

Concepts of elements and connections between them are used. There are three types of communication:

- simple connection (usually one-way)
- complex connection (a lot of connections)
- conditional links (which can be broken).

Database requirements:

1) Adequacy of data, consistency and reliability.

2) The organization of the database should ensure that the time of data loading by programs is coordinated with the frequency of their use.

3) Versality, that is, the availability of all necessary data in the database and the possibility of accessing them in the process of solving tasks.

4) Openness of the database.

5) Availability of languages with a high level of user interaction with the database.

6) Confidentiality.

7) Security (protection against loss).

8) Optimal organization of data (minimum excess).

The database consists of at least three levels:

- fixed part;
- semi-permanent;
- variable.

The permanent part contains background information (fixed, previously obtained solutions). Semi-permanent - those data that are used to solve the problem. A variable is the data with which any program is currently working.

A hierarchical database consists of an ordered set of trees; more precisely, from an ordered set of multiple instances of the same type of tree.

A tree type consists of a single "root" entry type and an ordered set of zero or more subtree types (each of which is a type of tree). A type tree is typically a hierarchically organized set of record types.

A descendant type with a common instance of an ancestor type is called a twin. The complete traversal order is defined for the database - from top to bottom and from left to right.

The disadvantage: access to information is possible only through root access, and possible connection types are fixed in advance.

A conceptual data model is the basis for building a database. The earliest hierarchical databases were based on one-to-many relational data analysis.

The disadvantage of the database is that access to information is possible only through the root record, and the possible connection types are fixed in advance. Obtaining non-standard information requires a lot of work, which follows from the type of connection description.

A network database is also called a semantic or personnel database.

Semantic databases are built on the basis of many-to-many relationships and practically contain several hierarchical levels with repeated data and relationships between them.

These databases are not characterized by saving data in one place. In this case, there are difficulties with replication (datarolling back and saving in all places).

Relational databases are two-dimensional tables in which each row contains one data item.

Any column can be a key. Communication between tables occurs through any element of these tables.

A sequence of attributes associated with one element is called a tuple. Each column is called a domain.

Relational database tables have the following properties:

- each table element is one data element;
- there are no duplicate records;
- all columns in the table are the same (each column contains elements of the same type);

- columns are assigned unique names;
- absence of two identical lines in the tables;
- rows and columns can be seen in any order, regardless of the information, content and value they containin tableoperations.

The concept of connection is interpreted as a query in SQL.

These databases use a strict mathematical apparatus - "relational algebra", which allows you to model data and create queries of almost any form based on these models.

SECTION II. PRINCIPLES OF DESIGN OF RADIO- ELECTRONIC DEVICES

2.1. General principles

Designing is the development of technical documentation that allows you to implement a given device under given conditions.

The design strategy is a functional distribution, which includes the external description of the block (inputs and outputs) and the internal description - that is, the function, the work algorithm: $F = \Phi(X , t)$, where X is a vector of input values; F is a vector of initial values; time. During the decomposition the function Φ is divided into simpler functions $\Phi 1 ,..., \Phi k$, between which there must be certain connections corresponding to the accepted algorithm for implementing the function Φ. As a result of the decomposition the final structure is obtained. Transition from function to structure - synthesis.

The optimal option is selected based on the results of the analysis, during which the correct operation and some indicators characterizing the device are checked. Decomposition of block functions is carried out to obtain typical functions, each of them can be implemented by one or another microcircuit.

The design is a multi-stage and iterativeprocess, which involves returning and revising previously made decisions. It is worth adding the capabilities of a modern base of elements to this process, the distribution of which ends standard functions corresponding to individual microcircuits or elements of functional libraries programmed by VIS / HVIS are obtained.

The nature of the design largely depends on the type of element base used. The classification of integrated circuits according to features related to their design methods is shown in fig. 2.1.

Standard microcircuits include low- and medium-level integrated circuits (MIS and SIS). These microcircuits are mass produced and implement standard elements and nodes, the functioning of which is not determined by specific consumers. Microprocessors (MPs), microcontrollers (MCs) and memory devices (SDs) belong to standard highly integrated circuits (HIC/VHICs) that remain unchanged after production, regardless of the devices and systems in which they are used. Standard integrated circuits have a large market, which helps to reduce their cost.

Specialized integrated circuits (SIC) are those which design, unlike standard integrated circuits of mass production is adapted to the specific requirements of a particular project.A SIC includes semi-customized and customized classes. Types of non-standard microcircuits are completely non-standard and designed "according to standard schemes".

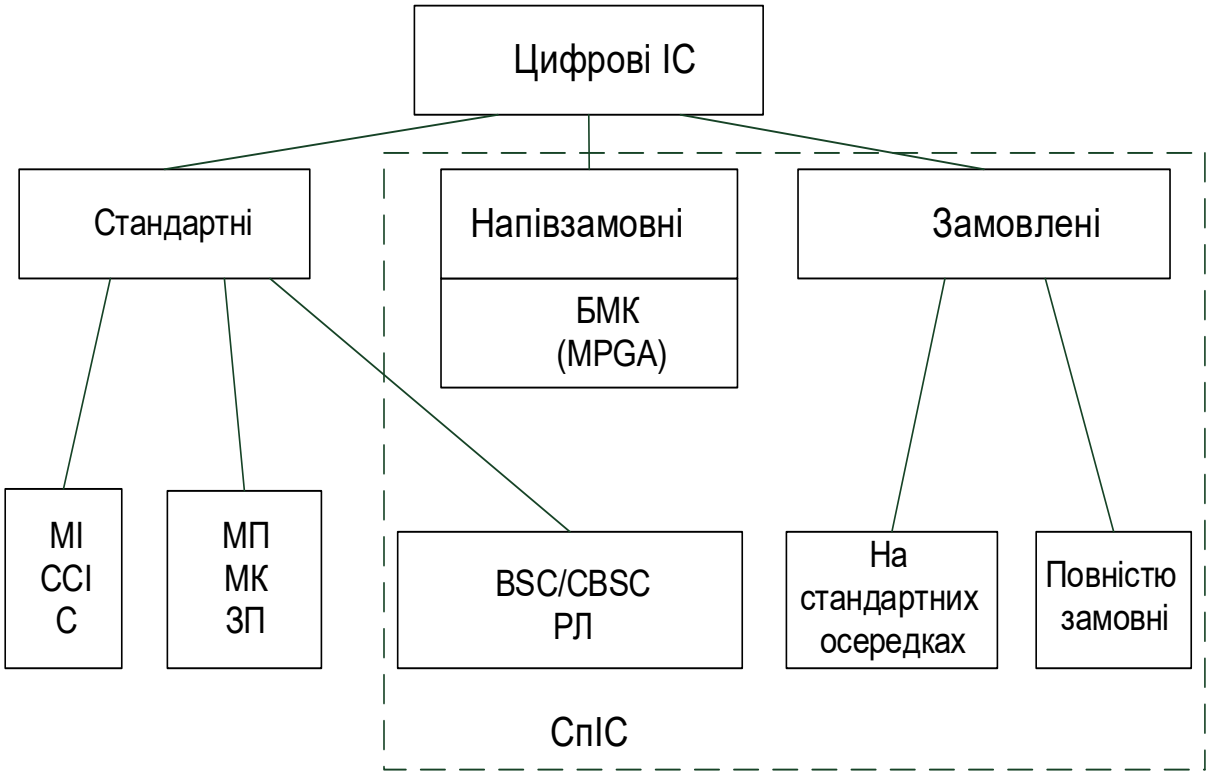


Рис.2.1. Класіфікація цифрових інтегральних схем

Fully personalized projects are designed according to the specific client's requirements. The developer has complete freedom of action, defining the scheme at his discretion, right down to the level of system components (individual transistors, etc.). Schematic production requires the development of a whole set of photo templates, checking and debugging of all individual fragments. Such schemes are very expensive and have long design cycles.

Circuits on standard crystals differ from fully custom ones in that their fragments come from a previously developed library of circuit designs. Fragments are already well developed, and the cost and duration of the project are reduced. The production of schemes also requires the production of a full set of photo templates, but their development is facilitated. Losses compared to the total number of IC orders consist in the fact that the designer has fewer opportunities in the construction

of the scheme, that is, the results of its optimization according to the criteria of crystal simplicity, speed, etc., are less effective.

The highest technical characteristics are achieved with completely non-standard circuits, but the method of standard crystals is popular, since it is possible to significantly simplify the design of the circuit with its helpwith a small loss oftechnicalcharacteristics. Fully custom circuits are designed approximately twice as fast as standard crystals. Semi-custom circuits include base matrix crystals (BMC). In this case we are dealing with a standard semi-finished product, which is brought to the finished product using separate connectors. The implementation required the production of only a small number of templates. The cost and duration of the project in comparison with completely non-standard schemes is reduced by 3..4 times, but the result is still far from optimal, since the surface of the crystal is less rationally used in the BMC (unused elements remain for the crystal, etc.), the connection lengths are not minimal , and the speed is not the maximum.

The similarity of design methods on BMC and standard crystals lies in the use of libraries of functional elements. The difference is that for circuits developed by the standard crystal method, the set of library elements has a more pronounced topological width. For example, only the height of the cells is standardized, and their length can be different. When designing, the necessary functional blocks are first selected from the collection of library elements, and then the tasks of their layout and tracking are considered.

CAD for the design of a standard cell is more complex than for the design of a BMC. Fully personalized projects are designed according to the specific client's requirements. The developer has complete freedom of action, defining the scheme at his discretion, right down to the level of system components (individual transistors, etc.). Schematic production requires the development of a whole set of photo templates, checking and debugging of all individual fragments. Such schemes are very expensive and have long design cycles, which have tighter topological constraints. Limitations were also introduced to the standard cell method (constant

cell height, determination of geometric dimensions and positions of power rails, timing, etc.), but as more powerful CAD systems are used, the limitations are reduced.

An important place in the classification is occupied by super-largeprogrammable logic integrated circuits. On the one hand, they belong to the CIS because they are ultimately adapted to the requirements of a specific project.However, this process (circuit configuration) does not affect the manufacturer for whom the circuit is a standard product.

According to experts expressed in Electronic Design magazine, "programming equipment will make the same revolution in the coming years as microcomputers did in the early 70s."

2.2. Areas of application of various types of HVIS

All types of ASIC have their areas of application. Each type is characterized by a certain ratio of such parameters as complexity (achievable level of integration), speed and cost. The main concept can be explained from the point of view of economics by referring to the formula for the SI value produced in an already mastered technological process:

$$C_{\text{IC}} = C_{\text{виг}} + \frac{C_{\text{пр}}}{N},$$

Where $C_{\text{виг}}$ is IC manufacturing cost (cost of crystal and other materials, cost of technological operations for IC manufacturing, control tests). Production costs apply to each IC, that is, they are repeated as many times as the IC is produced; Cpr is cost of IS design, i.e. one-time costs for this type of AI; N - volume of production (circulation), which is the number of integrated circuits that will be produced.

The cost of designing a VIS/HVIS is high and can reach hundreds of millions of dollars. In the case of expensive VIS/HVIS design options, production becomes profitable only with high sales volumes.

The costs of design *Cpr* is interrelated with the cost of manufacturing $C_{\text{виг}}$. An increase in design costs usually leads to a decrease of $C_{\text{виг}}$, because the more perfect the design, the more rationally the surface of the crystal and other resources are used.

The following provisions apply to programming logic. Simple devices with a complexity of hundreds of equivalent gates should be implemented on PLD (PAL, GAL, PLA). As design complexity increases, there is a natural transition to FPQA and CPLD if the IC effort is relatively small. The increase in circulation (about tens of thousands) leads to the advantages of implementations on BMC since the cost of manufacturing a small number of templates for creating connections will be distributed over a large number of microcircuits, and the cost of manufacturing each IC will be reduced due to the exclusion of software connections from the circuit and methods oftheir programming.

With an even larger circulation the method of standard cells turns out to be cost-effective, which allows you to further improve the parameters of the circuit, to place its elements more densely on the crystal, that is, to reduce and increase the speed of $C_{\text{виг}}$.In this case, the $\frac{C_{\text{пр}}}{N}$ term in the IS cost formula will not be too large due to the high value of N, although the need to design a whole set of templates for technological processes leads to high costs.

Fully structured design is not typical for the CIS. It is so expensive that it is practically only used to create a mass-produced VIS/HVIS standard. For example, the development of the first 32-bit microprocessor cost then $140 million, and 1 Mbit of RAM cost $395 million.

It is worth noting that the leading computer company IBM uses a mixed-type design methodology, placing a standard VIS/HVIS cell in the same areas. Critical signalprocessingcircuits usedenser and fastercircuits another direction is BMC transistors.

The design of standard, serial ICs as well as design by non-standard methods in general is the fate of large specialized companies. Circuit engineers are mainly engaged in other developments: simple digital devices based on MIS and SIS, microprocessor systems of technical means and technological processes, small-sized equipment or system prototypes based on programmable IS logic.

Designing based on MIS and CSI is the most traditional process that uses both heuristic approaches and formalized methods. The designer determines the design of the devicebased on his knowledge, ideas and the acquired experience of the predecessors, and when determining the functions of individual blocks, the designer also uses formal methods. In this case, quality design requires knowledge of typical functional nodes, their properties and parameters.

A microprocessor system is created as a result of the development of a complex of hardware and software tools. The development of the hardware part comes down to the assembly of the system from standard modules: the central unit, various types of memory, adapters, controllers and external devices.

The main part of the engineering development of equipment in the conditions of modern Ukraine is certainly the use of programmable logic to create the necessary devices or to debug them.

Designing based on very complex logical programming schemes is carried out exclusively with the help of automated design systems. The simplified structure of design algorithms is shown in fig. 2. 2.

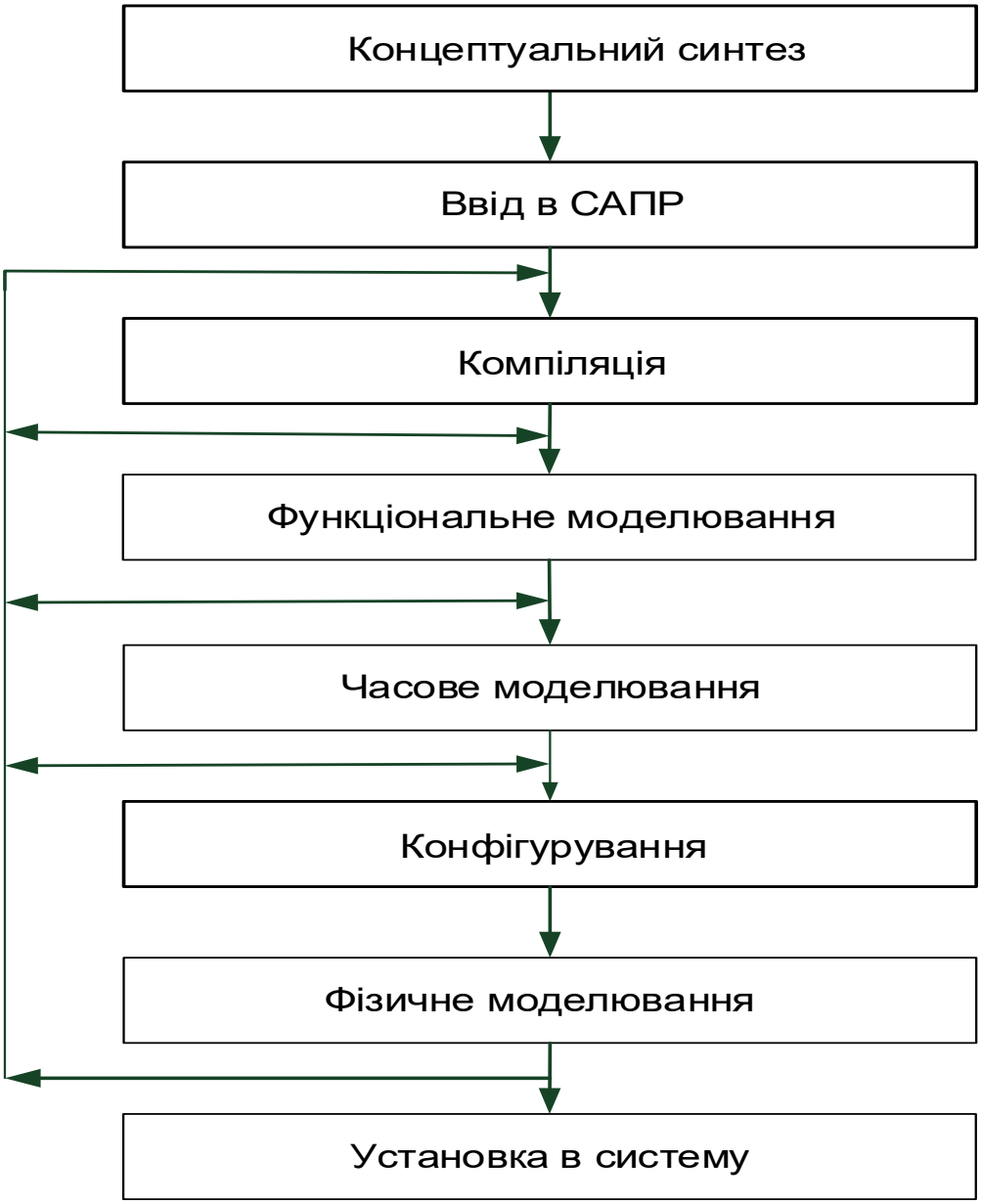


Fig. 2.2. The structure of the automatic design algorithm

Designing at the conceptual level is the work of the designer, which has little relation to automation. At this level the necessary functioning of the device, input and output signals, their nature and connections, division of the project into parts, etc. are determined. The results of the conceptual synthesis are entered into the CAD program, which compiles the project, that is, synthesizes the device in the database of its model library. The resulting design needs careful verification, so the synthesis stage is followed by an analysis stage, which is carried out through modeling and

theoretical verification. Modeling has several levels with different degrees of reflection of the properties of a real object. It can be functional, which checks the correctness of the logical structure of the device, temporal, which takes into account signal delays in device circuits without a final path topology, etc. As a result of the simulation, there may be errors that need to be corrected, which enriches the iterative design process. After successful completion of the physical simulation, the device is ready to be installed in the system.

2.3. Methods and means of automated design of digital devices

All modern methods of designing processors based on complex VIS/HVIS programming are based on the use of CAD. The right choice of CAD is the most important prerequisite for effective design and production acceleration.

Design methods and tools are closely related to the choice of CAD and vice versa, the choice of CAD determines the acceptable and appropriate methods of the design tool, therefore,these issues cannot be considered in isolation from each other.

Methods of project description.

The use of CAD requires effective, visual, manageable and controllable means of project description. A designed device can be described in many ways, and the most common method is to describe the design as a whole.

Currently, the most common universal description methods used at each level of the project hierarchy are graphics and text. Direct division of PLD circuits in the topology editor, description in the form of necessary time diagrams, etc. is less used. Each of the methods of describing the structure has its advantages and disadvantages. The similarity of the method of description, internal organization and operation of the calling device significantly reduces the time of creating the project, simplifies its testing and, as a rule, turns out to be the most visible and understandable.

The graphic representation of the project is created in the database of elements of the library, acceptable for the selected CAD, for example, in the database of elements of the standard TTL series. The main advantage of the graphic method is its traditionality and visibility, which are connected with familiarizing designers with the perception of schematic images. Of course, this advantage is revealed only after a proper hierarchical and structural analysis of the project.

Modern hardware description languages (HDL, Hardware DescriptionLanguages) make it possible to describe the developed device, both in terms of its behavior and from the point of view of its design. These capabilities make it increasingly common to present the project in the form of a textual description of the algorithms of its fragments in combination with a textual description of inter-block connections for complex projects. The advantages of the text method of project description are its compactness and relative ease of automating all transformations, including the initial generation of the project description. It is very important to be able to use standard universal languages, such as HDL, which provides easy transfer of the project from one hardware platform to another and transition from one CAD to another.

Unlike textual, graphic methods of presenting the project are usually highly specialized and require special means of transferring information about the project to another environment.For this purpose, special universal languages for transmitting information about the project, which are becoming increasingly widespreadcan be used. Text descriptions are divided into two main types - low-level languages and high-level languages.

Low-level languages are closer to the hardware, creating potential opportunities for compilers to create projects with more favorable parameters. The price for this, of course, is a strong focus on specific equipment and the company that produces it. Examples of such languages include AHDL (AltéraHDL) and ABEL (Xilinx). Using low-level languages makes it easier to create designs with the best timing options,

as the designs will take into account the architecture specifics of a particular CPLD or FPGA.

High-level languages are less tied to hardware platforms are therefore more versatile. The most common languages are VHDL and Verilog among them. These languages, like other high-level algorithmic languages, basically allow any algorithm to be described in a sequential form, that is, through a sequence of assignment and decision statements. The main difference is the ability to display actions performed in parallel on the hardware, represented by individual parallel processes.

When designing a processor, it is most effective to divide it into two blocks: operational and control. The operational unit (OB) performs data conversion and consists of standard parts, and the control unit (BC) provides the necessary sequence of operations performed in one or moreOB. For this purpose, BC sends control signals to the inputs of the OB. The sequence of actions and control signals depends on the results of the operation in the OB. It follows from this that it is convenient to define BC in terms of a finite state machine with memory of one or another type. In complex designs, it is possible to divide the processor into several functional weakly connected pairs of OB-BC at the same level of hierarchy or to create a pair of hierarchical connections of this system.

The operating unit is usually represented by a set of registers, logic circuits (usually multifunctional and controlled), buffer circuits and switching connections between them.

2.4. Stages design procedures

In general, the processor development procedure is presented above (see fig. 1.2). Designing using CAD will be considered in more detail below. The development occurs in the following order:

Compilation of a meaningful graphic diagram or a functional algorithm of the structural diagram of the device. The first task consists in the transition from the technical specification (TS) to the formalized description of the designed device. The technical specification as a rule, is a mixture of verbal and technical description, its formalization leads to the identification of the main blocks of the device (or algorithm) and the determination of their connections and interactions. The method and manner of division are most often and primarily determined by the preferences of the designer and are fixed only occasionally. The CT form itself may define certain activities for the designer, although another way of describing the project or its fragments may be more effective. In fact, the first steps of the first stage are being carried out at the moment. Formally, the first stage is the division of the task into functionally distinct subtasks, the stage of decomposition.

Decomposition can be reduced to a combination of schemes of algorithms for the work of fragments or a functional scheme of the device and its parts. A possible option for fairly complex systems would be a judicious combination of behavioral and structural design decomposition. Division does not occur only within one level of the hierarchy. Most projects are also divided into hierarchically organized levels. An important task that must be solved at this stage is the clarification and coordination with the client of the functions of thedesign interface. Implementation of external exchange protocols is determined. Temporary characteristics and rules of interaction with external devices determine the acceptable organization and structure of internal nodes of thedesign.

The use of CAD at this stage of design is still quite rare, although for the implementation of modern, very complex projects (several hundred thousand fittings), special block editors are increasingly used, which allow the decomposition of the project without legalization of the component parts.

An example is Altera's Quartus CAD, which includes a special block-level editing tool.

Development of the overall structure of the project.

The main task is the selection of acceptable elements for the implementation of each level of the hierarchy, the determination of the relationships between them, and if the parameters of the elements are adjustable, then their adjustment. Several points are decisive: on the one hand, it is a source of a set of acceptable elements, on the other hand, it is a way of describing the connections of elements with each other and, if necessary, the possibility of describing new (specific for this project) elements.

When developing devices with a digital representation of information, it is natural to divide them into two blocks: operational and control. The operational unit (OB) performs the transformed data and is built from standard parts (behavioral parts), and the control unit (control device, BC) provides the necessary sequence of operations performed in the OB (one or more). For this purpose, BC sends control signals to the inputs of the OB. The sequence of actions and/or control signals depends on the results of work in the OB and external influences. It follows that BC is conveniently installed in the form of a finite state machine with memory of one or another type.

In complex structures, it is possible to divide the BC into several functionally weakly connected OB-BC pairs at the same hierarchical level, or to create a pair hierarchically embedded in the OB (less often in the BC). The resources and capabilities of modern CAD programs force us to approach the design of machines in a slightly new way. Two main problems should be noted here. Firstly, the problem of optimal coding of states and transition conditions of an automaton is of more theoretical interest than practical for programmers. The developer most often only defines how to code the machine, and the specific coding is done by CAD. Only in some cases (usually based on an understanding of the speed of execution of specific transitions or the creation of an output signal) it is advisable to manually code the machine and impose the accepted version on the CAD. The possibility of such a certain definition of standard compliance is supported either by assigning permanent

assignments to individual elements of the designed structure in the CAD program, or by using a more detailed method of describing the design structure.

Secondly, the organization and structure of FPGA cells (as well as the practical absence of restrictions on the complexity of automata connections) allow to significantly expand the range of synthesized structural diagrams of CAD automata, the structures of canonical automata of the Miley and Moore type whichturn out to be intertwined in various combinationsduring the synthesis.

A meaningful description of the project and its parts.

The implementation of CAD allows creating effectively managed and controlled descriptions of projects and their parts. In addition, the same device can be described by different CAD tools. The methods used are usually suitable both for describing the project as a whole and for describing its individual fragments. Moreover, most CAD programs allow you to convert one type of description to another. Currently, the most common, universal way of describing a project, which is used at any level of its hierarchy is graphics and text. Direct drawing of diagrams in the topology editor, description in the form of necessary time diagrams, etc. is less commonly used. Each of the methods has its advantages and disadvantages.

The graphic representation of the project in modern CAD can be created both on the basis of the designer's graphic designations and on the basis of library elements acceptable for the selected CAD, for example, elements of the standard TTL(III) series. It is permissible to mix these two bases in different combinations. The main advantages of the graphic method are its traditionality and transparency, which are related to the creators' knowledge of the way of perceiving diagram images. Of course, these advantages are revealed only with the correct hierarchical and structural distribution of the project.

Depending on the purpose, the programmer should choose one or another editor. The result of the editors' work is the creation of a description of the project and/or its fragments in one of the languages of the equipment description. Even if

CAD does not support graphical input, it can display a textual description in graphical form.

The lack of exact standards of correspondence between textual and graphical representation can be attributed to the disadvantages of graphical presentation. Unlike textual, graphic methods: project presentations are usually highly specialized and require special means of transferring information about the project to another environment and special universal languages for transferring information about the project can be used for this purpose.

Modern hardware description languagesallow you to describe the developed device both from the point of view of its behavior and from the point of view of its design. These possibilities make it common for complex projects to represent the project in the form of a textual description of the algorithms of its fragments in combination with a textual description of inter-block connections. The advantages of the text method of describing the project are its compactness, the relative ease of automating all transformations, including the initial generation of the project description. It is very important to be able to use standard, universal languages, such as HDL, which provide easy transfer of the project from one hardware platform to another and the transition from one CAD to another.

The basis of the architectural and constructive description of the operation unit is the task of building connections of individual elements. The traditional graphic way of representing the structure of connections, which remains the most visual way is accompanied by a textual way of description in modern CAD. The tools included in the CAD program allow automatic, direct and reverse conversion of descriptions.

The composition of the elements of the operation unit depends on the composition of the used library. Most CAD supports a hierarchical description of projects. At each level, the project is represented as a set of elements of that level. The set of functionality of library elements offered by standard CAD is extremely wide, and libraries are divided into:

• standard libraries of the CAD development company, the content of which corresponds to one or another common MIS or SIS series (for example, the IS type 74 series);

• standard elements of computing equipment (logic elements, decoders, multiplexers, counters, etc.), the parameters of which are constant;

• typical elements of computing equipment (counters, registers, multiplexers, etc.), the detailed parameters of which (capacity, polarization of control signals, etc.) can be freely specified by the designer;

• typical nodes of computer systems (peripheral devices, hardware cores of microcontrollers and microprocessors), some parameters of which are changed by the designer (as a rule, the configuration of these nodes is developed either by the company developing the node or in cooperation with it);

• elements created by the designer and linked in the designer's library.

Each project can be used as a subproject in a more complex project. It is impossible to create a built-in library of designer modules, although it is advisable to have your own design modules or a database of ready-made projects.

As a rule, at each level of the hierarchy the basic set of elements of the operating unit (at this level) is supplemented by a set of registers, logic systems (multifunctional and controlled), buffer circuits and the necessary switching connections between them for its functioning. It is important that at the lower hierarchical levels of the project description there is an unambiguous interpretation of the functioning of all elements of the OB.

At this stage, the functioning of the control unit is determined, which ensures the necessary interaction of the elements of the operating unit. It should be emphasized that the last two stages are highly interdependent and, if not developed in parallel, are implemented iteratively.

There are different forms and methods of describing a machine. The modern trend is to move from the notation of logical expressions, limited by the provisions of the Constitutional Tribunal, to a graphic form. The description in the form of a transition graph (state diagram) is becoming one of the most common variants of automatic tasks. Graphical editors for creating automata are part of the initial design tools of modern CAD.

Editors from different HVIS PL manufacturers have their own specifics, but all of them are distinguished by exceptional simplicity, naturalness and convenience, andas well as the absence of an absolute need to know the source language of the editor. The highest-quality versions of design programs have a complete set of tools for performing the entire design procedure of project development, which allows you to implement the following operations:

• draw atransition graph containing names of states, direction, conditions and priorities of transition conditions, generated signals and methods of their formation;

• check the correctness of the prepared transition schedule (repetition of names, ambiguity of transitions, incorrectness of transitions, etc.);

• compile the project (from the source text file) in the selected mode;

• simulate the operation of the automaton in interactive mode and compilation mode.

An important advantage of StateCAD is the possibility of a wide choice of forms of presentation of results (description in high-level languages VHDL and Verilog and in low-level languages ABEL, AHDL).

The specificity of the products of a particular company is also reflected in high-level languages, expressing, first of all, the difference in the libraries required for work, as well as in the complexity and variety of syntactic structures acceptable to compilers. The final results of the compilation of the same original graph scheme or the automaton of the subsequent compilation of the same program from a high-level

language to the PL chipset initial file, obtained from compilers of different companies can be significantly different and have different performance. StateCADVersion 3.2 of the WorkviewOffice package is convenient in that, before translating the transition diagram, it is necessary to specify not only the desired representation of the language (VHDL, AHDL, Verilog, ABEL, etc.), but also reserved attributes, which allows you to optimize the notation of the machine and avoid the use of syntactic constructions.

As already noted, when using graphic editors, the user does not need to know the source language of the editor. However, in some cases it is extremely useful to have it. The usefulness of orientation in linguistic constructions is revealed, for example, in situations where the automaton needs to be minimized by one or another parameter, primarily by the time intervals between generated output signals, which can lead to temporary signal failures. It is in such cases that the knowledge of the language and the skill of the designer make it easier to get the best results.

When the project and all its parts are ready, you can proceed to the most important stage of designing - drawing up the project. A compilation can include both the entire project and its components. Compilation of separate fragments, on the one hand, simplifies the project, as it reduces the size of the analyzed problems, on the other hand, it requires taking into account different ways of functioning of internal resources and external elements. All hidden errors and inconsistencies are detected during compilation. Compilation is divided into a number of sub-steps: collection of the project database, connection control, logical minimization of the project, creation of the startup (configuration) file, etc. At each stage errors may occur that require recompilation after they have been corrected.

The process of compilation in CAD of PL VIS manufacturers differs from compilation in CAD of other companies.The main difference is the absence of the step of creating a start file. As a rule, the effect of the external compiler is the structure of the project based on the selected FPGA family. A classic example of such a compiler is the LeonardoSpektrum package, a package that supports design

for FPGAs from Actel, Altera, AgireSystems, QuickLogik, Xilinx, etc. Itprovides text input in VHDL, Verilog, proofreading at the level of register transfers, optimization based on constraints set by the programmer, analysis of the project's time characteristics, creation of the source information necessary to place and connect the project to the selected VIS and review the project at the gate level.

The result of compilation using CAD from manufacturers of HVIS is a start file, that is, configuration information for the selected programmable microcircuit. In addition, a report file is created that contains all the information about the build process and its results. There is a significant difference in procedures construction of CPLD and FPGA schemes. Except for the automatic placement and tracking of FPGA connections, in general, adjustments made by the developer are allowed at any stage of the process. This action is used by topology editors, which allow you to change the structure of project crystals and increase the efficiency of the designed devices. For CPLD, the designer's influence on the structure of the compiled circuit is possible only through indirect intervention, by changing the design description or changing the parameters set before compilation. After the successful completion of the project or its part, you can proceed to its verification.

Testing the developed device or its fragments is one of the most important design stages, since there are practically designs without defects. Finding design flaws is a difficult task, the speed and quality of which depends on the experience of the developers.

In modern CAD the most common testing is working with the time diagram editor, these editors are divided into compiling and interpreting. Editors of the interpretation type simplify the procedure of debugging the project and finding its defects related to the incorrectness of the creator of the structural or behavioral way of implementing the system or the characteristics of the implementation of the used element base. In the 6-window interpretation type CAD, the simulation results at a certain point in time are simply displayed on all types of design presentations (signals, electrical circuits, topology), which allows you to easily change the flow of

the experiment and the composition of the signal display. In the case of a schematic description of the design, the signal tracing is simplified.

The advantage of assembling modeling systems is the minimization of time costs. At the same time, it remains problematic to determine the correspondence between lines of text and the states of individual signals. The division was given two approaches to generating external impact on the project. One approach is to create these effects by defining the timing sequence of the input signals in the timing diagram editor. Another approach (convenient for smaller project tasks) is to write a specialized test program.

In most real-world digital devices several cycles are performed after initial data input. The operation of the device should be tested on several data sets of the same type, so that the following structure of the software module representing the test effect can be used: initial fill signals are generated, then two nested loops are executed, and the inner loop sequentially generates test signals to act on the inputs of one set data, and their change occurs in the outer loop.

Modern CADs have complete information about the design of the designed device and time parameters in all components - this allows you to automate the process of calculating various time characteristics of the project.

For example, CAD MAX+ PLUSII allows automatic calculation of three main classes of time parameters:

• minimum and maximum delays between sources of input signals and receivers of output signals, information about which is provided in the form of a delay matrix;

• the maximum performance of the device in the form of the maximum clock frequency of the memory elements used in the design;

• preliminary setting and waiting time of the signal, which guarantees the reliable operation of the circuits when fixing the signals in the elements of synchronous memory.

Many CAD programs also allow you to highlight critical paths of information transfer and transformation to obtain a schematic or topological representation of the project.

Although the performance of these calculations does not guarantee the detection of all design errors related to processor synchronization processes, it significantly reduces the number of such errors, or at least allows you to find places in the design that are dangerous from the point of view of failure.

One of the final stages of designing is the stage of experimental verification of the developed device. Despite all the accuracy ofprevious stages, there is always a far-from-zero probability that there will be defects in the design that may appear at the stage of implementation or even normal use of the device and cause serious consequences.

Carrying out full-scale experiments significantly increases the probability of defect-free products. The method of speeding up work at this stage and the possibility of moving it to an early stages of development, that is, before the end of the production of the final product are known system prototypes and methods of conducting experiments with them. Development boards were widely used in the past, especially in creating microprocessor systems.A similar situation occurs when developing systems and devices based on programming logic. Various foreign companies manufacture and supply a wide range of development boards containing programming logic and additional hardware (mainly fast RAM chips). Here you can specify Alter tools (Demo board); PLD programs (PCI bus _ _ _ evaluation board); Xilinx, Virtual Computer Corp. , Software for video (project boards HOT PCI set),etc.

It is necessary to involve hardware and software tools that will load the configuration, generate effects and control the correct operation of the tested devicewhen using product prototypes and n testing the final product. An important moment of conducting experiments is the generation of test reductions. In this regard, PL VIS opens up new, previously unattainable opportunities. An additional chip, and in some cases a part of the configured VIS resources, can be used as a signal generator. At the same time, the content of the research can be easily modified in the process of conducting experiments depending on the results of the previous stages of experimental work. A useful means of debugging can be a method of transferring data about the state of the object under investigation during the experiment to the computer of the device for visualization and detailed analysis. Many FPGAs and their corresponding CAD programs support this interaction. However, in many cases, it may be necessary to develop special software to accelerate the analysis of the behavior of the test object.Usually in certain situations it is acceptable to use serial devices such as multi-beam oscilloscopes, logic analyzers, etc.

After the successful completion of full-scale experiments with a test or a prototype, the designer must ensure the release of a pilot lot of the developed product. At the same time, the most important task is to ensure quality support for products produced in the form of IS PL.

Since the basic functionality of the project is no longer in doubt, the task of the research equipment used in this phase is different from the tasks of the equipment used in the previous phase. The testing equipment and the way it is used must detect defective products in the shortest possible time and at the minimum cost, and be performed by personnel with minimum professional requirements. And only for a small batch of products, it is possible to use test tools that allow localizing defects.

It is necessary to separate the requirements for the test equipment from the requirements for the project itself. The performance of test equipment largely depends on the foresight of the developer. The development procedure from the first

steps should be focused on the need to test the final product. If the designer (or the customer of the project) has correctly chosen the design goal, then after the production of product prototypes, the task of organizing the production of serial products may arise. At the same time, the main efforts of the developers are focused on the design and manufacture of hardware and software, which allows to reduce the price and speed up the production of serial products. If the demand for manufactured products and real retail outlets increases, the issue of product modernization can be considered, including the transition to a cheaper IC elemental base. This direction is related to the development of strategies and tactics for transferring projects to a new element base or switching to new technologies for the production of final products.

2.5. General information about CAD MAX+PLUS II

Until recently, MAX+PLUS II was Altera's only FPGA-based device design system. Only in 1999, a new generation Quartus design system appeared, designed for designing devices on APEX20K FPGA systems. MAX+PLUS II's integrated software gives you control over your logic design environment and helps you achieve maximum efficiency and productivity. All packages run on both the IBM PC platform and the SUN, IBM RISC/6000, and HP9000 platforms. In the future, we will consider working on the IBM PC platform.

For normal installation and operation of the CAD MAX+PLUS II system, an IBM PC-compatible computer with a processor no worse than Pentium, at least 16 MB of RAM and approximately 150-400 MB of free hard disk space depending on the system configuration is required . From my own experience, for the development of large chips on the FLEX10K50 and higher FPGA, it is desirable to have at least 64 MB of RAM (128 is better, 256 is even better, 384 MB and more is very good) and a Pentium II (P -3 does not really provide much benefit). Of course, you can use weaker machines, but then the compilation time increases and the load on the

hard disk increases as a result of the replacement. Increasing the amount of RAM and cache gives better results compared to increasing the processor clock frequency. If you don't plan to keep track of large crystals, then 32 MB of RAM is enough to get a good project compilation speed. As for the choice of operating system Windows NT is definitely better, Windows 95 OSR2 is worse, Windows 98 is worse, especially the localized version. This is due to the fact that the package was originally developed for Unix and does not take full advantage of all Windows mechanisms. This is especially noticeablein the temporary simulation of complex DSP devices, where screen redrawing takes up most of the time. Since the package is not localized, it is better to use non-localized (US or European) versions of Windows.

During the MAX+PLUS II system installation, two directories are created: \maxplus2 and \max2work. The \maxplus2 directory contains the system software and data files and is divided into the subdirectories listed in table 2.1.

Table 2.1. The structure of the \maxplus2 system directory of the MAX+PLUS II system

Sub directory	Description
.\drivers	Includes device drivers for WINDOWS NT (for WINDOWS NT PC platform installation only)
.\edc	Altera-provided command files (.edc) that generate output data files (.edo) for specific testing conditions.
.\lmf	Altera-provided macro library files (.lmf) that map user logic functions to equivalent MAX+PLUS II logic functions
.\max2inc	Include files (header files) with function prototypes for macro functions developed by Altera. Function prototypes display a list of ports (pins) for macro functions implemented in project text files (.tdf) written in AHDL

.\max2lib\edif	Contains the basic macro elements and functions used in the DIF user interface
.\max2lib\mega_lpm	Contains megafunctions, including Library of Parameterized Modules (LPM) functions, as well as files with corresponding prototypes in AHDL
.\max2lib\mf	Includes special and legacy macro functions (74 series)
.\max2lib\prim	core items provided by Altera
.\vhdlnn\altera	Contains the altera library with the maxplus2 software package. This package contains all MAX+PLUS II primitives, megafunctions, and macrofunctions supported by VHDL
.\vhdlnn\ieee	Contains the ieee library of VHDL packages including std_logic_1164, std_logic_arith , std_logic_signed and std_logic_unsigned
.\vhdlnn\std	Contains the std library with standard packages and text input/output tools described in the IEEE Standard Language VHDL Handbook R

The \max2work directory contains the instructional files and examples and is divided into subdirectories described in the table. 2.2.

Table 2.2. The structure of the working directory \max2work of the MAX+PLUS II system

Sub directory	Description
.\ahdl	Contains sample files that illustrate "How to use AHDL" in the electronic reference (MAX+PLUS II Help) and in the MAX+PLUS II AHDL manual

.\chiptrip	Contains all the project files for the chiptrip tutorial described in the MAX+PLUS II AHDL manual
.\edif	Contains all sample files illustrating EDIF functions in the electronic reference (MAX+PLUS II Reference)
.\tutorial	Contains a read.me informational file for the chiptrip learning project. All files created in the chiptrip project must be located in this subdirectory
.\vhdl	Contains sample files illustrating the topic "How to use VHDL" in the online manual (MAX+PLUS II Help) and in the MAX+PLUS II VHDL manual
.\verilog	Contains sample files illustrating "How to Use the Verilog HDL Verification Protocol Language" in the online manual (MAX+PLUS II Help) and in the MAX+PLUS II Verilog HDL Manual

MAX+PLUS II is developed by Altera and provides a cross-platform, architecture-independent development environment that can be easily adapted to specific user requirements. The MAX+PLUS II system has tools that allow convenient project entry, quick commissioning and direct device programming.

Shown in fig. 2.1 contents MAX+PLUS II System Software is a complete package that allows you to create logic circuits for Altera programmable logic devices, including the Classic, MAX 5000, MAX 7000, MAX 9000, FLEX 6000, FLEX 8000 and FLEX 10K families. devices. Information about other supported Altera device families can be found in the read.me file on the MAX+PLUS II system.

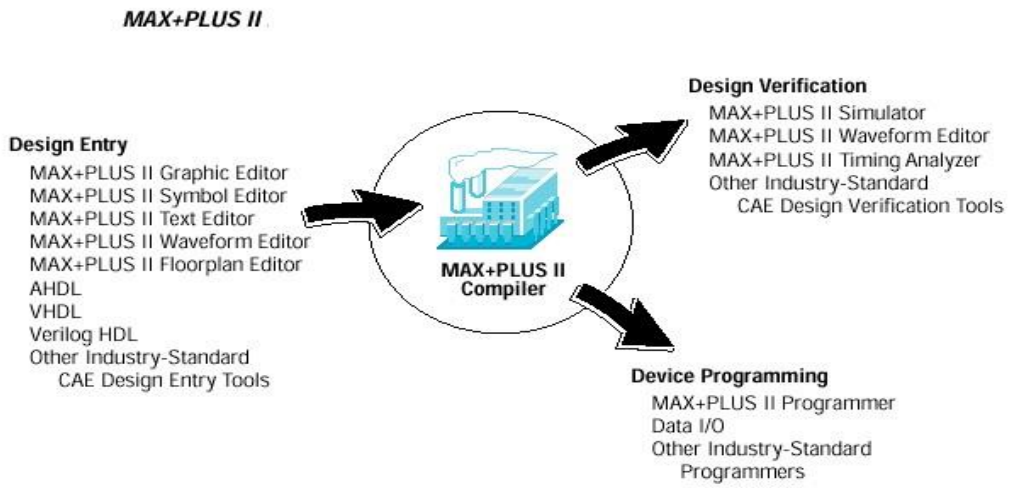
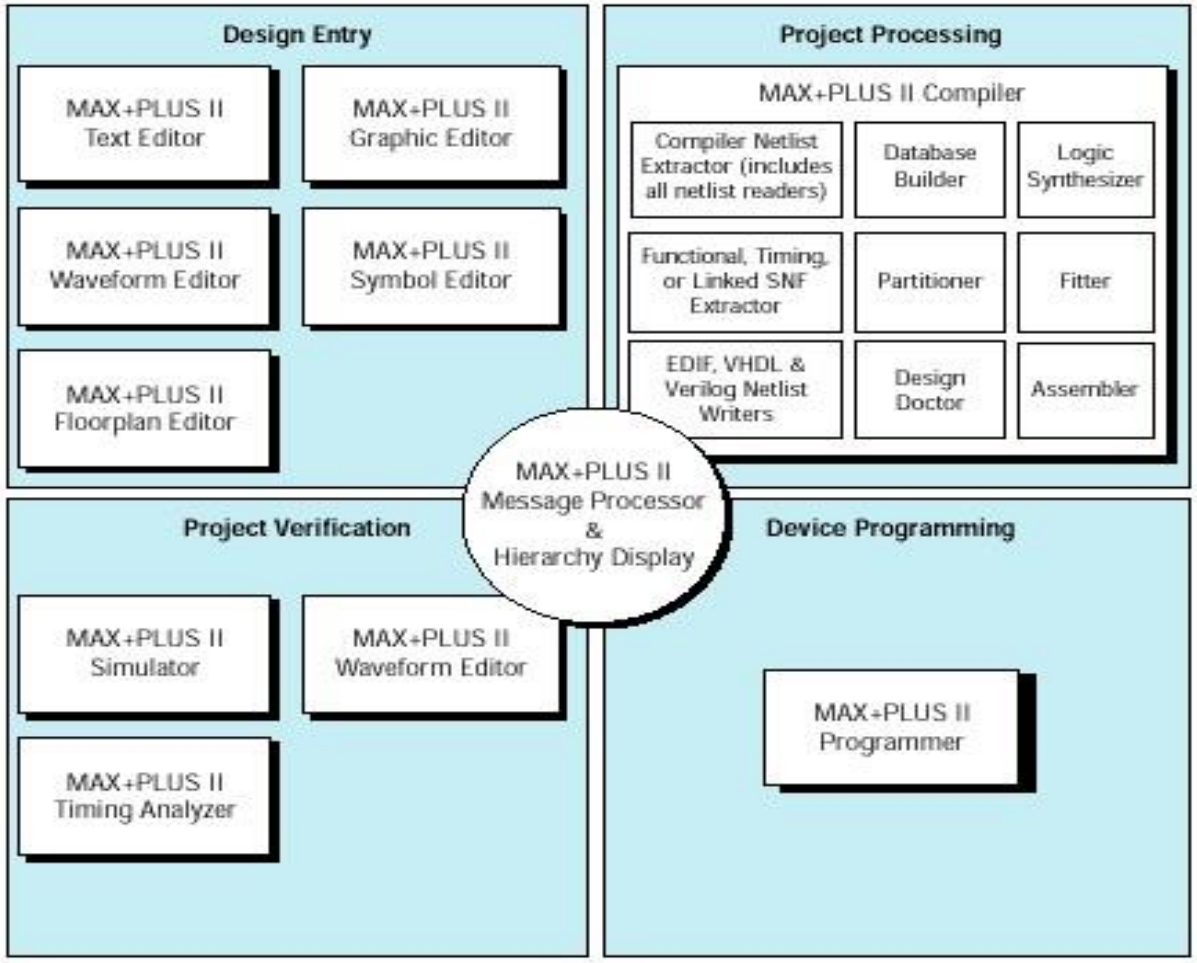


Fig. 2.1. The design environment in the MAX+PLUS II system

The MAX+PLUS II system offers a full range of logic design capabilities: a variety of design tools for creating hierarchical designs, efficient logic synthesis, synchronization compilation, separation, functional and timing testing (simulation), testing of multiple connected devices, analysis of system synchronization parameters, automatic fault localization and programming and testing devices. The MAX+PLUS II system can read and write AHDL files and EDIF trace files, Verilog HDL files, and OrCAD schematic files. In addition, MAX+PLUS II reads trace files generated by Xilinx software and saves delay files in SDF format for easy interoperability with other industry standard packages.

The MAX+PLUS II system offers the user a rich graphical interface supplemented by an illustrated online help system. The complete MAX+PLUS II system consists of 11 fully integrated programs (fig. 2.2). (A logical project,

including all subprojects) in the MAX+PLUS II system is called a project)



Lecture I.1. *Fig. 2.2. Applications in the MAX+PLUS II system*

To enter the design description (Design Entry), the project can be described in the form of a file in the language of the equipment description, created either in an external editor or in the text editor MAX+PLUS II (Text Editor), in the form of an electrical diagram using the graphic editor in the form of a temporary diagram created in the waveform editor. For the convenience of working with complex hierarchical projects, each subproject can be associated with a symbol that can be edited using the graphical symbol editor. The arrangement of nodes along the LB and FPGA contacts is performed using a level planner called the Floorplan editor.

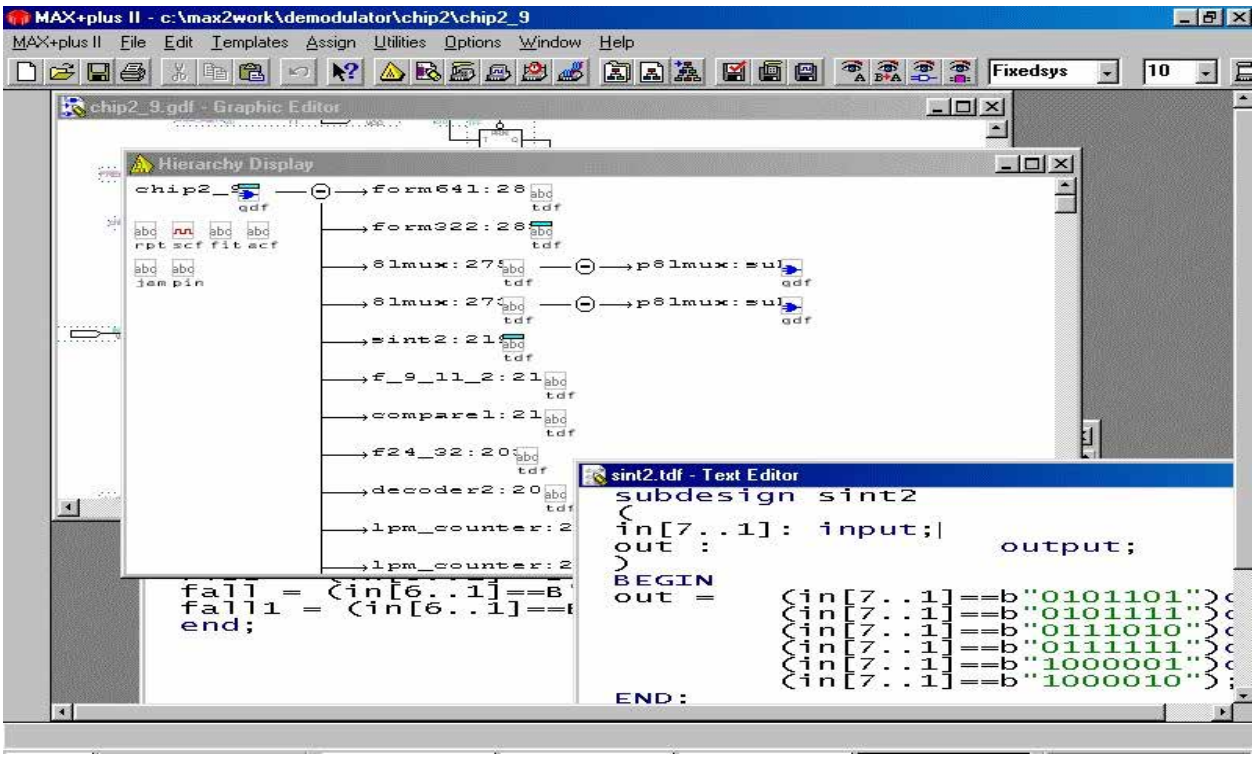
Project verification is performed using a simulator, the results of which are conveniently viewed in the Waveform Editor signal editor, in which test effects are created.

Compilation of the project, including downloading the list of networks (Netlist Extractor), creating a project database, logical synthesis (log synthesis), extracting temporary functional design parameters (SNF Extractor), partitioning (Partioner), tracing (Fitter) and generation of a programming or loading file (Assembler) are executed using the system compiler (Compiler).

Direct programming or loading of device configurations using the appropriate hardware is carried out using the Programmer module.

Many functions and commands, such as opening files, entering a device, assigning pins and gateways, and compiling the current project are similar in many MAX+PLUS II programs. Design editors (graphics, text, and signals) have a lot in common with secondary editors (level layout and symbolic). Each project creation editor allows you to perform similar tasks in a similar way (such as searching for a signal or symbol). You can easily combine different types of project files into a hierarchical project by choosing the project description format that best fits each functional block. Altera's large library of mega- and macro-functions, including functions from the LPM library, provides extensive design input capabilities.

You can work simultaneously with different programs of the MAX+PLUS II system. For example, you can open multiple project files and transfer information between them while building or testing another project. Or, for example, display the entire project tree and move from one level to another in the preview window, and the selected file will appear in the editor window and the corresponding editor for each file will be called automatically (fig. 2.3)



Lecture I.2. *Fig. 2.3. Hierarchical view of the project*

The compiler, which provides powerful tools for processing projects and allows you to set the desired modes of operation of the compiler is the basis of the MAX+PLUS II system. Automatic error localization, reporting, and extensiveerror documentation make project changes quick and easy. You can create output files in different formats for different purposes, such as function operation, synchronization and communication between multiple devices; analysis of time parameters; device programming.

2.1 Project development procedure

The procedure for developing a new project from the concept to implementation can be simplified as follows:

- creation of a new file (design file of the project or hierarchical structure of several project files using different editors for creating the project in the MAX+PLUS II system, i.e. graphic, text and signal editors;

- specifying the name of the top-level project file (Top hierarchy) as the name of the project (Project name);

- allocation of the FPGA family for project implementation. The user can independently assign a specific device or leave the evaluation of the necessary resources to the compiler;

- by opening the compiler window (Compiler) and starting it by clicking the "Start" button to start compiling the project. Optionally, you can plug in the Timing SNF Extractor to generate a logging file used for timing testing and analysis;

If the build is successful, you can check and analyze the timing by following these steps:

• to perform a time analysis, open the "Time" window, Analyzer, select the analysis mode and press the Start button;

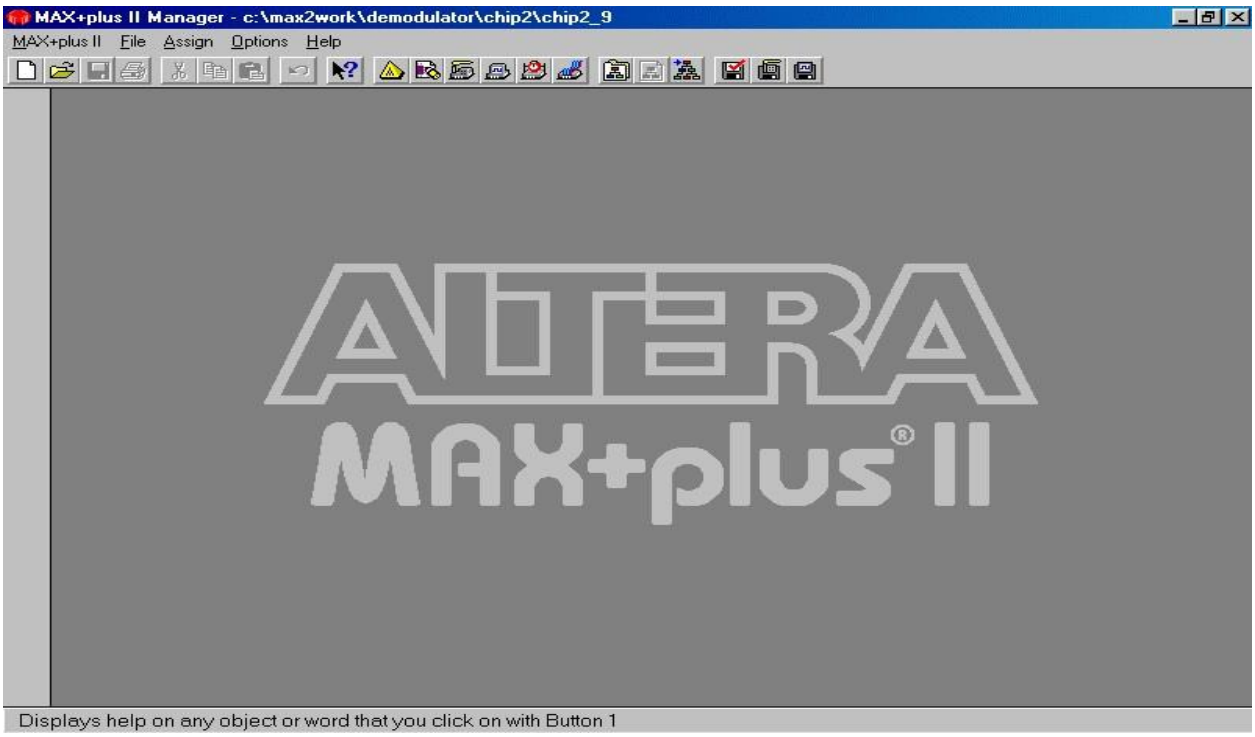
• to performa test, you must first create a vector test vector in a test channel file (.scf) using a signal editor or in a vector file (.vec) using a text editor. Then open the debugger simulator window and click Start ;

• programming or downloading the configuration is performed by starting the programmer module (Programmer) and then inserting the device into the MPU programmer adapter MPU (Master programming Block) or connecting the MasterBlaster , BitBlaster , ByteBlaster devices or the download cable FLEX Cable) to the device programmed in the system;

• selecting the "Program" button to program EPROM or EEPROM devices (MAX, EPC) or selecting the "Configure" button to load the configuration of a SRAM (FLEX) device.

Below, we will consider in detail the main elements of project implementation in the MAX+PLUS II system.

The MAX+PLUS II system can be started in two ways: by double-clicking the left mouse button on the MAX+PLUS II icon) or by typing maxplus2 at the command line.

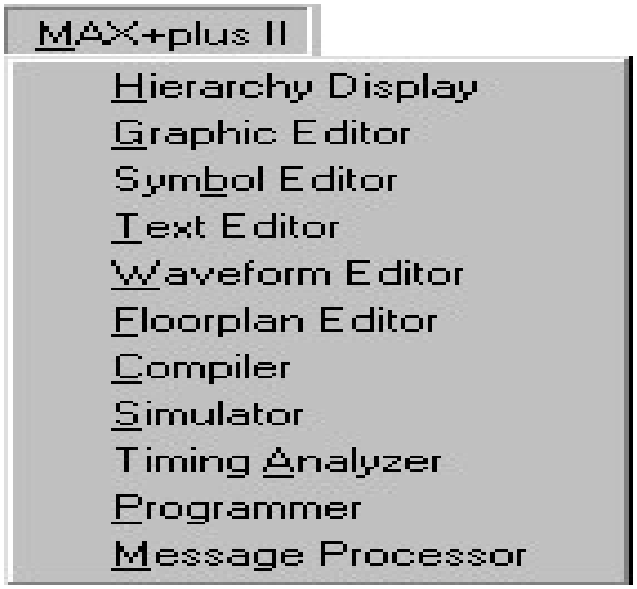


Lecture I.3 *Fig. 2.4. The main window of the MAX+PLUS II system*

After starting the MAX+PLUS II system, its main window opens automatically, the menu of which contains all programs of the MAX+PLUS II system (see fig. 2..4).

The project name and the current project file are displayed at the top of the window. Behind it there is a menu bar, and below- a panel of the main tools of the system, which provides quick access to its elements. There is a help/prompting bar at the bottom of the screen.

The elements of the system can be conveniently called using the MAX+PLUS II program, as shown in Fig. 2.5.



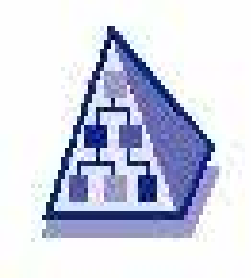
Lecture I.4. *Fig. 2.5. MAX+PLUS II menu window*

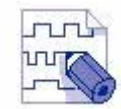
Let's take a closer look at the MAX+PLUS II menu (fig. 2.5). The MAX+PLUS II system software includes 11 applications and a basic control shell. Various programs that allow you to create project files can be launched immediately, allowing the user to switch between them with a mouse click or menu commands. One of the background programs, such as the compiler, simulator, timing analyzer, and programmer, can run simultaneously. The same commands from different programs work in the same way, which simplifies the task of creating projects. You can minimize the window of any application to an icon without closing the application, and then maximize it again. This allows the user to work efficiently without cluttering the home screen.

Table 2.3 shows the icons and the description of the programs.

Table 2.3 Application of the MAX+PLUS II system

Object	Function completed

	Hierarchy View – displays the current file hierarchy as a tree with branches representing subprojects. You can visually determine whether a project file is a diagram, text, or signal; which files are currently open; which supporting files in the project are available for editing by the user. You can also directly open or close one or more tree files and enter resource assignments for them
	Graphical editor - allows you to develop logic circuits in the actual WYSIWYG screen display format. Using Altera's proprietary primitives, mega- and macro-functions as basic programming blocks, the user can also use their own symbols
	Symbol editor - allows you to edit existing symbols and create new ones
! 	Text editor - allows you to create and edit project text files written in HDL, VHDL, and Verilog HDL hardware description languages. You can also use this editor to create, view, and edit other ASCII files used by other MAX+PLUS II programs. You can create files in HDL and other text editors, but this MAX+PLUS II text editor offers the benefits of context-sensitive help, syntax highlighting, and ready-made templates for AHDL, VHDL, and Verilog

# 	Signal Editor - has a dual function: a design tool and a tool for entering test signals and monitoring test results
$ 	level by level - allows you to graphically assign device contacts and resources of logical elements and blocks. You can edit the pinout in the device case drawing and assign signals to individual logic elements in a more detailed LAB view. You can also view the results of the latest build
%	The compiler is the logic of the process, designed for families of Altera classic devices. MAX 5000, MAX 7000, MAX 9000, FLEX 6000, FLEX 8000 and FLEX 10K. Most tasks are performed automatically. However, the user can control the compilation process
	in whole or in part
	Simulator - allows you to test logical operations and internal synchronization of the designed logical system. Three testing modes are possible: functional, timed and testing of several interconnected devices
!& ' 	Time analyzer - analyzes the operation of the developed logic circuit after its synthesis and optimization by the compiler, which allows you to estimate the delays that occur in the circuit.

(	Programmer - allows you to program, configure, verify and test Altera devices
(	Message Generator – displays error messages, warnings, and status information for the user's project and allows the user to automatically locate the source of the message in the source or additional file and in the destination plan.

The process of developing a multi-window project, when the hierarchy overview window and the text editor are open at the same time is illustrated in fig. 2.3.

Before you start working with the MAX+PLUS II system, you should understand the difference between design files, support files and projects.

A project file is a graphic, text, or signal file created using MAX+PLUS II graphics or signal editors or any other industry standard circuit or text editor or netlist writer available in packages that support EDIF, VHDL, and Verilog HDL. This file contains the MAX+PLUS II project logic and is processed by the compiler. The compiler can automatically process the following project files:

graphic files of the project (.gdf);

design text files in AHDL (.tdf);

project signal files (.wdf);

VHDL project files (.vhd);

Verilog project files (.v);

OrCAD scheme files (.sch);

EDIF input files (edf);

files in Xilinx Network List (.xnf) format;

Altera project files (.adf);

digital machine files (.smf).

Support files are files that are related to a MAX+PLUS II project but are not a part of the project hierarchy tree. Most of these files do not contain design logic. Some of them are created automatically by the MAX+PLUS II system program, others are created by a user. Examples of support files are assignment and configuration files (.acf), symbol files (.sym), report files (.rpt), and test vector files (.vec).

A project consists of all the files in the project hierarchy, including support files and output files. The project name is the name of the top-level ghjtrnf file without the extension. The MAX+PLUS II system compiles, tests, synchronizes, and programs the entire project simultaneously, although the user can simultaneously edit the files of that project in another project. For example, when creating a project1, the user can edit a TDF project file that is also a project file and save it; however, if it wants to compile it, it will have to set project2 as the project name first.

For each project, create a separate subdirectory in the MAX+PLUS II working directory (\max2work).

All the tools that allow you to create a logical design are easily accessible in the MAX+PLUS II system. Adesign development is accelerated by available standard logic functions, including primitives, megafunctions, the LPM parameterized module library and outdated 74-series macrofunctions. It is extremely harmful to use outdated libraries and port standard TTL-series circuits to

FPGAs. The design should be designed specifically for the FPGA architecture to get more or less reasonable results.

Project schematic files are created in the MAX+PLUS II graphic editor. You can also open, edit, and save diagrams created with the OrCAD Diagram Editor,projects on AHDL, VHDL and Verilog HDL in the text editor MAX+PLUS II or any other text editor.

Signal designs are created in the MAX+PLUS II signal editor.

EDIF and Xilinx developed with other standard EDA tools, can be imported into the MAX+PLUS II environment.

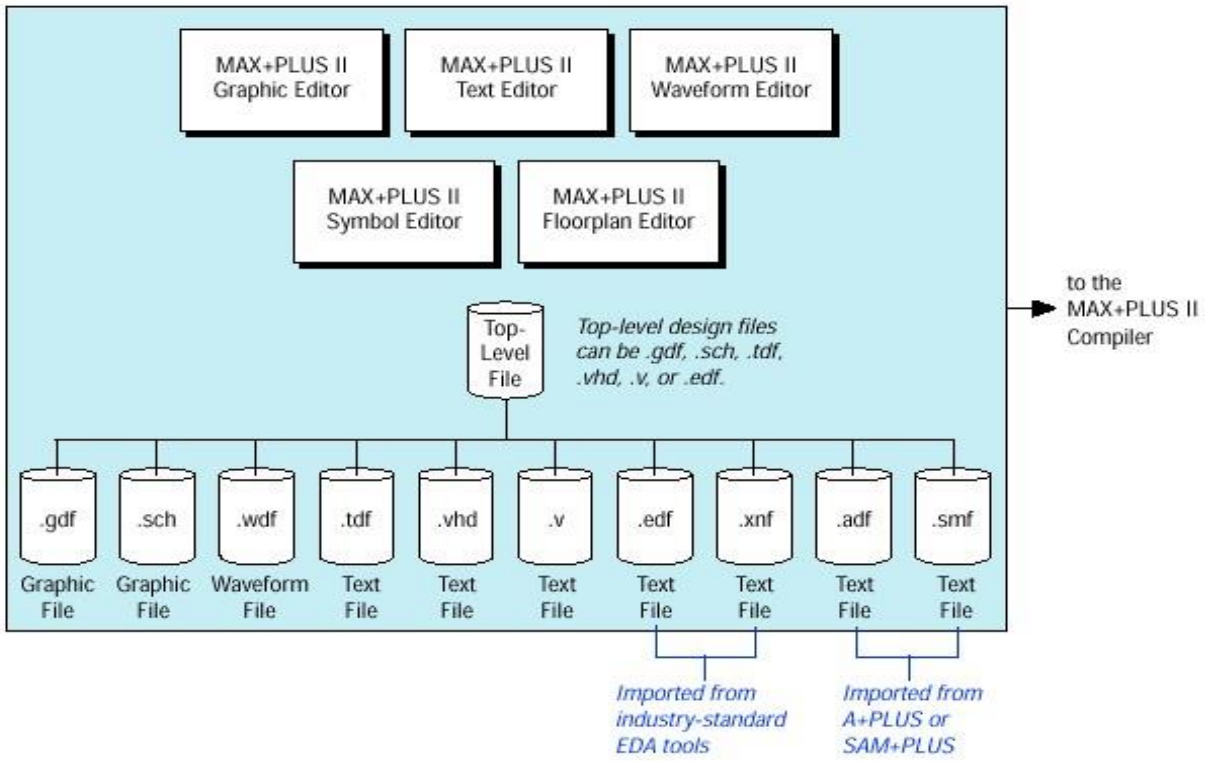
Altera A+PLUS and SAM+PLUS software packages can be integrated into the MAX+PLUS II environment.

The allocation of physical resources for any node or contact in the current project can be entered into the graphical environment using the multi-level scheduler. It saves project tasks in a file with the extension acf, which stores all types of resource assignments, probes and devices as well as configuration parameters (Assign) for the compiler, simulator and time analyzer.

Any type of project file can be automatically created in any MAX+PLUS II project editor using the File / Create command Defaults to the Command symbol. You can edit symbols or create your own and then use them in any diagram file in your projectwith the MAX+PLUS II Symbol Editor.

The use of files with extensions is mixed.gdfat each level in the hierarchical structure of the project. tdf vhd.v. edf sch .However, files with the extension . wdf xnf. adf smf must be at the lowest level of the project hierarchy or be a single file.

Methods of defining project files are shown in figure 2.6.



Lecture I.5. *Fig.2.6. Ways to describe project files*

“Assign” menu commands to enter, edit and delete resource assignment types, devices, and options that control project compilation, including logical synthesis, splitting, and alignment. The "Assign" menu commands areshownin fig. 2.7. The user can make assignments for the current project regardless of whether the project file or the application window is open. The MAX+PLUS II system stores project information in a file with the extension . acf. Changes to assignments made in the Level Planner window are also saved in the ACF file. You can also edit the project's ACF file in a text editor.

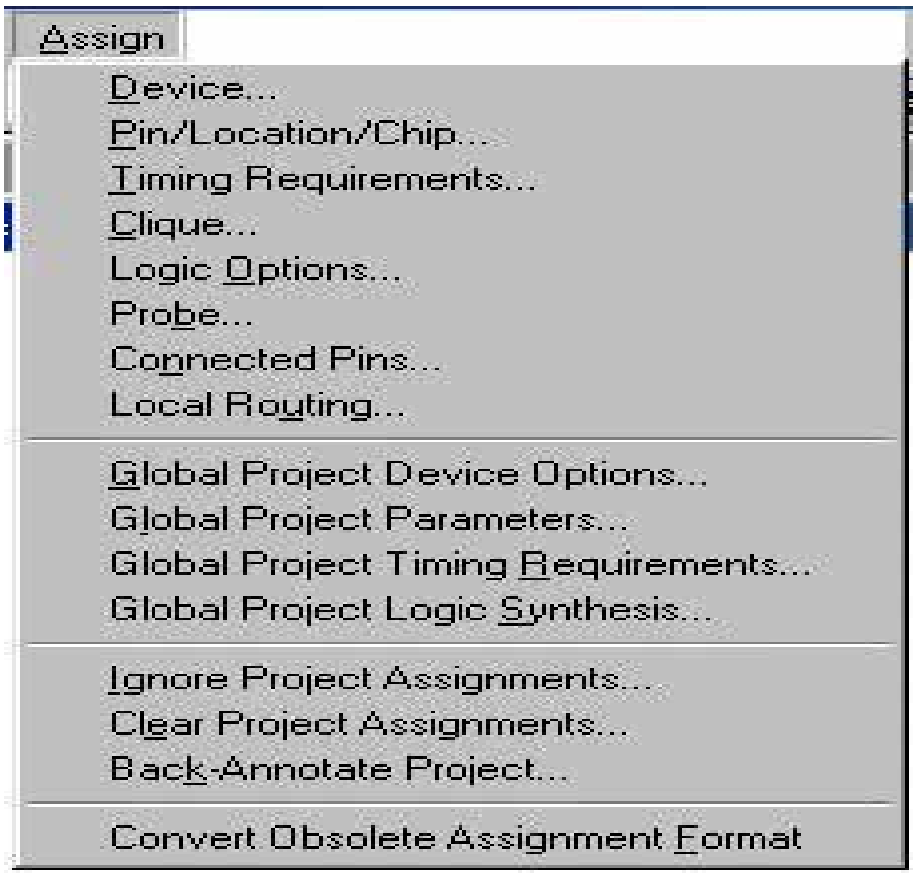
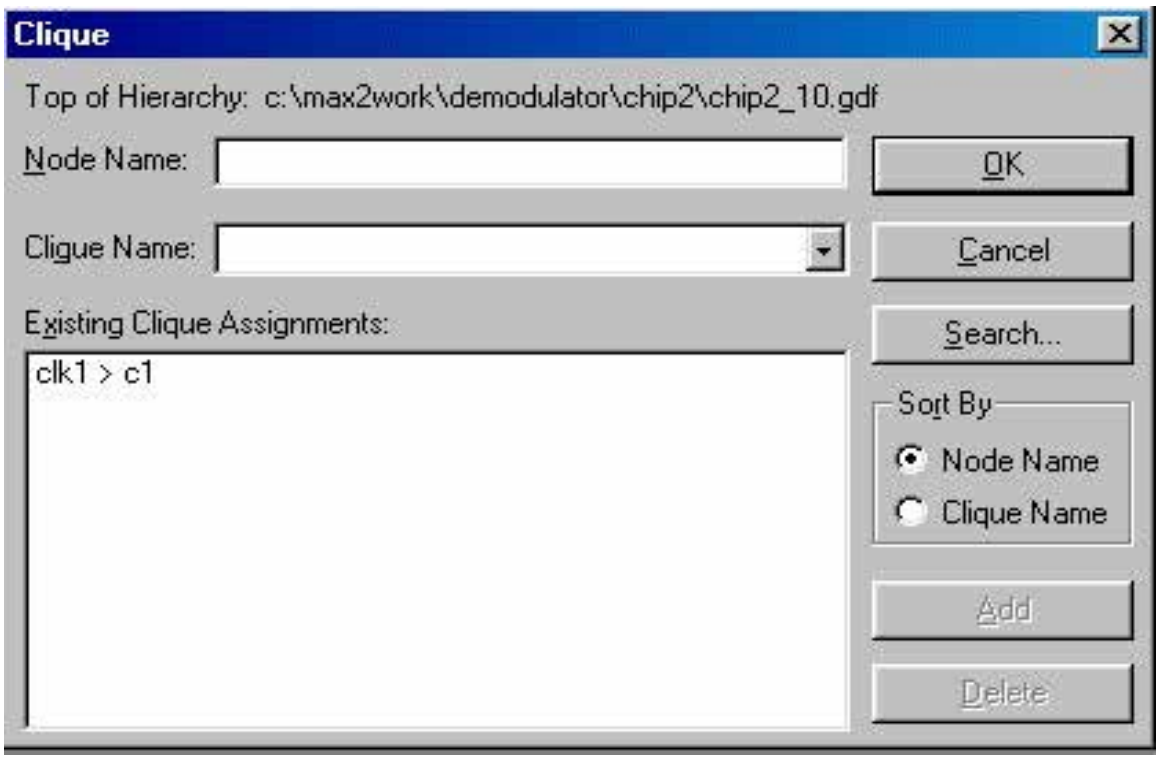


Fig. 2.7. Project assignment menu assign .

The following features are common to all MAX+PLUS II programs: device, resource and probe assignment; saving of the previous version; global device parameters in the project; global project parameters. global requirements for project time parameters; project of global logical synthesis. Let's take a closer look at them.

An Altera deviceis a contact or a logic element, that performs a specific, user-defined task. The user can assign logic to device resources to ensure that the MAX+PLUS II compiler adapts to the design as desired. The following types of tasks are distinguished.

The assignment clique specifies which logical functions should stay together. Grouping logic functions into cliques ensures that they are implemented in a single LAB logic structure block, EAB memory cell block, a row or a device. To assign a click the Assign / Click command is used (fig. 2.8)



Lecture I.6. *Fig. 2.8. Click assignment (Assign / Click command)*

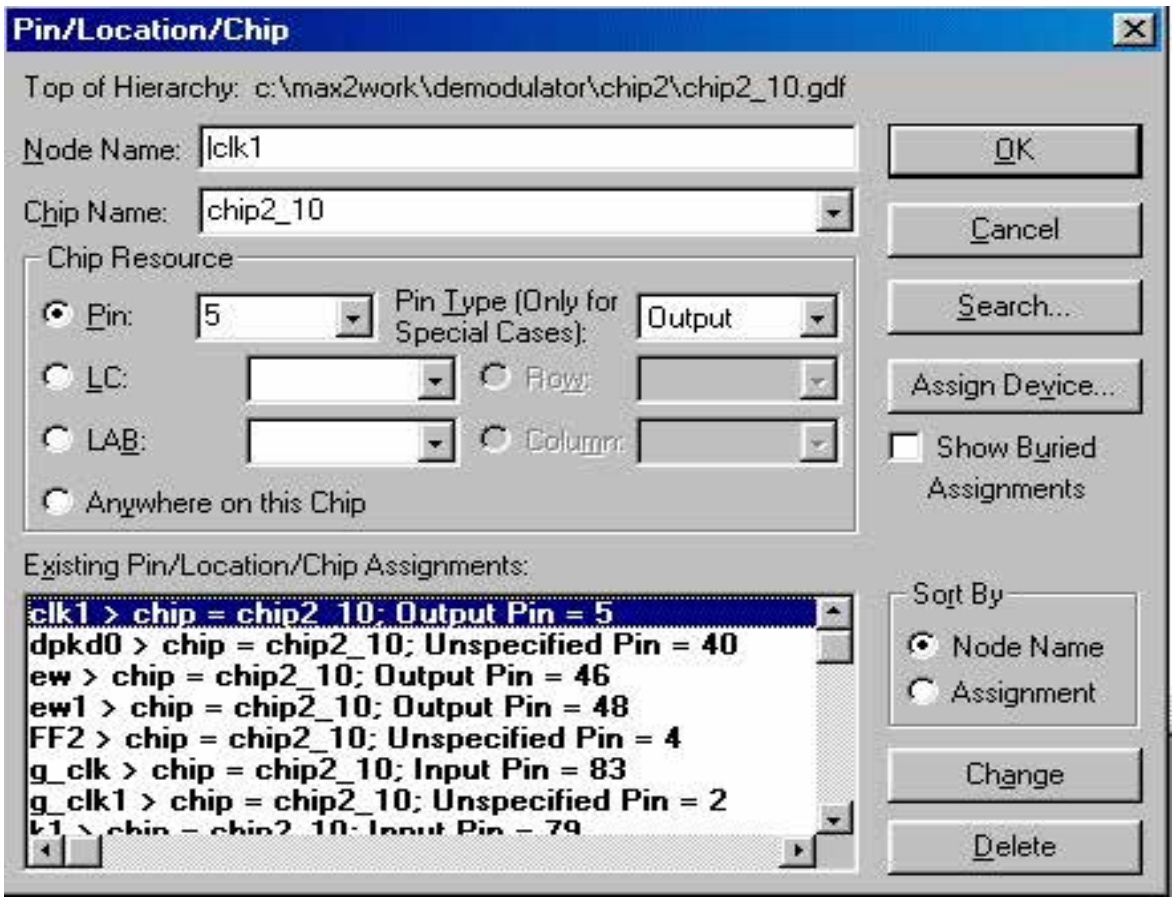
Name specifies the name of the click in the click field. In the Node field name specifies the name of the node. Nodes connected in cliques (sometimes called groups, but this name leads to confusion with the group concept in AHDL) will be placed in the same LB at compile time.

The assignment token determines which logical functions should be implemented in one device, if the project is divided into parts (several devices).

Pin assignment assigns the input or output of one logic function, such as a primitive function or a *megafunction,* to a specific pin or vertical (horizontal) row of pins on the FPGA.

A destination assigns a single logic function, such as the output of a primitive function or megafunction to a specific cell on an integrated circuit, such as a logic gate, I/O cell, memory cell, LAB and EAB blocks, horizontal or vertical rows.

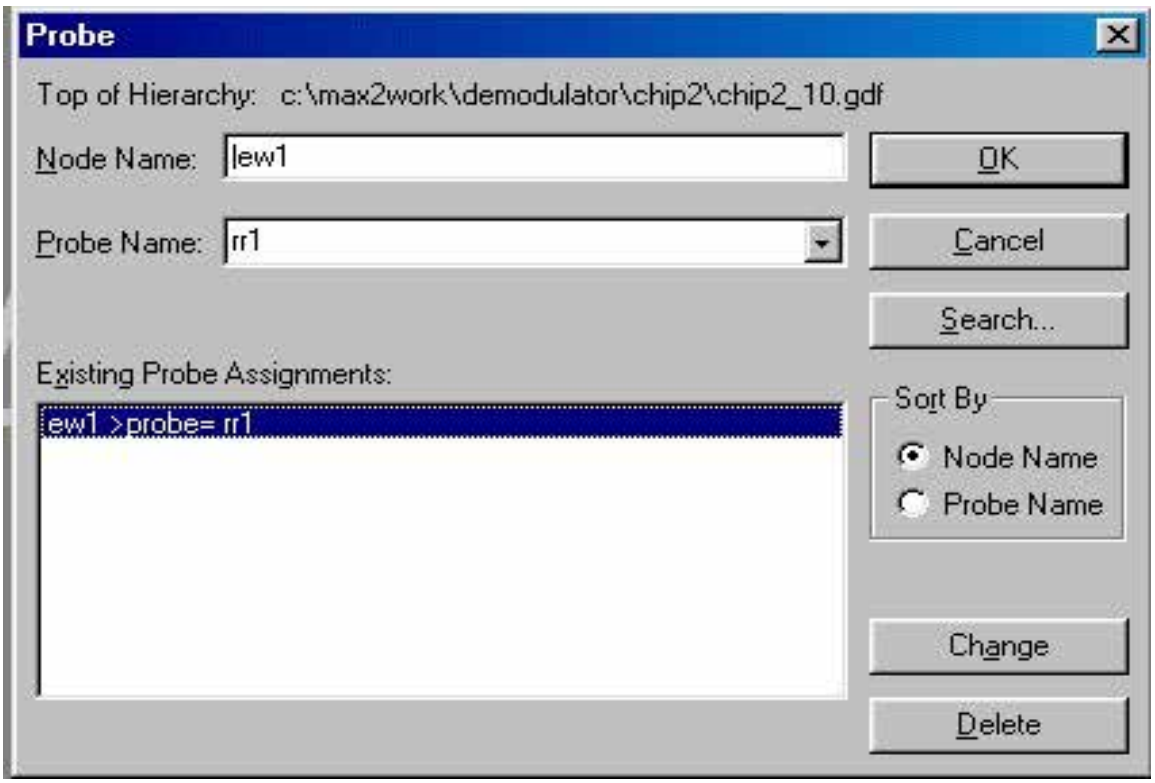
Pins, microcircuits and cells are assigned using the Assign / Pin / Location / Chip command, the window of which is shown in fig. 2.9



Lecture I.7. *Fig. 2.9. Command window Assign/Pin/Place/Create item*

You can set a PIN number (Pin), logic cell (LC) or block (LAB)in the fields of this window and use the "Change" and "Delete" buttons to change the assignment.

The assignment probe assigns a unique, *easy-to-remember* name to the input or output of a logic function. This task is very useful when modeling a system. To assign a probe, use the command Assign / Probe (fig. 2.10)



Lecture I.8. *Fig. 2.10. Command menu Assign / Survey*

The connected pin assignment defines the external connection of two or more pins in the user circuit. This information is also useful in timing testing mode and when testing multiple connected projects. To assign connected pins use the Assign / Connected Pins command (fig. 2.11)

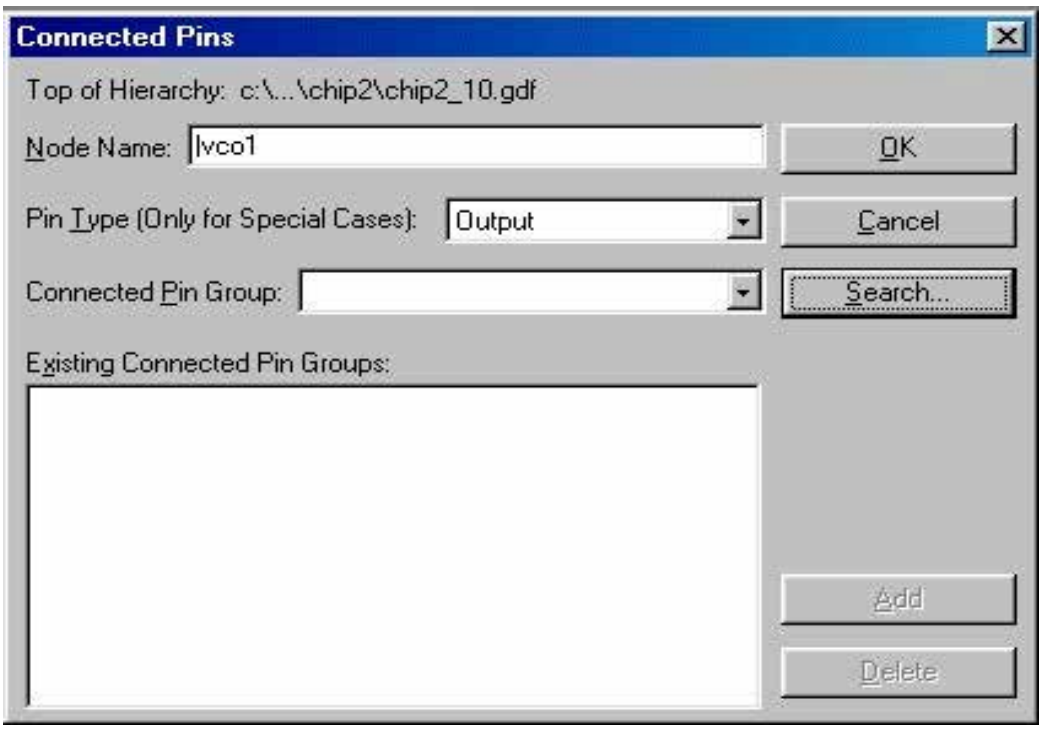
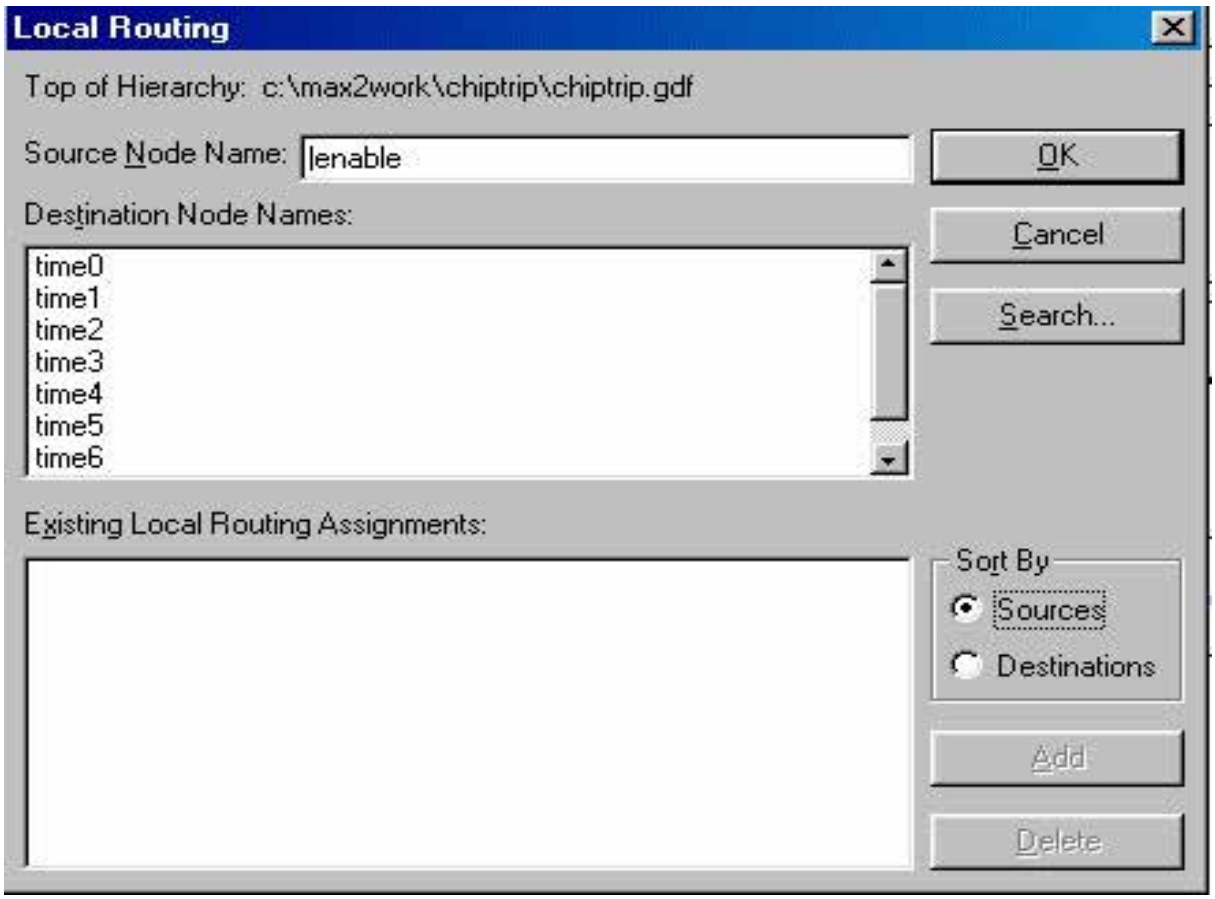


Fig. 2.11. Menu Assign/Connect contacts Commands.

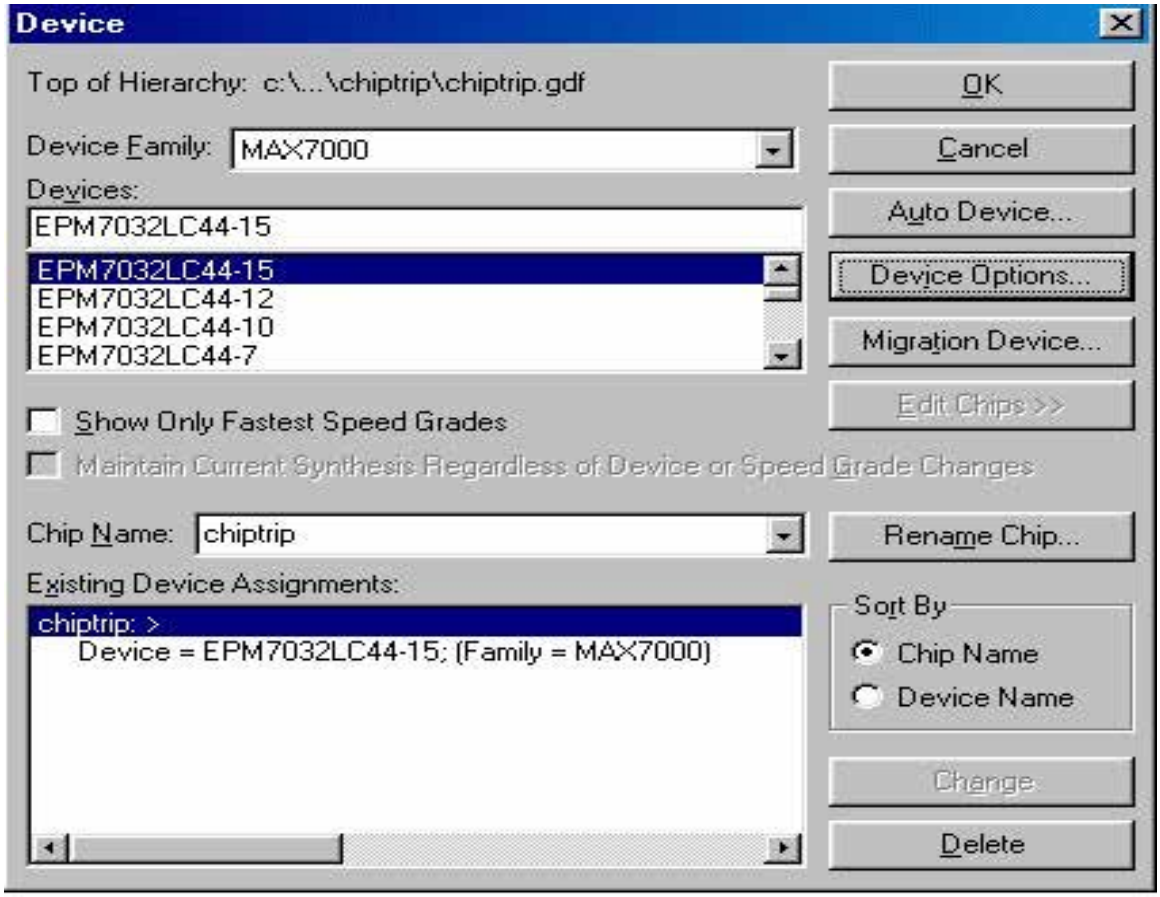
The local routing assignment assigns the node's output spreading factor to a logic gate in the same LAB as the node or to a neighboring LAB adjacent to the selected node using shared local connections. Local routing also occurs between the node located in the LAB block on the device circuit and the output pin to which it is connected. Assigning a local route is performed using the Assign / Local route command (fig. 2.12)



Lecture I.9.*Fig. 2.12. Destination/Local Routing Commands window*

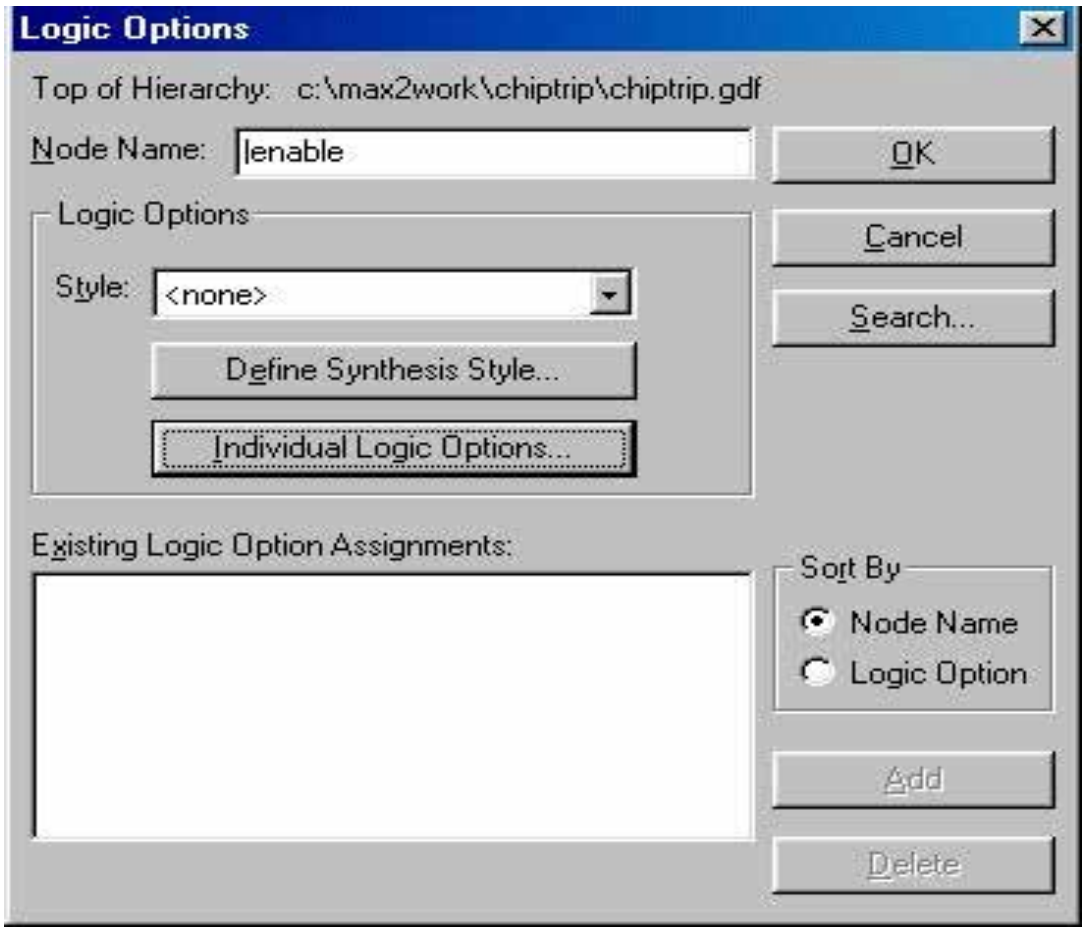
The assignment device assigns the type of FPGA in which the project will be implemented. If your project consists of multiple devices, this assignment type assigns chips to specific devices. You can also choose AUTO and let the compiler choose a device from a specific device family. You can control the automatic device selection process by specifying the scope and number of devices in the family. If the project is too large to run on one device, you can specify the type and the number

of additional devices. To select a device use the Assign / Device command(fig.2.13)



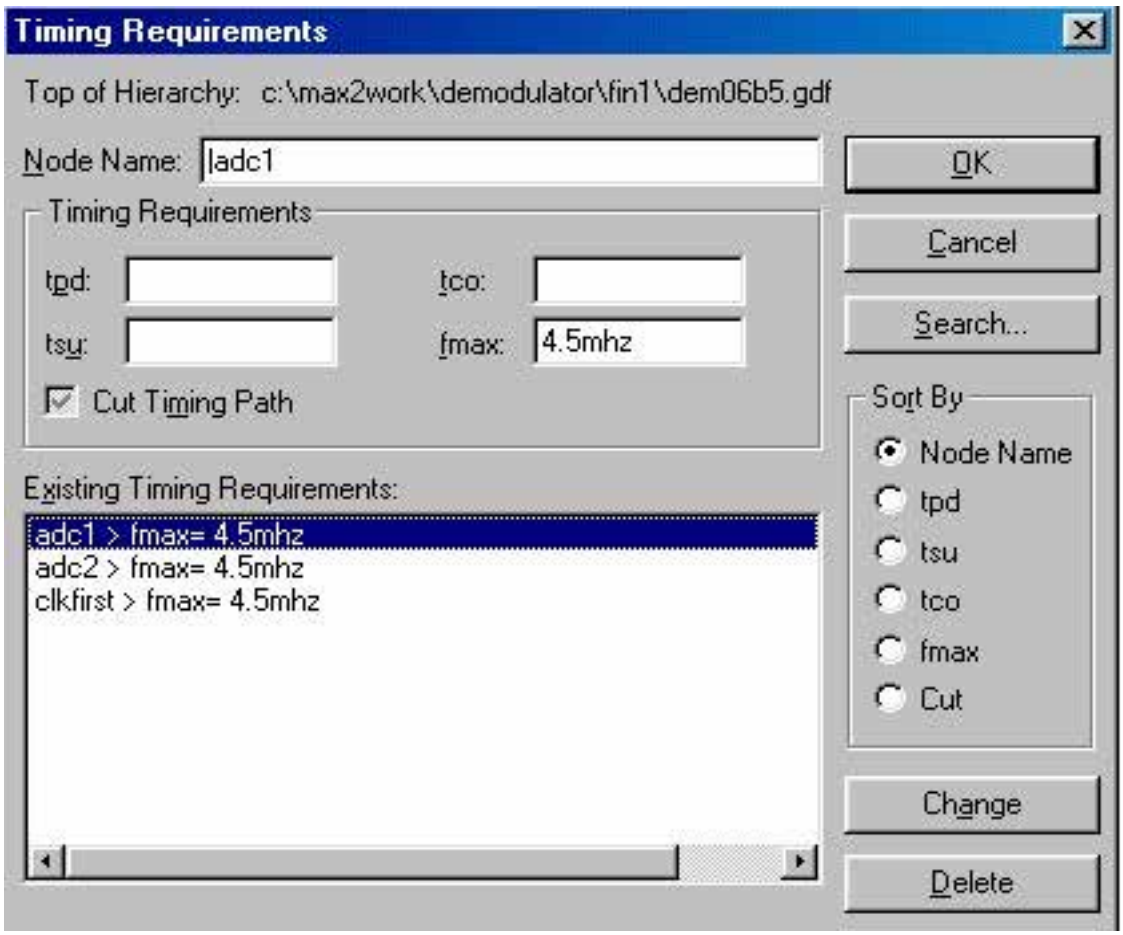
Lecture I.10. *Fig. 2.13. "Assign / device commands" window.*

The logic assignment option controls the logic synthesis of individual logic functions at compile time using the logic synthesis style and/or the individual logic synthesis parameters. Altera provides a large number of logical parameters, as well as pre-created styles, each of which is a set of logical parameters united by a single synthesis style name (Synthesis style). The user can use ready-made styles or create new ones. Synthesis styles allow you to customize the synthesis parameters for a specific device family taking into account the architecture of the family. To set synthesis styles, use the command Assign / Logic Options (fig. 2.14)



Lecture I.11. *Fig.2.14. Assignment of commands/ logical parameters window*

The timing of the assignment is controlled by the logic synthesis and setting of individual logic functions to obtain the required characteristics of the delay time t_{PD} (input - unregistered output), t_{CO} (clock signal - output), t_{SU} (clock signal - time setting), f_{MAX} (clock frequency). The user can also break connections between these signal paths (called "nodes" in the MAX+PLUS II system) and other nodes in the project. Assigning node synchronization parameters is performed using the Assign / Timing Requirements command (fig. 2.15)



Lecture I.12. *Fig.2.15 Assignment time requirements/ command window design*

From the "Assign" menu commands assignments can be made by right-clicking on the selected project node and selecting the appropriate assignment from the context menu (fig. 2.16)

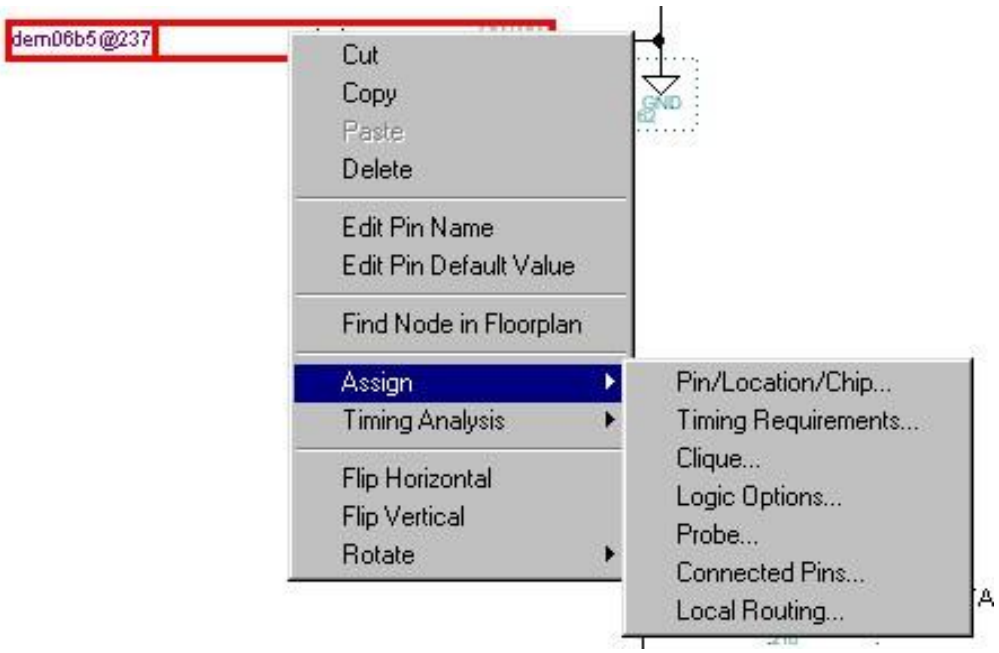
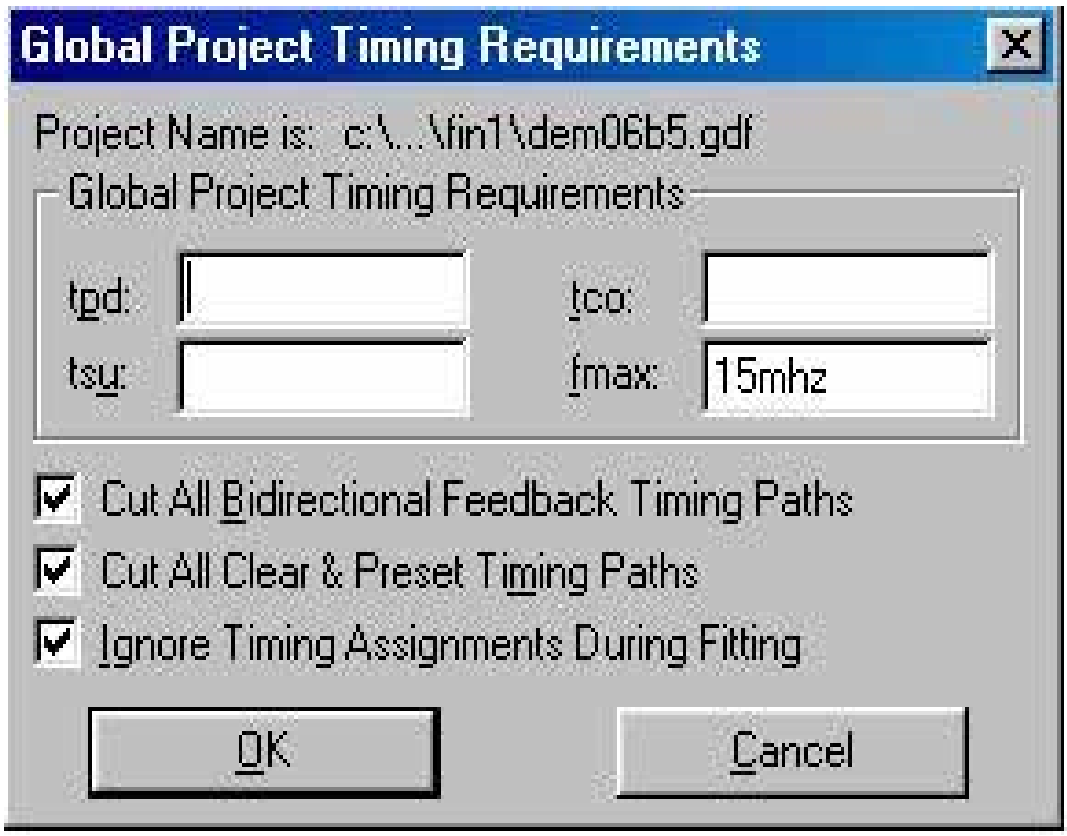


Fig. 2.16.

You can define global device options that the compiler will use for all devices when processing your project. To reserve additional logical capacity for the future, you can specify a percentage of pins and gateways that should remain unused during the current compilation. You can also specify device option bit and pin settings in the device configuration, which is used for many purposes. For example, you can set the read protection bit (Security bit) which is the global by default that prevents piracy of EPROM topology or EEPROM-based devices.

You can specify names and global settings that the compiler will use for parameters in all parameterized functions in your project.

You can enter global timing requirements for a project by defining general TPD_ delay time characteristics (input - output not registered), t_{CO} (clock signal - output), t_{SU} (clock signal - time setting), f_{MAX} (clock signal frequency). You can also disconnect all bidirectional feedback loops, preset (set 1) and clear (set 0) signal paths and other sync networks in the project. The Assigned/global project timing requirements command is used for this purpose (fig. 2.17)



Lecture I.13. *Fig. 2.17. AssignCommand window/General project time requirements*

Uncheck all Bidirectional Feedback terms Paths allow you to remove all feedback circuits for bidirectional contacts

Cut All check box. Clear and Set Paths that allows you to remove the connections between all reset and programmed circuits in the project.

Ignore task deadline check box while the Fitting checkbox allows you to run the router (Fitter) without taking into account the time constraints of the project. After removing this check box and setting the time parameters, the so-called controlled synthesis (driving time synthesis).

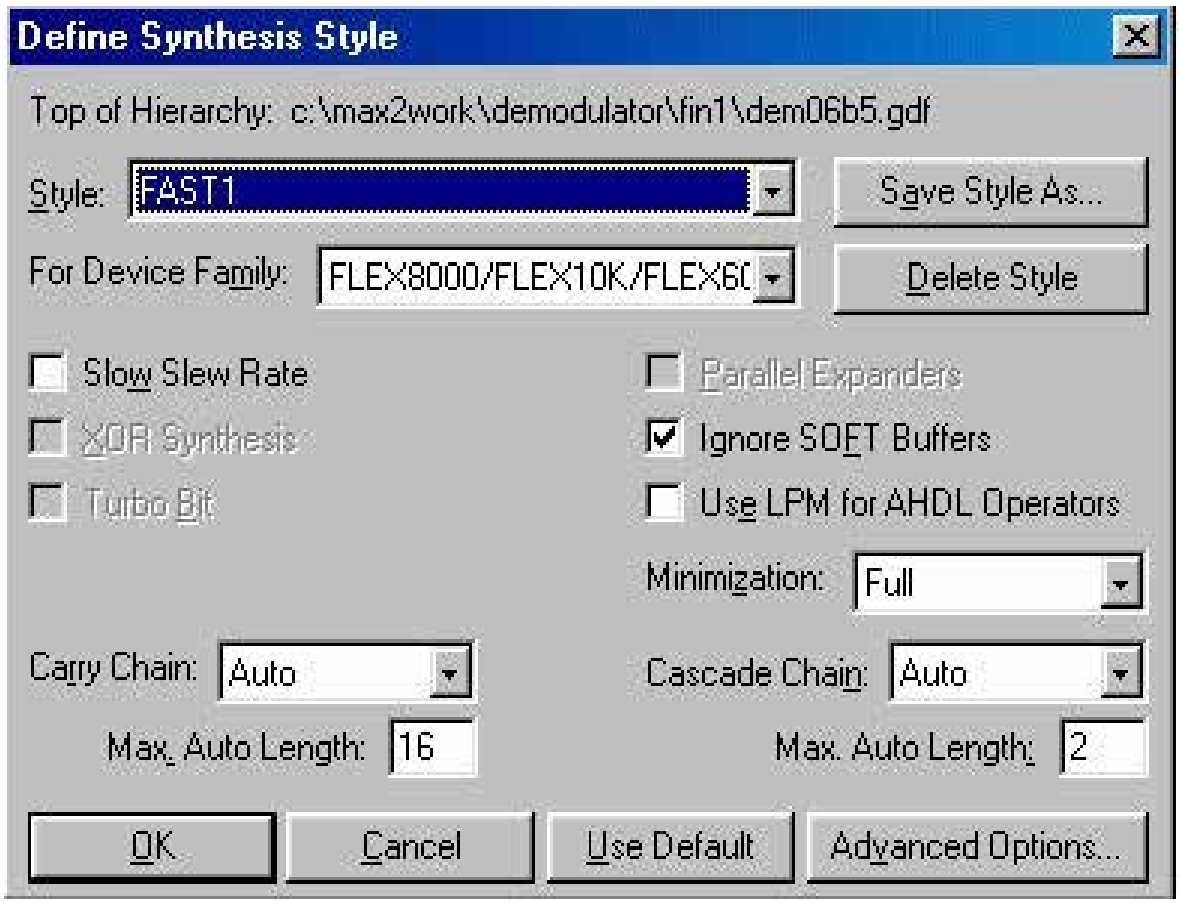
We noticed from our own experience that at the beginning, in order to speed up the compilation you should check this box, and then, after "licking" the project, remove it. Remember that controlled time synthesis is possible only for FLEX

devices, for MAX devices the delay time is predetermined, and in the case of assigning time parameters, only the parameters obtained during the synthesis are checked for compliance with the specified ones.

You can make global compiler settings for the logical synthesis of your project. You can set the default logic synthesis style, specify the degree of speed resource optimization, and tell the compiler to select automatic global control signals such as Clock , Clear (set to 0), Preset (set to 1), and Output Enable (allow output). You can also select standard or multi-level compiler synthesis mode, digital machine coding mode with 1 on power-up, and automatic register packing mode. You can also choose to automatically implement logic in high-speed input or output gates and I/O cells (Slew flow), open drain clamps (open drain contacts), and EAB memory cell blocks. To assign global logic synthesis parameters use the Assign / Global Project Logic Synthesis command (fig. 2.18)

Fig. 2.18. Assignment command window /global project logic synthesis.

The button Define Synthesis Style allows you to select more sophisticated parameters of the synthesis style (fig. 2.19).



Lecture I.14. *Fig.2.19. Definition of synthesis style*

As can be seen from figure 2.19, you can choose the name of the style, the method of implementation and the maximum length of the transmission and cascade chains, the degree of minimization of logical functions and other synthesis parameters.

The advanced Options button allows you to select synthesis options in the dialog box shown in figure 2.20.

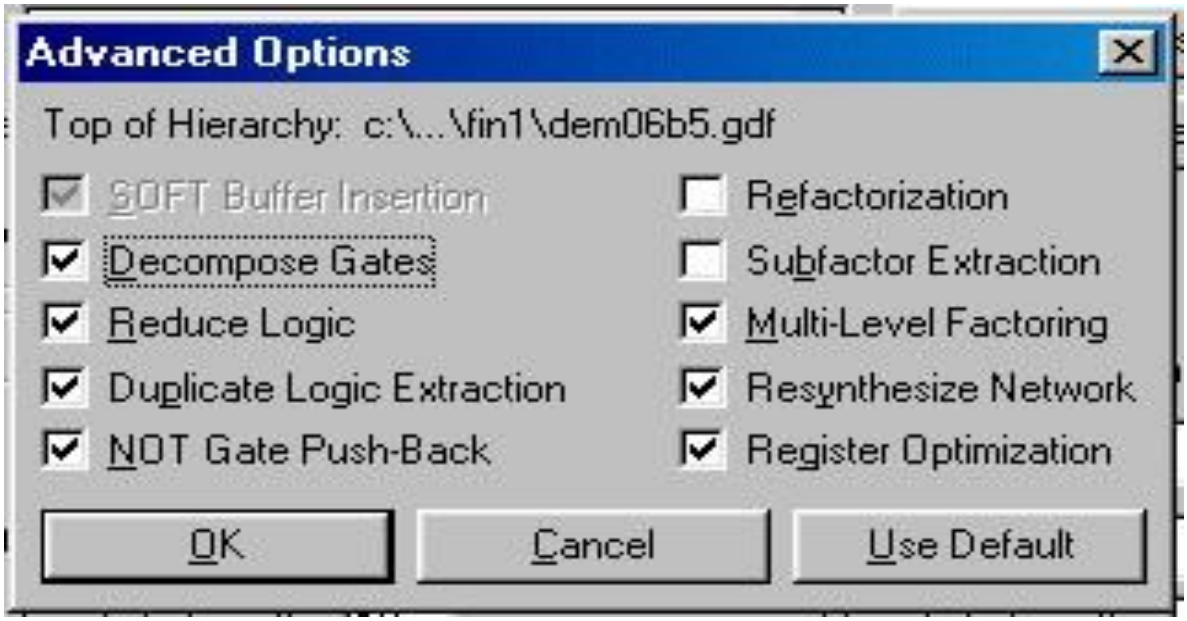


Fig. 2.20.Advanced Options window.

14.1 MAX PLUS II editors

All five MAX PLUS II editors and the three editors for creating project files (graphics, text, and signal) share functions such as saving and recalling a file. In addition, the MAX PLUS II Editor programs share the following features: Creating symbol files and function prototype files:

node search (node location);

traversal of a hierarchical tree (hierarchy transition);

context menu command menu); timing analysis

search and replacement of text fragments (Find & Replace Text);

canceling the last editing step, going back to it, cutting, copying,inserting and deleting selected fragments, exchanging fragments between MAX PLUS II programs.

Figure 2.21 shows the MAX PLUS II Graphic Editor window, which provides a project in the format of a real image (WYSIWIG). You can create new

files (command New from the File menu). Access to the graphic editor is carried out from the MAX PLUS II menu.

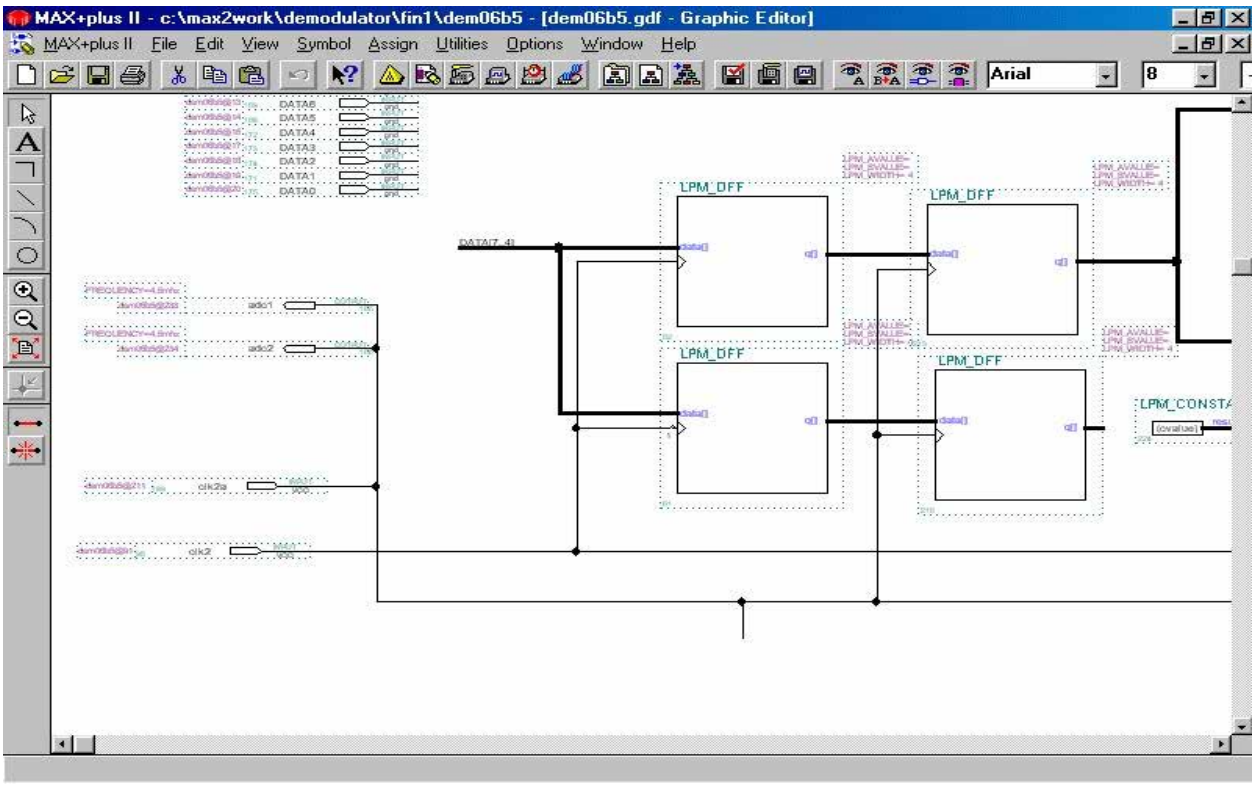


Fig. 2.21. Graphic editor MAX PLUS II

Design graphics files (.gdf) or OrCAD schematic files (.sch) created in this graphics editor can contain any combination of primitive symbols, megafunctions, and macrofunctions. Project files of any type (.gdf.sch.tdf.vhd.v.wdf.edf.xnf.adf.smf) can be the symbols.

The versatility of the graphic editor is characterized by the following features:

- the selection tool (an"arrow") facilitates project development. It allows you to move and copy objects as well as enter new symbols. When you place it on a pin or at the end of a line, it automatically turns into an orthogonal line drawing tool. If you click on the text, it will be converted automatically to the text editing tool;

- symbols are connected by signal lines called nodes or bus lines, which are several logically grouped nodes. By giving a name to a node you can link it to other nodes or symbols just by name. Buses are linked by name, but can also be linked graphically;
- the user can override and change the ports used in each instance of a mega or macro function symbol. At the same time, an inversion circle appears indicating the inverted port;
- you can select multiple objects in a rectangular area and edit them together or individually. When you move the highlighted area, the signal connections will be preserved;
- you can view sensor assignments, pins, locations, chips, clicks, timing, local routing, logic parameters, and parameter for each symbol. To simplify testingyou can also create contact groups that define the connection of external devices between contacts;
- altera, mega- and macro-functions reduce the time of project development. The users can also create their own feature libraries. When you edit a symbol or restore its default settings, you can automatically create selected examples or all examples of that symbol in a file in the graphics editor.

The graphic editor provides many other possibilities. For example, you can zoom in or out to see the entire project or any of its details. You can choose a typeface and font size, set line styles, and set and display guides. You can copy, cut, paste and delete selected fragments; get a mirror image (vertically or horizontally; rotate selected fragments by 90, 180 or 270 degrees; set the size and position of the current diagram sheet vertically or horizontally).

Figure 2.22 shows the MAX PLUS II Symbol Editor window which allows you to view, create, and edit a symbol that is a logic circuit. You can create new files (New command from the File menu). The symbol editor is called from the MAX PLUS II menu.

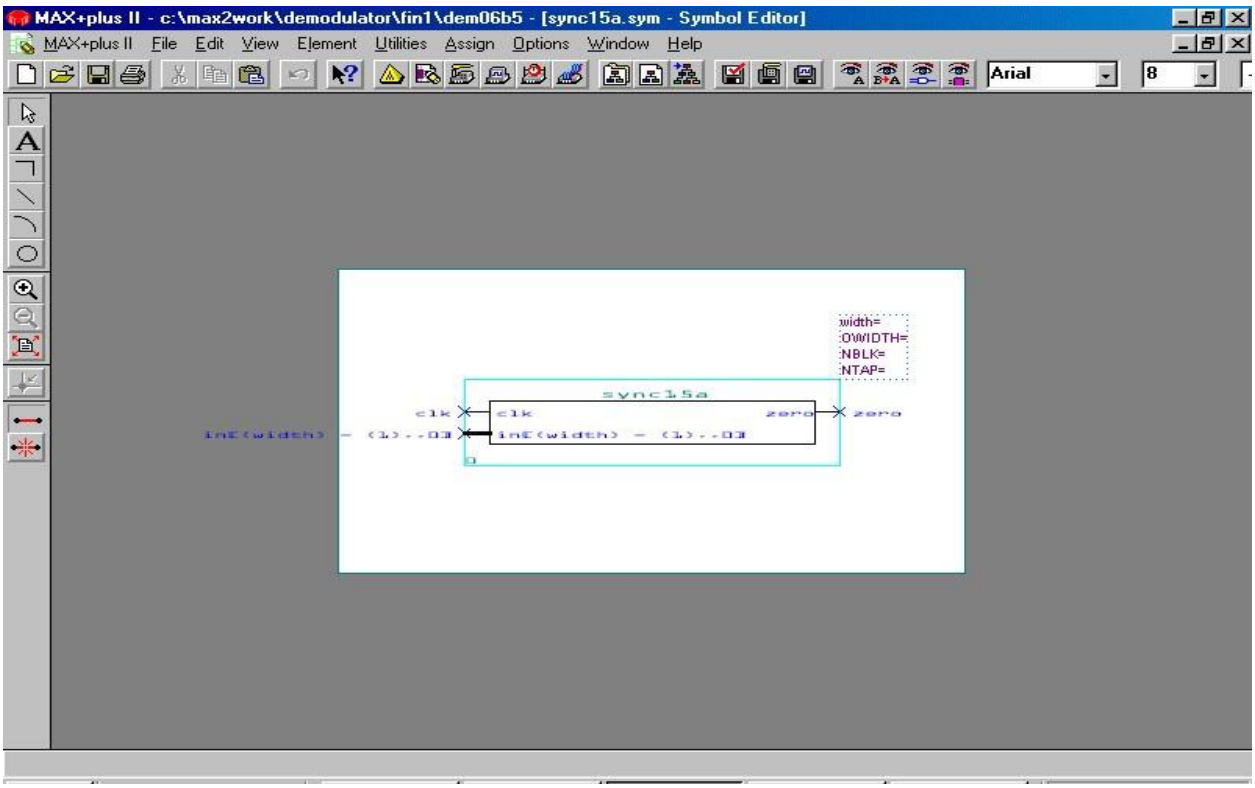


Fig. 2.22. MAX PLUS II symbol editor

The symbol file has the same name and the extension . sim . as the project file.Create command By default, the Symbol File menu available in the graphics, text, and signal editors creates a symbol for any project file. The symbol editor has the following features:

- you can replace the symbol representing the project file;

- you can create and edit pins and their names, expand input, output and bidirectional pins, as well as configure the parameters of the symbol input in the graphic editor file: with or without displaying the names of the pins on the screen, with the display of the full or abbreviated name. Therefore, the full name of the port and the name displayed in the file in the graphical editor window may differ;

- pin names are automatically duplicated across character boundaries. Only the names inside the symbol can be edited. The names outside the symbol are not editable, they simply illustrate how the contacts are connected;

- you can set parameter values and their default values;

- grid and guides help to align objectsaccurately;

- You can add comments or useful notes to the symbol, which will also appear after entering the symbol into the file in the graphic editor.

Figure 2.23 shows the MAX PLUS II text editor window, which is a flexible tool for creating text project files in hardware description languages: AHDL (.tdf), VHDL (.vhd), Verilog HDL (.v). This text editor can also work with any ASCII file. You can create new files (command New from the File menu). The symbol editor is called from the MAX PLUS II menu.

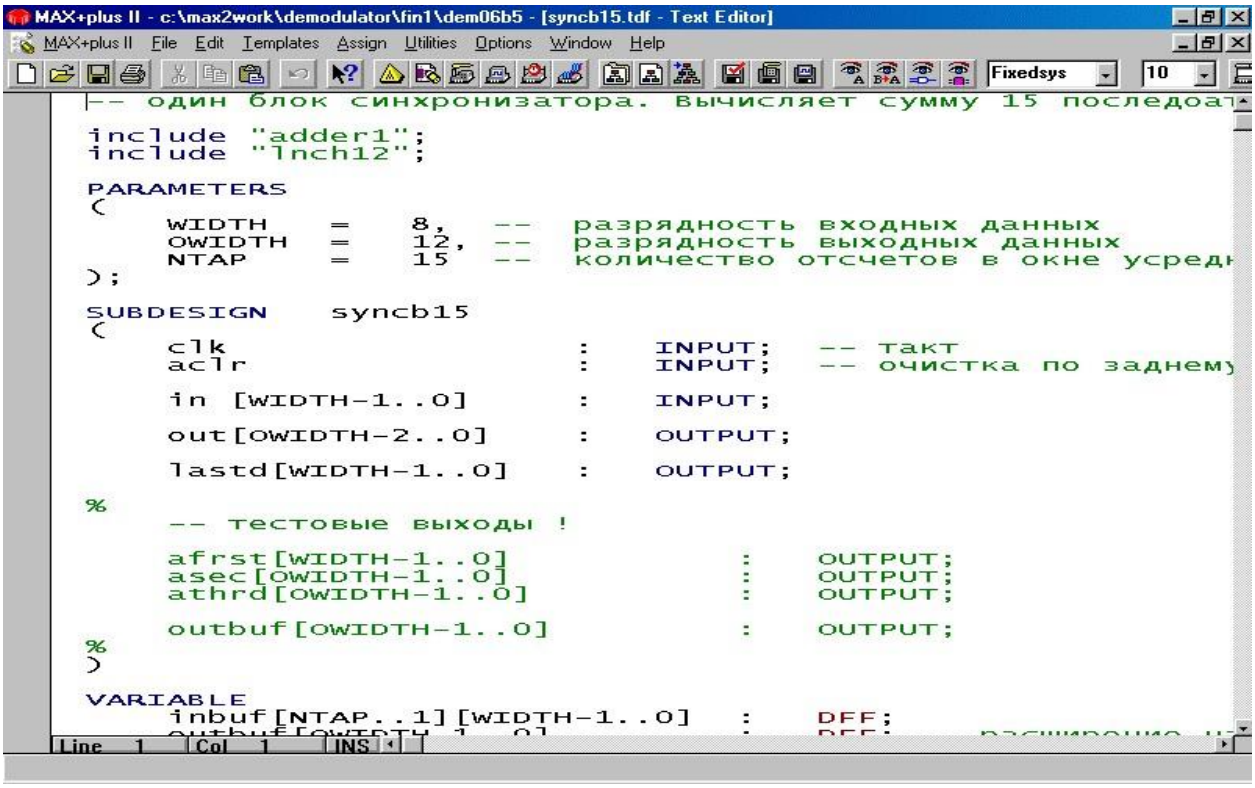


Fig. 2.23. Text editor MAX PLUS II

All of the listed project files can be created in any text editor, but this editor has built-in capabilities for convenient input of project files, their compilation and debugging, display of error messages and their location in the source text or in the text of additional files. In addition, language construction templates for AHDL, VHDL and Verilog HDL are available as well as syntactic construction coloring. You can manually edit assignment and configuration files (.acf) and enter configuration options for the compiler, simulator and timing analyzer inthis editor.

You can create test vectors (.vec) that are used for testing, debugging functions, and when entering signal designswith this text editor. You can also create batch files (.cmd for the simulator and .edc for EDIF) as well as a macro library (.lmf).

Text editor MAX PLUS II provides context-sensitive help.

The waveform editor (fig. 2.24) performs two roles: it serves as a tool for creating a project and as a tool for entering test vectors and viewing test results. The user can create project waveform files (.wdf) that contain the design logic of the project, as well as test channel (.scf) files that contain input vectors for functional testing and debugging A new file is created using the”New” command in the “File” menu. The Signal Editor is accessed from the MAX PLUS II menu.

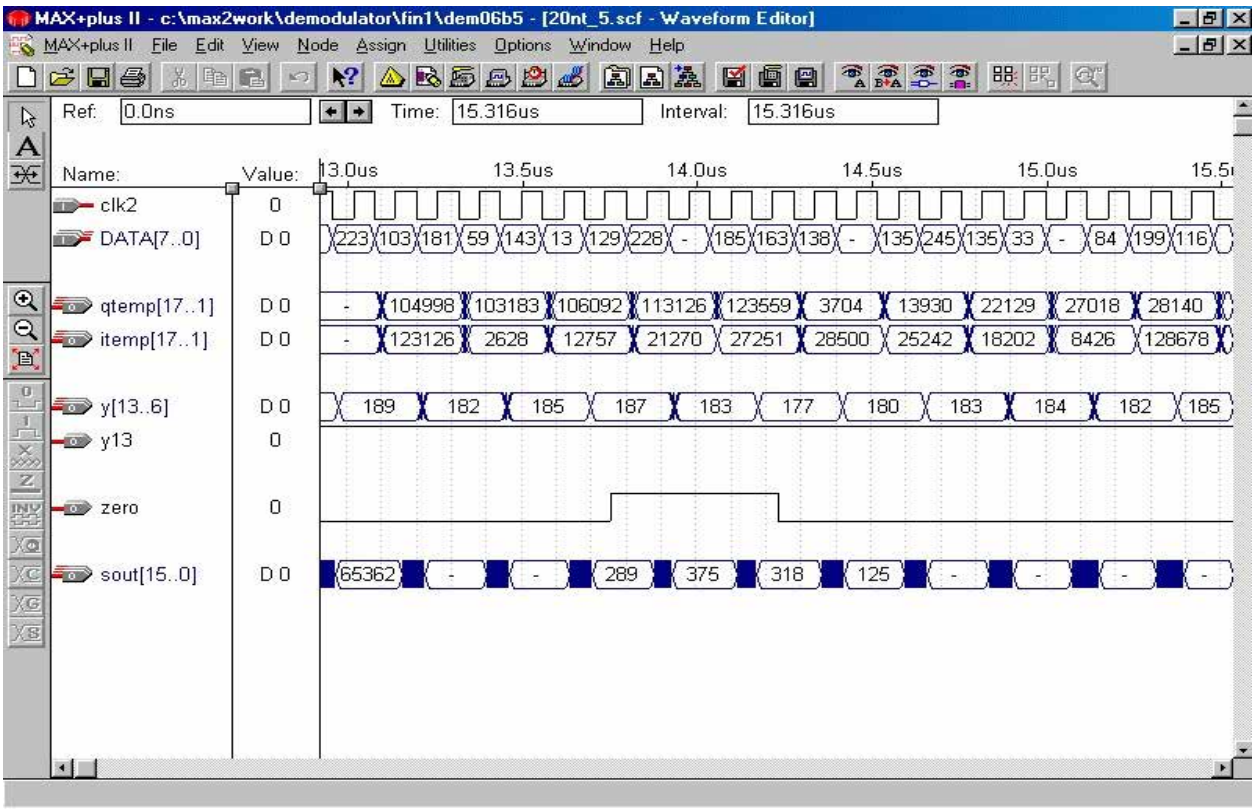


Fig. 2.24. MAX PLUS II signal editor

Designing a project in the signal editor is an alternative to creating a project in graphic or text editors.You can graphically specify the combinations of input logic levels and required outputs in it. The resulting file has the extension WDF (Waveform design file) and can contain both logic inputs and inputs of a digital machine, as well as combinational logic, a counter and outputs of a digital machine. You can also use "hidden" nodes to help define the results you need.

The signal editor design method is best suited for circuits with well-defined serial inputs and outputs, i.e. digital machines, counters and registers.

Using the signal editor, you can easily transform signals in whole or in part by creating and editing nodes and groups. You can create an ASCII character table file (.tbl) or import an ASCII test vector file (.vec) to create SCF test channel and WDF signal design filesusing simple commands. You can also save the WDF file as SCF for testing or convert SCF to WDF for using as a project file.

The signal editor has the following features:

- you can create or edit a node to get an I/O type (input/output) that represents an input or output pin or a block logic;

- when developing a WDF you can specify the type of logic that makes each node a contact and input, register, combinatorial or digital automaton;

- you can also specify the default value for the active logic level in the logic node: high (1), undefined (X) or high impedance (Z), as well as the name of the default state in the digital machine type of the node;

- simulator information file (.snf) to the SCF test channel file that exists for a fully compiled and optimized design;

- you can connect from 2 to 256 nodes to create a new group (bus) or ungroup nodes previously connected to a group. You can also merge groups with other groups. The group value can be displayed in binary, decimal, hexadecimal, or octal system with or without Gray code conversion;

- you can copy, paste, move or delete a selected part ("interval") of a waveform or the entire waveform as well as the entire node or group (i.e .the name of the node or group plus the waveform). You can edit multiple intervals, entire runs, entire nodes and groups in one operation. Copies of entire nodes and groups are linked so that editorial changes in one copy are reflected in all copies. You can also reverse, insert, rewrite, repeat, expand or compress a signal interval of any length, with any logic level, clock, count sequence, or a state name;

- it is possible to define and additionally display a grid for leveling transitions between logical levels before or after their creation;

- you can enter comments between passes anywhere in the file;

- you can change the display scale;

- to show the difference between the test outputs and the outputs of the real device, you can overlay any of the outputs in the current file or overlay a second

waveform editor file to compare the waveforms of its nodes and groups with those in the current file.

To debug DSP devices it is often necessary to test the algorithm on real or simulated signals. It is convenient to use a vector signal file for this.

Vector File (ASCII text format) is used to define input simulation conditions and nodes to be modeled. A vector file can also be used to create a wave design input project file. Let's take a closer look at the vector file format.

All sections used in Vector File are discussed below in the order in which they normally appear in the file. You can repeat any section to add additional input conditions.

14.2 Unit Section

It starts with the keyword UNIT , and then specifies the units of measurements in the file. The parameter is optional. Typical units are ns. Possible measurement units: ns, ms, s, mHz. The section ends with ; .

Example: UNIT ms ;

Start Section

It starts with the keyword START followed by an initial temporary value. The parameter is optional. The default value is zero. If units are not specified, they are taken from the Unit section. Thesection ends with ; .

Example: START 5ns;

14.3 Stop Section

It is similar to the Start section. By default, the time value of the last model vector is assumed.

Example: STOP 150ms;

Please note that Vector File must contain a multiple of Start-Stop Section representing time intervals. It is unacceptable to attribute different model vectors to the same time period.

14.4 Interval Section

The keyword INTERVAL is followed by a temporary value. Defines the vector input time interval. The parameter is optional. The default value is 1 ns. The section ends with ; . *Example*: INTERVAL 15ns;

14.5 Group Create Section

The keyword is GROUP CREATE . This section is not always needed for groups, buses, or state machines created in project source files. All nodes in a group must be of input/output type. In the File vector used for simulation, the nodes must be named according to the node names entered in the project file, including the hierarchical path ,if necessary. The section ends with ; .

Example: GROUP CREATE groupABC = nodeA node B nodeC ;

14.6 Radix Section

The keyword RADIX is followed by the numeric system designation. The parameter is optional. By default, the hexadecimal number system is used. There are four systems: BIN (binary), DEC (decimal), HEX (hexadecimal), OCT (octal). The section ends with ; .

Example: RADIX DEC;

14.7 Inputs Section

The keyword is INPUTS . Below there is a list of hostnames and/or groups. In vector File used for simulation node namesmust match the node names in the project file, including the hierarchical path if necessary.

Example: data bus INPUT clk OEN nodeA ;

The section ends with ; .

In the example below, when the next input section is used, the values from the previous section will be lost.

Example: INPUTS A1 A2;

START 0 ;

Przystań 25;

TEMPLATE % Model 1 section with inputs A1 and A2%

0 0 0

0 0 1;

START 26;

STOP 50;

TEMPLATE % section of model 2 with inputs A1 and A2%

0 10

1 0 0;

INPUTS A1 B1;

START 51;

HOLD 100;

TEMPLATE % Model 3 section with inputs A1 and B1%

1 1 0

0 1 1;

14.8 Outputs Section

The keyword is OUTPUTS. It is used to describe results. Analogy with input Section.

Example: RCO OUTPUTS QA QB QC;

Example: OUTPUTS A1 A2;

START 0 ;

Przystań 25;

TEMPLATE %Model 1 section with outputs A1 and A2%

0 0 0

0 0 1;

START 26;

STOP 50;

TEMPLATE %Section of model 2 with outputs A1 and A2%

0 10

1 0 0;

OUTPUTS A1 B1;

START 51;

HOLD 100;

TEMPLATE %Section of model 3 with outputs A1 and B1%

1 1 0

0 1 1;

14.9 Buried Section

It begins with the keyword POBURIED and used to describe nodes. Analogy with outputs Section .

Example: BURIEDnodeQA0 nodeQA1 nodeQB0 nodeQB1;

14.10 Pattern Section

It starts with the PATTERN keyword. This section uses data from the previous sections Inputs, Outputs and Buried Sections. Adding new inputs, outputs and buried Section deletes all previous data of this type, i.e. old data is replaced by new data (see example for entering data Section). The section ends with ; .

Vector File used to create a WDF file may contain additional sections Combinatorial Section , machine Section , registered Section . These sections are ignored if a Vector file is used for simulation.

Let us consider an example of creating a File vector for an eight-bit adder. AHDL description

```
sum8_.tdf

SUBPROJECT SUM8_

% 8-bit unsigned addition %

(

clk: INPUT;

a[7..0]:INPUT = GND;

b[7..0]:INPUT = GND;

sum[7..0]:OUTPUT;

cr:OUTPUT; ) VARIABLE
```

```
c[7..1]:NODE;

sum[7..0]:DFF;

cr: DFF; BEGIN

sum[7..0].clk=clk;

sum[7..0].prn=VCC;

sum[7..0].clrn=VCC;

cr.clk=clk;

cr.prn=VCC;

cr.clrn=VCC;


If (a[0]&b[0])==VCC then

sum[0]=GND;

c[1]=VCC;

ElsIf (a[0]#b[0])==GND then

sum[0]=GND;

c[1]=GND; Else sum[0]=VCC;

c[1]=GND;

End If;

FOR i IN 1 TO 6 GENERATE If (a[i]&b[i]&c[i])==VCC then

sum[i]=VCC;

c[i+1]=VCC;
```

```
ElsIf (a[i]#b[i]#c[i])==GND then
sum[i]=GND;
c[i+1]=GND;
ElsIf (((a[i]&b[i])#(a[i]&c[i])#(b[i]&c[i]))&!(a[i]&b[i]&c[i ]))==VCC then
sum[i]=GND;
c[i+1]=VCC; Else sum[i]=VCC;
c[i+1]=GND; End If;
END GENERATE;
If (a[7]&b[7]&c[7])==VCC then
sum[7]=VCC;
cr=VCC;
ElsIf (a[7]#b[7]#c[7])==GND then
sum[7]=GND;
cr=GND;
ElsIf   (((a[7]&b[7])#(a[7]&c[7])#(b[7]&c[7]))&!(a[7]&b[7]&c[7   ]))==VCC
then
sum[7]=GND;
cr=VCC; Else sum[7]=VCC;
cr=GND; End If;
END;
```

In this example 6 pairs of binary numbers will be added. In this case, the clocked meander clk (clock pulse for D triggers) will be set with a repetition period of 50 ns and a frequency of 2. The initial value of the pulse will be 0. A conversion from 0 ns to 500 ns will be performed.

When running the simulation, the corresponding sum8_.scf file must not be in the working directory, otherwise the simulator will default to using a file other than . vec and a file with the extension . scf. After the simulation process using . vec - a file with the extension . scf will be created automatically.

Figure 2.25 shows the FloorPlan Editor window, through which the user allocates resources to physical devices and views the results of branching and splicing performed by a compiler. The floor plan window opens after selecting the Floor Plan Editor option in the MAX PLUS II menu.

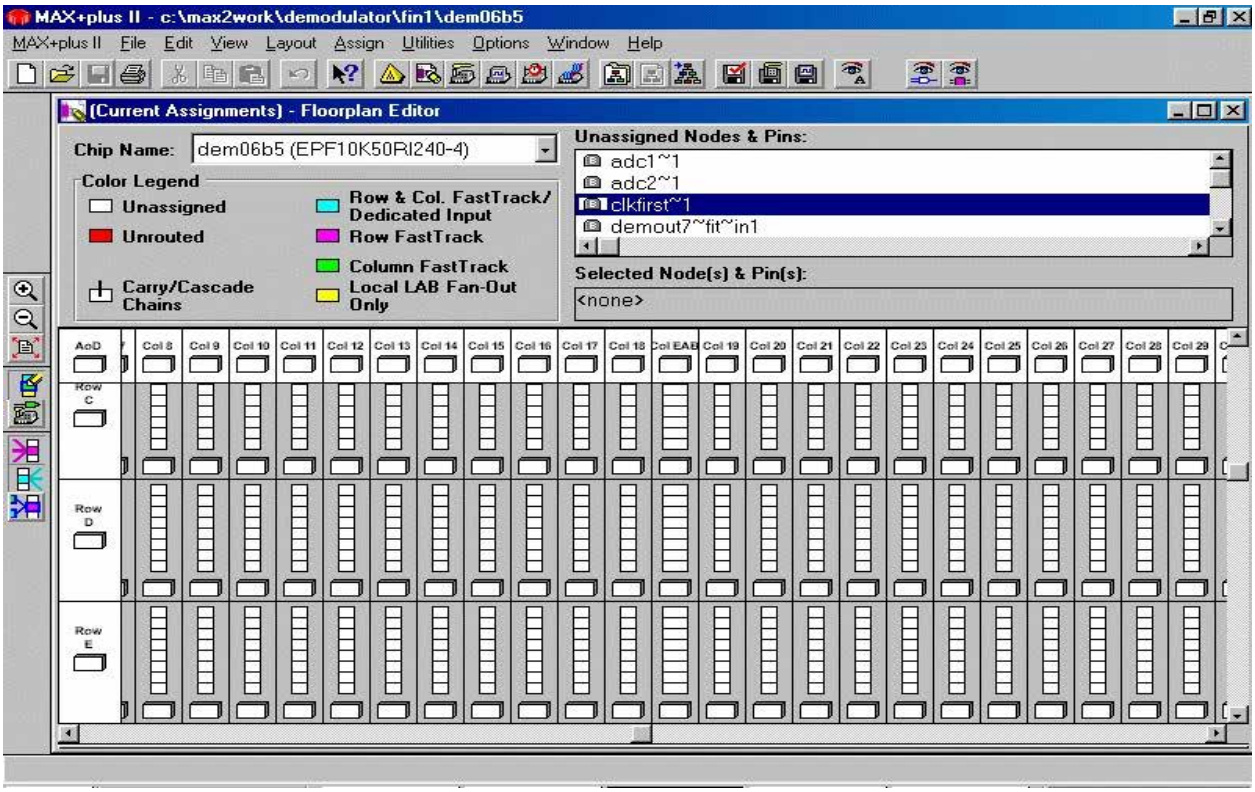


Fig. 2.25. MAX PLUS II FloorPlan Editor

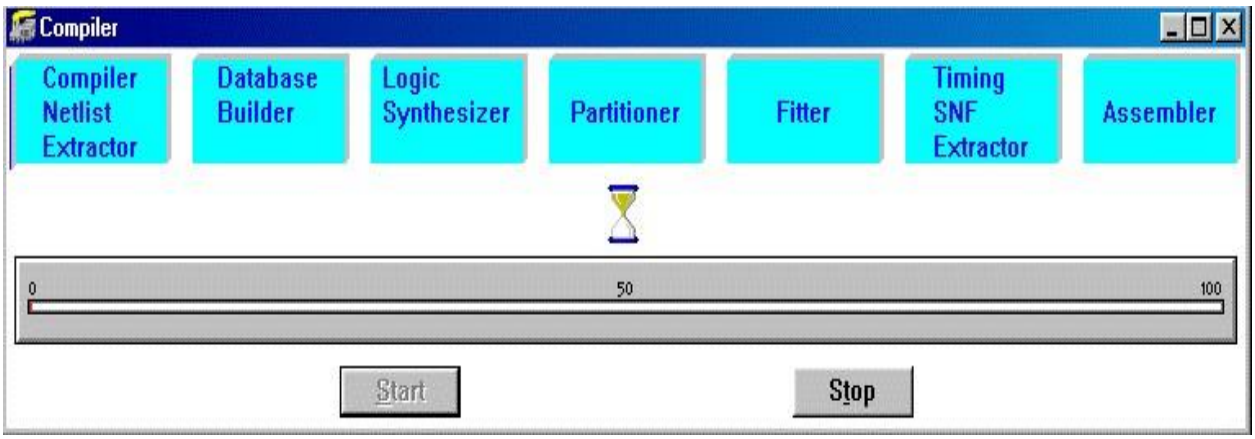
14.10.1 Project Construction process

First, the compiler extracts information about the hierarchical relationships between project files and checks the project for simple design input errors. It creates an organizational map of the project and then consolidates all the project files into a non-hierarchical database that can be processed efficiently.

The compiler uses a variety of techniques to improve the design efficiency and minimize the use of device resources. If a project is too large to implement on a single FPGA, the compiler can automatically split it up to implement on multiple devices in the same FPGA family, minimizing the number of connections between devices. The report file (.rpt) will show how the project will be implemented on one or more devices.

Although a compiler can automatically compile a project, it is possible to specify the processing of the project according to the programmer's precise instructions. For example, you can set a standard project logic synthesis style and other project-wide logic synthesis options. In addition, it is convenient to determine the time requirements for the entire project, to determine accurately the division of a large project into parts that will be implemented on several devices and to select device parameters that will apply to the entire project as a whole. You can also choose how many pins and gates to leave unused during the current building to reserve for future design changes.

You can start compilation from any MAX PLUS II program or a compiler window. The compiler automatically processes all input files of the current project. The compilation process can be observed in the compiler window (fig. 2.26):



Lecture I.15. *Fig. 2.26. Construction process*

- the rectangles of the compiler module are highlighted one by one as the compiler completes each processing step;

- an icon of the output file created by this module appears below the compiler module rectangle. To open the corresponding file, double-click the left mouse button on the icon and it will open;

- building completion speed gradually increases (up to 100%), which is also reflected in the growing rectangle"thermometer";

- stopping the compiler changes to the Stop / Show Status button, which you can select to open a dialog window that displays the current status of splitting and editing the project;

- if any errors or potential problems are detected during the build process, a message processing window is automatically opened, displaying a list of error messages, warnings and informational messages, as well as an immediate help to correct the error. Alternatively, you can define news sources in the project files or in its layer assignment plan.

The compiler can run in the background. You can minimize the compiler window while your project is processing and continue working on other project files.

A growing rectangular "thermometer" below the reduced compiler window icon allows you to monitor the progress of the build process while you focus on another task. However, remember that the luxury of multitasking is only possible on a decent machine. If you do not have much RAM, it's best to have a cup of tea while the compilation process is in progress.

The MAX PLUS II system compiler processes the project using the following modules and tools:

Extractor contains built-in EDIF, VHD;

Verilog and XNF;

Database Builder

Logic Synthesizer

Partitioner (separator);

Fitter (router);

Functional SNF Extractor;

Timing SNF Extractor (extractor for testing time parameters);

Linked SNF Extractor (extractor for layout testing);

EDIF Netlist Writer (a program for recording the output file in EDIF format);

Verilog Netlist Writer (a program for recording the output file in Verilog format);

VHDL Netlist Writer VHDL (program for recording the output file in VHDL format);

Assembler (assembler module);

Design Doctor Utility (project diagnostics utility).

The Netlist Extractor compiler module converts each project file into one or more binary files with the *cnf* extension. Because the compiler overwrites the values of all parameters used in parameterized functions, the contents of the CNF file may change during sequential compilation if the parameter values change. This module also creates a hierarchy file (.hif) (hierarchy connect to file) that documents the hierarchical relationships between project files and also contains the information needed to display the project's hierarchical tree in the Hierarchy Display window. In addition, this module creates a node database (.ndb) file that contains the project node names for the resource assignment database.

Built-in EDIF, VHDL, Verilog, and XNF format readers automatically translate project information into appropriate file formats. edf vhd , .v , . xnf to a format compatible with the MAX PLUS II system. EDIF readerprocesses input EDIF files using library files. lmf , (library display file) , which establish the compatibility of logic functions developed in other CAD systems with functions of the MAX PLUS II system. An XNF reader can create a text design export (.tdx) file (text design export file) that contains AHDL information equivalent to that contained in an XNF file (. xnf); this is done to edit the project in AHDL.

The database module Constructor (database developer) uses the HIF file to create compiler-generated CNF files containing the project description. Based on the hierarchical structure of the project, this module copies each CNF file into a single database without a hierarchical structure. In this way, the database supports the electrical communication of the project.

During database creation the module checks for logical completeness and design consistency, and checks for boundary consistency and syntax errors (such as a node with no source orassignment). Most errors are detected at this stage of compilation and can be easily and immediately fixed. Each compiler module sequentially processes and updates this database.

When the compiler processes a project for the first time, all project files are compiled. You can use the "smart recompile" function to create an extended project database that will help speed up subsequent builds. This database allows you to reassign physical device resources, such as pin and gate assignments, and recompile the design without rebuilding the database and re-synthesizing the design logic. Using the "full recompilation" functionyou can choose between recompiling only those files that have been edited since the last compilation or completely recompiling the entire project.

Logic synthesizer module uses a series of algorithms that reduce resource consumption and eliminate redundant logic, thus providing the most efficient use of the gate structure for the target device family architecture. This compiler module also applies logic synthesis techniques to user requirements for timing parameters, etc. In addition, the logic synthesizer looks for logic for unconnected nodes.If it finds an unconnected node, it removes the primitives associated with that node.

Three predefined styles and a large number of logic options are available to control logic synthesis.

In any MAX PLUS II program you can enter design time values and select logic parameters as well as define logic synthesis styles. You can set default global logic synthesis and project synchronization options for the entire project, as well as all additional logic options and project synchronization parameter assignments for individual logic functions.

If the project does not fit for installation on one device, Partitioner divides the database updated by the logic synthesizer into several FPGAs of the same family, simultaneously trying to divide the design into the minimum possible number of devices. The structure is divided by the boundaries of logical elements, and the number of pins used for communication between devices is minimized.

The division can be performed fully automatically, under partial or full control of the user. Device assignments and automatic device selection settings allow you to apply the level of control that best suits your project.

You can pause the build while the Partitioner and Fitter modules are running. The compiler will display information about the current state of the partitioning and routing processes, including a comparison of required and available resources. This is necessary in order to make a decision to continue construction or make fundamental changes to the project.

Using the database updated by the partitioner, the configuration program matches the design requirements with the known resources of one or more devices. It assignsthe location of the logic gate to each logic function that implements it, and selects the appropriate connection paths and pin assignments. This module attempts to negotiate resource allocation,i.e. Pins, Logic Gates, I/O Gates, Memory Cells, ICs, Clicks, Devices, Local Routing, Timing, and Pin Assignments from an Assignment and Configuration File (.acf). file) with available resources.

The module has options that allow you to define routing methods, for example, automatical introduction of logical elements or restrictions on the input connection ratio. If tracing fails, the module sends a message and gives you the option to ignore some or all assignments or stop the compilation.

Whether or not a full project trace has been completed, this module generates a report (.rpt) file that documents project split information, input and output contact names, project timings and unused resources for each device in the project. You can include sections in your report file showing user assignments, file hierarchies, logic gate connections, and equations implemented in LE.

The compiler automatically creates a trace (.fit) file that documents resource and device assignments throughout the project as well as trace information. Regardless of trace success,a user can view match, split, and trace information from the match file in the Layer Planner window. It is also possible to rewrite the

assignments from the negotiation file to the ACF assignments and configuration file for further editing.

You can tell the tracker (Fitter) to generate project output text files in AHDL format (.tdo). Since a multi-device project creates one file per device, you should split the project into multiple projects, each for a single device . If you want to capture logic on some devices, save the TDO file for that device as a text project file (.tdf) and recompile the logic for that device, keeping the logic synthesis results obtained from the previous compilation.

Functional Test Extractor SNFcreates a functional test file (.snf). The compiler generates this file before synthesizing the project, so it contains all nodes that are present in the project source files. This SNF file does not contain time information. Its generation is possible only if the project is compiled without errors.

The Timing SNF extractor creates (if theproject is compiled without errors) a timing file (.snf) that contains the project's timing data. This file is intended for testing and timing analysis. In addition, these SNF files also use compiler modules containing EDIF, Verilog, and VHDL entries that generate output files in these formats, as well as (optionally) standard delayed output (.sdo) files (standard delay format Output file) .

You can tell the compiler to generate an optimized SNF file using the Processing / Timing SNF Extractor command. Optimizing the SNF file increases compilation time, but helps save time on testing and analysis time.

The linked SNF (Layout Test Extractor) extractor creates a (.snf) file for layout testing when testing multiple designs (board level). This SNF file combines information from two types of SNF files: timing tests and functional tests that were created separately for these multiple projects. Complex projects can use devices from different families. If the build test file only contains timing information, it can also be used for timing analysis.

List of networks EDIF Writer (EDIF writing program). The MAX PLUS II compiler can work with most standard CAD programs that read standard EDIF 200 or 300 files. This (optional) compiler module containing an EDIF writer produces one or more EDIF output (.edo) files containing function information and (optionally) time parameters obtained after synthesis. Timing information can also be written to separate output files in the standard delay format (.sdo).

15.1*Verilog Network List Writer (program for writing Verilog*)

The Verilog writer creates output files with the extension . vo, containing information about functions and time parameters obtained after synthesis. Timing information can also be written to separate output files in the standard delay format (.sdo). List of VHDL Networks VHDL Recorder.

The VHDL Write Compiler add-on generates one or more output VHDL 1987 or 1993 (.vho) files containing the function and (optional) timing information from the synthesis. Timing information can also be written to separate output files in the standard delay format (.sdo).

Source files in hardware description languages can be used in design verification using an external simulator. These files are generated only after the project has been successfully compiled.

Assembler (assembly module)

The assembler module converts the gate, pin, and device assignments made by the Fitter module into a program image for the device(s) in the form of one or more binary programmer object files (.pof) or SRAM object files (.sof).).); For some devices the compiler also creates JEDEC ASCII files (.jed) containing information about a developer, ASCII configuration files (.ttf), and ASCII files in Intel format (.hex). The POF and SOF object files, as well as the JEDEC configuration files are then processed by the MAX PLUS II system programmer and Altera programming equipment (or other programmer). HEX and TTF files can be used to configure the

FLEX 6000, FLEX 8000 and FLEX 10K devices differently. The assembler module creates files for the programmer only after the project has successfully completed.

Once compiled, the MAX PLUS II System Compiler and Programmer allows you to generate additional device programming files that can be used in other programming environments. For example, you can create serial bitstream files (.bbs) and raw binary files (.rbf) to configure the FLEX 6000, FLEX 8000, and FLEX 10K devices. You can create sequential test vector files (.svf) or JAM language files (.jam) for device programming in automated test equipment such as ATE .

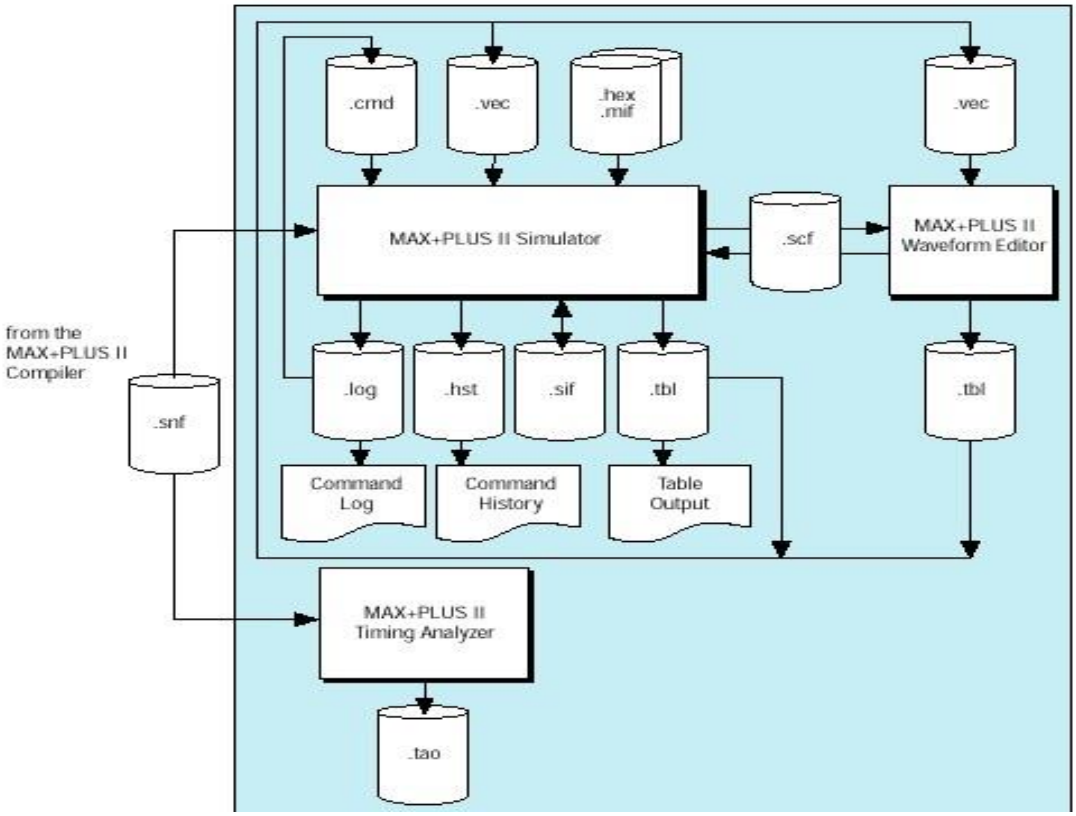
Design doctor Tool (project diagnosis tool). An additional project diagnostic tool examines the logic of each project file to identify elements that may cause system-level reliability issues. Typically, these problems are detected only after hardware loading of the device. You can choose from three predefined sets of project development rules with different levels. Alternatively, you can develop your own set of design principles.

The design rules are based on reliability principles that include logic containing asynchronous inputs, clocks, multi-level clock logic, preset and explicit configurations and competition conditions. Setting up validation rules is done using the Design Doctor Settings (Fig. 2.27)

Lecture I.16. *Fig.2.27. Command window for Design Doctor settings*

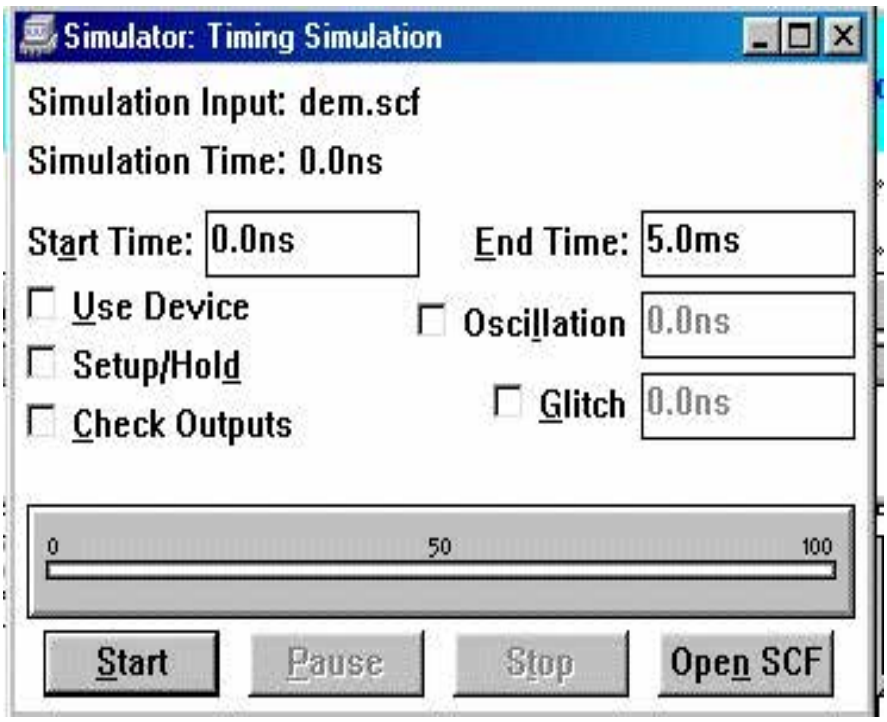
16.1 Project Verification

To check the design (see fig. 2.28), the MAX PLUS II system uses three programs: a simulator ,Timing Analyzer and a Waveform editor.



Lecture I.17. *Fig.2.28. Checking the project in the MAX PLUS II Simulator system*

The MAX PLUS II system simulator verifies the logic operations and internal timing of the design, allowing the user to simulate the design. The simulator can work in interactive or automatic (batch) mode. The simulator window is shown in fig. 2.29.



Lecture I.18. *Fig.2.29. MAX PLUS II simulator*

Before testing the project must be compiled with the specified compiler option to create a file (.snf) for functional testing, timing testing or multi-project (device) layout testing. The resulting SNF file for the current project is automatically loaded when you open the simulator.

For memory-related projects you can specify the initial memory contents in hexadecimal (Intel) files with the extension . hex or in memory initialization files with the extension . mif. The signal editor can automatically generate a default SCF file that the user can edit to obtain the required test input vectors. If an ASCII vector text file is used instead, the waveform editor will automatically generate an SCF test channel file from it.

The simulator allows you to check the output values obtained during the tests with the results contained in the SCF file (predicted values specified by the user or the results of previous tests). Using the appropriate programming equipment,

functional tests can also be performed to verify the actual output values of the programmable device based on the test results.

Using various simulator settings you can monitor your project for errors, as well as set violations and time delays. After the tests are completed, you can open the signal editor to view the updated SCF file or save the obtained output values to a table file with the extension . tbl , and then view the results in a text editor.

Functional tests. If a compiler is "tasked" to generate an SNF file for functional testing, it generates it before design synthesis. Thus, all nodes of the structure can be simulated during functional testing.

During functional testing, the simulator ignores all propagation delays. Thus, there is no delay in the SNF file for functional testing; the output logic levels change simultaneously with the input vectors.

Lecture I.19. *Testing time parameters*

The compiler generates an SNF file for testing the synchronization parameters after the project is fully synthesized and optimized. Therefore, this file contains only those nodes that were not destroyed in the process of logical synthesis.

The simulator obtains hardware information from this file loaded from device model files (.dmf) connected to the MAX PLUS II system.

If you split your project into multiple devices, the compiler creates an SNF file for the entire project and for each device. However, the time check is only performed for the project as a whole.

Synchronization testing can be accelerated by telling the compiler to generate an optimized SNF file containing dynamic models representing different types of combinatorial logic. This increases the compiler process time, but the resulting optimized SNF can reduce testing time because the simulator can work with dynamic models instead of interpreting all the logic in a combinatorial circuit.

When creating a SNF file for testing links across multiple projects, the compiler combines functional test SNF files and/or synchronization test files for multiple separate projects. Separate "subprojects" in the SNF assembly can be dedicated to devices of different families.In addition, since SNF files for functional testing are created before the full build is complete, subprojects can be introduced to represent logic unrealised in the Altera device.

The SNF chip can be used for board-level testing. Alternatively, if it only contains time information, it can be used to run the MAX PLUS II System Time Analyzer.

The simulator, together with other applications of the MAX PLUS II system, allows you to perform the following tasks:

- set the expected logic levels at the output, which can be compared with the test results;

- simulate individual nodes or nodes combined into groups. You can combine state machine bits in a project, model them as a group and refer to them by state name;

- determine the time period that represents the vibration or faults and analyze the design for both or one of these conditions.

- monitor the presence of violations of the initial registry settings and time delays in the project;

- record the actual output values of the device instead of simulated ones;

- perform functional tests. It can be verified that the simulated output values are functionally equivalent to the actual output of the device;

- set breakpoint conditions that cause the simulator to pause during the testing process;

- a list of names and logical levels of any combination of nodes and groups, as well as initialization of logical levels of a node or group before testing;

- initialize the contents of the RAM or ROM blocks before testing;

- store initialized node and group values, including initialized memory contents in the simulator initialization file (.sif) or reload initialized values stored in the file;

- write simulator commands to a test log text file (.log) in the same format as the command file (.cmd) used when testing in automatic(batch) mode and then use this log file to repeat this test cycle. Simulator commands and results can also be saved in a test history file with the extension . hst.

So, we considered the main methods of using the MAX+PLUS II package. Of course, within the framework of one chapter, it is almost impossible to consider in detail all the methods of working with such a complex and diverse software tool.However, aninterested user can master the package on his own, using this book and proprietary software.

Section III. PRINCIPLES OF HARDWARE DESCRIPTION IN AHDL LANGUAGE

General information about the special language

AHDL programming

AHDL is a low-level modular language fully integrated into the MAX+ PLUSII system. It is well suited for designing complex combinational logic, finite state machines, truth tables of parametric logic. To create text files of the project, you can use text editors of the MAX+ PLUSII system or any others ANDLtextDesignFiles (.tdf)). Then you can compile. tdf for output styles suitable for further simulation, timing analysis and device programming. In addition, the MAX + PLUSII system compiler allows you to create export text files (ANDL text Design Export Files) and output text files (Text Design Output Files (.tdo)), which can be saved as . tdf for reuse as project files.

You can create an entire hierarchical project using AHDL or mix. tdf with other file types into one project using any text editor to create . tdf , but only the text editor of the MAX + PLUSII system allows you to use it when entering, compiling and debugging the project.

AHDL can be easily inserted into the project hierarchy. In a text editor you can automatically create a symbol representing . tdf and place it in the graphic file of the project (Graphic Design File (.gdf)). Similarly, combine your own functions and over 300 mega- and macro-functions provided by Altera in any . tdf files. You can use the “Assign” or “Assign and Configure” .File (.acf) menu commands to create the resource and select the device, check the syntax and perform a full build to debug and run the project. Any errors are automatically detected by the MessageProcessor and highlighted in the text editor window.

A text editor is used to apply AHDL. This editor has built-in facilities for convenient input, compilation and debugging of project files with error messages and their location.

3.1. Elements of the AHDL language

Reserved keywords are used to control AHDL operators as well as defined GND and VCC constants. Reserved keywords differ from reserved identifiers in that keywords can be used as symbolic names when enclosed in single quotes ('), while reserved identifiers cannot. Both of them can be freely used in the comments.

Altera recommends to enter all keywords in capital letters to make them easier to read.

To get help with keywords, first make sure the TDF file is saved with a .tdf extension. Then open the file in a text editor window and press Shift+F1 and select a specific help button on the toolbar.

a list of reserved words:

AND	FUNCTION	OUTPUT
ASSERT	GENERATE	PARAMETERS
BEGIN	GND	REPORT
BIDIR	HELP_ID	RETURNS
BITS	IF	SEGMENTS
BURIED	INCLUDE	SEVERITY

CASE	INPUT	STATES
CLIQUE	IS	SUBDESIGN
CONNECTED_PINS	LOG2	TABLE
CONSTANT	MACHINE	THEN
DEFAULTS	MOD	TITLE
DEFINE	NAND	TO
DESIGN	NODE	TRI_STATE_NODE
DEVICE	NOR	VARIABLE
DIV	NOT	VCC
ELSE	OF	WHEN
ELSIF	OPTIONS	WITH
END	OR	XNOR
FOR	OTHERS	XOR

a list of reserved identifiers:

CARRY	JKFFE	SRFFE
CASCADE	JKFF	SRFF
CEIL	LATCH	TFFE
DFFE	LCELL	TFF
DDF	MCELL	TRI
EXP	MEMORY	USED

FLOOR	OPENDRN	WIRE
GLOBAL	SOFT	X

Symbols

A specific value in AHDL. This list contains symbols used as operators and comparators in logical expressions and as operators in arithmetic expressions.

Symbols	function
_(underline)	User-defined identifiers used as valid characters in character names
--(two dashes)	inline comment in VHDL style.
% (percent)	Restricts AHDL-style comments.
(parentheses)	Limit and define consistent bus names. For example: bus (a, b, c) consists of nodes a, b and c. SUBDESIGN sections and function prototype instructions. In addition, it restricts the input and output of truth tables in TruthTable statements. Contains the StateMachine and state declaration bits. Limit operations with the highest priority in logical and arithmetic expressions. Limit parameter definitions in Parameters statements, instance declarations, and parameter

	names in FunctionPrototype statements and embedded links. Also, restricting the condition in the Assert statement. Restrict evaluation function arguments in Define statements.
[] (brackets)	Check out the bus offer.
" ... " (quotes)	Limit symbolic names
" ... " (double quotes)	A header, parameters and validation statements. Restrict filenames in include statements. Limit numbers to non-decimal numbers.
. (point)	Separates symbolic names of logical function variables from port names. Separates extensions from filenames.
..(ellipse)	Separates the high bit from the low bit.
; (semicolon)	AHDL manuals and sections .
, (coma)	Separates symbolic names from types in declarations.
= (equal to)	Assigns the default values of GND and VCC to the inputs in the Subdesign section. Assigns values to options in the Option statement. Assigns default values to parameters in the Parameters statement or embedded link.

	Assigns values to the states of the finite state machine. Assigns values to logical equations. Connects a signal to a port by a link that uses a connection by port name.
= >(arrow)	Separates input from output in TruthTable statements. Separates WHEN clauses from logical expressions in CASE statements.
+ (plus)	The addition operator
- (minus)	The subtraction operator
== (two equal signs)	Line or number equivalence operator
! (exclamation mark)	operator number
!= (exclamation mark equals)	Inequality operator
➢ (more than)	There is something else to compare
>= (more than or equal to)	Greater than or equal to comparator
<(less)	The comparison is less than
<= (less than or equal to)	The comparator is less than or equal to
&(ampersanth)	Operator AND
!&(Ambient Apersant)	Operator AND-NOT

$ (dollar sign)	The exclusive operator is OR
!$ (dollar with an exclamation mark)	The only operator is OR-NOT
# (pound sign)	the OR operator
!# (exclamation mark)	the OR operator
? (question)	Ternary operator. It uses the following format: <expression 1>?<expression 2>:<expression 3> If the first expression is not equal to zero (true), then the second expression is evaluated and the result is returned to the triple expression. Otherwise, the value of the third expression is returned.

Line and symbolic names

There are three types of names. Symbolic names are user identifiers. They are used to declare the following parts of the TDF:

- internal and external nodes and buses;
- permanent
- machines changing states state bits and state names
- copies
- memory segments
- evaluation functions
- named operators

Subproject names are user-defined names for lower-level project files. The name of the subproject must match the name of the TDF file.

Port names are symbolic names that identify the inputs or outputs of a logic function.

The compiler generates names containing the tilde character (~) that may appear in the project's FitFile . If reverse annotation is used, these names also appear in the project's ACF file. The tilde character is reserved for compiler-generated names only and can be used in custom pin, node, and bus names.

There are two types of representation available for the three name types: quoted and unquoted.Line names are enclosed in single quotes ('), but character names are not.

When creating a tokenized representation of a TDF file containing lines of port names, quotes are not included in the pinstub symbol.

Buses

Symbolic names and ports of the same type can be declared as buses in logical expressions and equations.

A bus, which can contain up to 256 elements (or bits) is treated as a collection of nodes and acts as a whole.Constants GND and VCC can be duplicated to form a bus.

Buses can be reported in three ways:

A bus name consists of a symbolic name or port followed by a subrange specification enclosed in parentheses, i.e. [4..1]. The name and the longest number in the range can contain up to 32 characters. An example,

The name q[MAX..0] is valid if the constant MAX is described above in the Constant statement.

When a bus is defined, brackets [] are a cylindrical method of describing the entire range. example,

i[4..1] can be specified as [ ].

b[6..0][3..2] can be specified as b[][].

A bus name consists of a symbolic or port name followed by a subrange specification enclosed in parentheses, i.e. d[6..0][2..0]. The name and the longest number in the range can contain up to 32 characters. A single node on a bus can be named by name [y] [z] or by name *y_x ,* where *yix a*re numbers in the bus domain.

A bus sequence name consists of a comma-separated list of symbolic names, ports, or numbers enclosed in parentheses, such as (a, b, c).

example,

The input ports of the *variable* DFFreg can be written as *reg.(d, clk, clrn, prn).*

Below are two sets of examples that show two buses described by different methods:

b[5..0]

(b5,b4,b3,b2,b1,b0)

B[]

b[log2(256)..1+2-1]

b[2^8..3mod1]

b[2*8..div2]

Ranges in bus names can consist of numbers or arithmetic expressions separated by colons (..) and enclosed in square brackets [] . example,

bus [4..1] with elements *a 4, a 3, a 2 and a 1* .

d[B"10".. B"00"] bus with *d2, d1 and d0* .

b[2*2..2-1] bus with *b4, b3, b2 and b1* . *Range* delimiters are arithmetic expressions.q[MAX..0] is a valid bus if the MAX constant is described in the *Constant* statement.

c[MIN(a, b)..0] is an acceptable bus if the evaluated function MIN is described in the *Define* statement.

t[WIDTH-1..0] is an acceptable bus if the WIDTH parameter is described in the *Parameters s*tatement.

Regardless of whether the range separator is a number or an arithmetic expression, the compiler separates and interprets the separators as decimal (integer) values.

Subranges contain a subset of the nodes defined in the declared directions and only in the directions to the left of the logical equation or replaced link. An example:

if you declare a bus *c[*5..1], you can use the following subranges of that bus:

to [3..1], to [4..2], to 4, to [5], (to 2, , to 4)

A comma is used in the subrange *(c2, , c4*) to save the space not allocated to a bus member.

Ranges are usually listed in descending order. To specify the ranges in a different order, specify the VIT0 parameter with the *Options* statement so that the compiler does not display warning messages. For dual-range buses, this applies to bothranges.

Numbers in AHDL

You can use any combination of decimal, binary, octal, and hexadecimal numbers. The syntax for each base is shown below.

base:	Value:
Decimal	<string of numbers from 0 to 9>
binary	B"<line 0, 1 and X>"(X= "any state")
Eight years	O”<string of digits from 0 to 7>” or Q"<string of numbers from 0 to 7>"
sixteen	X"<string of numbers from 0 to 9, from A to F>"
	H"<string of numbers from 0 to 9, from A to F>"

The following rules apply to numbers:

● The MAX+ PLUSII compiler always interprets numbers in logical expressions as groups of binary digits, numbers in bus ranges as decimal values;

● in logical equations, numbers cannot be assigned to individual nodes, GND and VCC are used instead.

Arithmetic expressions

Operator/comparator	example	description	Priority
+(unary)	+1	Positive	1
-(unary)	-1	Negative	1
!	!I	HI	1
^	a^2	Degree	1

MOD	4MOD2	Module	2
DEPARTMENT	4DIV2	Distribution	2
*	a*2	Multiplication	2
LOG2	LOG2(4-3)	Logarithm to base 2	2
+	1+1	Addition	3
-	1-1	Subtraction	3
== (numerical)	5 == 5	Numerical equality	4
== (term)	"a" == "b"	Equality of conditions	4
!=	5!=4	Not equal	4
>	5>4	More than	4
>=	5>=5	greater than or equal to	4
<	a<b+2	Less than	4
<=	a<=b+2	Less than or equal to	4
&	a&b	I	5
I	a I б		
!&	1!&0	NAND	5

NAND	!NAND0		
$	1$1	XOR	6
XOR	1XOR1		
!$	1! 1 долар	XNOR	6
XNOR	1XNOR1		
#	a#b	OR	7
OR	a або b		
!#	a !# b	OR	7
OR	HI b		
?	(5<4)?3:4	Three-seater	8

Arithmetic expressions can be used to define computed functions in *Define* statements, constants in *Constant* statements, parameter values in *Parameters* statements, and as bus range delimiters.

These expressions use arithmetic operators and comparators to perform basic arithmetic and comparison operations on the numbers they contain. Arithmetic expressions use the following operators and comparators:

1. Arithmetic expressions must give non-negative numbers.

2. If the result of LOG2 is not an integer, it is automatically rounded up to the next integer. For example, LOG 2(257)=9.

The arithmetic operators supported in arithmetic expressions are a superset of the arithmetic operators supported in logical expressions.

Logical expressions

Logical expressions consist of operands separated by logical operators, arithmetic operators and comparators, and additionally grouped by parentheses. Expressions are used in logical equations as well as in other statements such as *Case and IfThen.*

A logical expression can have one of the following forms:

• operand

For example *a, b* [5..1], 7, VCC;

• A reference to a logical function with a wildcard

For example, out[15..0] = 16dmux(q[3..0]);

• A prefix operator (! or -) applied to a logical expression

For example, *!c;*

• For logical expressions separated by a binary operator

For example, *d 1 $ d 3* ;

• A logical expression enclosed in parentheses

For example *(!foo & bar*).

You can name logical operators and comparators in AHDL files to help you enter resource assignments and interpret the equation section of the report file.

Boolean operators

The following logical operators can be used in logical expressions:

Operator	example	description
!	! be	Appendix to 1
NO	Not you	

&	bread butter	AND
AND	bread and butter	
!&	a[3..1] !& b[5..3]	NO
NAND	a[3..1] NAND b[5..3]	
#	trick #treatment	OR
OR	trick or treat	
!#	c[8..5] !# d[7..4]	NO
OR	c[8..5] NOR d[7..4]	
$	foo $bar	Does not include OR
XOR	foo strip XOR	
!$	x2 !$ x4	Exclusive OR NOT
XNOR	x2XHI x4	

Each operator is a two-input logic gate, except for the NOT(!) operator, which is a single-node inversion prefix. At the same time, you can use a name or a symbol to represent a logical operator.

Expressions using these operators are interpreted differently depending on whether the operands are individual nodes, buses, or numbers.

Allow the compiler to replace AND operators and all compramers in boolean expressions with the lpm_add_sub and lpm_Compare *functions*, including the boolean option UseLPMforAHDLOperators.

Logical operators using NOT

The NOT operator is an inverter prefix. The behavior of the NOT operator depends on the operand it affects.

Three types of operands can be used with the NOT operator:

• if the argument is one node, GND or VCC, one inversion is performed, for example, !a means that the signal goes through the inverter;

• if the operand is a group of nodes, then each element of this group passes through the inverter, for example, bus !a[4..1] is interpreted as (!a4, !a3, !a2, !a1);

• if the operand is a number, it is interpreted as a binary number and each bit is inverted. For example, !9 is interpreted as !B"1001", which is B "0110".

Logical operators using AND, NAND, OR, NOR, XOR, and XNOR.

There are five combinations of operands with binary operators. Each of these combinations is interpreted differently:

• if both operands are separate nodes or constants GND and VCC, the operator performs a logical operation on two elements, for example (*a$ b*) ;

• if both operands are groups of nodes, the operator acts on the corresponding nodes of each group, performing bitwise operations between the groups, the groups must be of the same size, for example *(a , b, c) # (d, e, f)* is interpreted as (*a#d , b, #e , c , #f*) .

• if one operand is a single node, GND, or VCC, and another group of nodes, single node, or constant is duplicated to form a group of the same size as the second operator, then the expression is clocked as a group operation, for example, a &b [4 ..1] is interpreted as *(a&b4, a&b3, a&b2, a&b1*).

• if both operands are numbers, the shorter number is sign-extended to match the magnitude of the second number, and then interpreted as a group operation, for example, in the expression (3 # 8) 3 and 8 are converted to binary numbers "0011 " and "1000" respectively" , the result will be B "1011".

• if one operand is a number and the other is a node or a group of nodes, then the number is broken into bits according to the size of the group and the expression is treated as a group operation, for example, in the expression (a, b, c) & 1 , 1 is converted to B " 001", and the expression becomes (a, b, c) & (0, 0, 1), the result will be (*a& 0, b& 0, c& 1).*

An expression using VCC as an argument is interpreted depending on an expression using 1 as an operand. For example, in the first expression, 1 is a signed extended number. In the second expression, node VCC is duplicated. Each expression is then interpreted as a group operation.

(a , b , c) and 1 = (0, 0, c)

(a, b, c) and VCC = (a, b, c)

Arithmetic operators in Boolean expressions

Arithmetic operators are used to perform addition and subtraction arithmetic operations on numbers and strings in Boolean expressions. They use the following operators:

Operator	example	description
+(unary)	+1	Plus
-(unary)	- a[4..1]	Minus
+	number[7..0] + delta[7..0]	Addition

-	right_x[] – lcftmost_x[]	Subtraction

The following rules may apply to binary operators:

• operations are performed between two operands, which must be strings or numbers;

• if both operands are buses, they must be of the same size;

• if both operands are numbers, the shorter number is expanded to the size of the second operand;

• if one operand is a number and the other node is a group of nodes, then the number is truncated or expanded to fit the size of the operands. If any significant bits are discarded, the MAX+ PLUSII compiler will display an error message.

By adding two rails to the right of the boolean equation using the + operator, you can put a 0 to the left of the group to increase the width of the rail. This method adds an extra bit of data to the left side of the equation that can be used as the output data for the carry. For example, the counter [7..0] and delta [7..0] buses are padded with zeros to provide information about the counter signal:

(number , response [7..0]) = (0, number [7..0]) + (0, delta [7..0])

Comparators

Two types of comparators are used to compare individual nodes or buses: logical and arithmetic. The following comparators can be used in logical expressions.

comparator	example	description
==(boolean value)	address[19..4] ==B”B800”	Is equal to

!=(Boolean)	b 1 != b3	Not equal
< (arithmetic)	glory[] < power[]	Less than
<=(arithmetic)	money[] <= power[]	Less than or equal to
>(arithmetic)	love[] > money[]	More than
>=(arithmetic)	delta[] >= 0	greater than or equal to

Logical comparators can compare individual nodes, buses, and numbers without undefined values (X). When comparing tires or numbers, the tires must be the same size. The MAX+ PLUSII compiler performs a bitwise bus comparison, returning VCC when the comparison is true and GND when the comparison is false.

Arithmetic comparators can only compare tires and numbers; tires must be the same size. The compiler performs an unsigned comparison of bus values, meaning that each bus is interpreted as a positive binary number and compared to another bus.

Priorities of Boolean operators and comparators

Operands separated by logical, arithmetic, and comparison operators are evaluated according to the following precedence rules (priority 1 being the highest). Operations with the same priority are evaluated from left to right. You can use parentheses() to change the order of calculations.

Priority:	Operator/comparator:
1	- (minus)
1	! (NO)

2	+ (addition)
2	- (subtraction)
3	== (equal to)
3	!= (not equal)
3	< (less)
3	<= (less than or equal to)
3	> (more than)
3	>= (greater than or equal to)
4	and I)
4	!& (I HI)
5	$ (without OR)
5	! $ (exclusively OR NOT)
6	# (OR)
6	! # (OR-NOT)

3.3. Ports

A port is an input or output of a logic function. The port can be located in two places:

• the input or output port of the current file appears in the subproject section;

• a port that is the input or output of an instance of a primitive programming file or a lower-level programming file used in a logic section.

Ports of the current file

The input or output port of the current file is displayed in the "Subproject" section in the following format:

<port name>:<port type>[= <default>]

The following port types are available

INPUT MACHINE INPUT

OUTPUTMACHINEOUTPUT

BIDIR

You can import and export state machines between TDF and other project files by describing the input and output data as MACHINEINPUT or MACHINEOUTPUT in the Subproject section. The function prototypes that represent the file must specify which ports belong to the state machine. MACHINEINPUT and MACHINEOUTPUT can only be used in lower-level files in the project hierarchy.

Instance ports

A port that is the input or output of a boolean function instance.

In the example below, the trigger D is displayed as a regular variable in the "reg under Variable" section and then used in the Logic section.

```
Variable

reg: DFF;

BEGIN

reg.clk = clk

reg.d = d

out = reg.q

END;
```

All functions offered by Altera have defined port names (pinstubs) specified in the function prototype. The most common primitive port names are listed in the table below:

Description of the port name

. q trigger or latching output;

. d enter trigger or latch data;

. t trigger input T ;

. j J input JK trigger;

. k input K trigger JK ;

. s SR trigger configuration input;

. r SR release reset input;

. clock input trigger clk;

. ena input permission synchronization trigger, limit state machine permission to start;

. prn active preset low flip-flop input

. clrn active low bright flip-flop input;

. reset input of the active high automatic reset;

. oe permission to enter, exit, TRI primitive;

on the main input of primitives CARRY, CASCADE, EXP, TRI, OPNDRN, SOFT, GLOBAL, and LCELL;

. output of TRI, OPNDRN, SOFT, GLOBAL, and LCELL primitives;

3.4. Project structure

This section describes the project structure in AHDL. AHDL sections and operators are described in the order in which they appear in the text design file (TDF-TextDesignFile):

• operator;

• operator;

• Enable the operator;

• constant operator;

• Define the operator;

• prototype operator;

• operator;

• confirmation operator;

• section;

• section;

• logical section;

The Title operator

The Title operator allows you to enter a comment to the text file of the project, which will be placed in the report file (ReportFile) created by the compiler. The following example illustrates the use of the Title operator:

TITLE" DisplayController";

The following rules must be followed when using the title operator:

• The Title statement begins with the keyword TITLE followed by a text string enclosed in quotation marks. The operator ends with ";";

• if the Title statement begins in a project text file, the used title is placed at the beginning of the ReportFile. In the example above, the DisplayController title is placed in the report file;

• the title can contain 255 characters, in addition, it cannot contain line breaks and row breaks. To include quotation marks in a title, use pairs of double quotes, for example:

TITLE """EMP5130""DisplayController";

• no more than one title operator;

• the Title statement must be placed outside other AHDL sections.

Parameters operator

The *Parameters* operator allows you to define one or more parameters that control an instance of a parametric mega- or macro-function. The following example illustrates the use of *the Parameters* operator:

```
PARAMETERS
(
FILENAME ="myfile.mif", -- default to "=" optional.
WIDTH,
AD_WIDTH = 8,
NUMWORD = 2^AD_WIDTH
);
```

The following rules must be followed when using the *Parameters* operator:

• the *Parameters* statement begins with the PARAMETERS keyword, followed by a list of one or more parameters and optional default values, the entire list enclosed in parentheses;

• parameters in the list are separated by commas, parameter names are separated from optional default values by a symbol (=), in the above example only the WIDTH parameter has no defined value;

• parameter names can be defined by the user or by *Altera*;

• parameter values can be text strings enclosed in quotation marks, in that case, if the parameter values are not enclosed in quotation marks, the compiler will attempt to interpret them as arithmetic expressions, if this fails they will be interpreted as strings;

• Operator *parameters* end with a character (;);

• after defining the parameter, it can be used in the entire text file of the project;

• the parameter can be used only after its definition;

• parameter names must be unique;

• the name of the parameter should not contain spaces, use the underscore symbol to separate words and better understanding;

• the *Parameters* operator can be used any number of times in one text file of the project;

• Operator *parameters* must be outside of other AHDL sections;

• parameters used to define other parameters must be defined in advance;

• the use of circular references is prohibited, the following example illustrates the use of an invalid circular reference:

```
PARAMETERS
(
FOO = BAR,
BAR = FOO;
);
```

When compiling a project text file, the compiler looks for parameter values in the following order:

• an instance of a logical function is analyzed, for example, in a project text file, in an object (instance) created using an object declaration (instance declaration) or an inline reference, you can specify the parameters to be used, and optionally specify their values . In the project graphics file, you can select a symbol and use the *Edit Ports/Parameters* command in the Symbol menu to assign parameter values to that object;

• the analysis of the instance of a higher-level logical function is performed, the values of the parameters of the instance of the higher-level logical function are extended to the subfunctions of this logical function, if the instances of these logical subfunctions do not have their own values of these parameters;

• the analysis of the standard global values of the project parameters defined by *the Global Project Parameters* command from the *Assign* menu is performed, these values are stored in the configuration file (*Assig nmen t & Configuration file* - .asi) of the project;

• additional default values specified in the "*Parameters*" section of the project's text file (TDF) or via the PARAM primitive in the project's graphic file describing the logic function. These default values apply only to the file in which they are specified, and do not apply to subprojects already in that project.

Include operator

The Include statement allows you to import text from a file with an .inc extension into the current file. The following example illustrates the use of the Include operator:

INCLUDE "const.inc";

The Include operator has the following characteristics:

• the Include statement begins with the keyword INCLUDE, followed by the name of the .inc file to be included, enclosed in double quotes;

• if you do not specify the extension of the linked file, » the compiler assumes that the file has the default extension .inc;

• the Include operator ends with (,);

• at the compilation stage, the Include operator is replaced with information from the inc file, in the above example, the const.inc file replaces the INCLUDE text "çonst.inc".

The Include statement is often used to combine function prototypes for files that are lower in the hierarchy than a given project text file. To use mega- and macro-functions, it is necessary to first define the logic of their operation in the corresponding project file. Then use the *Function Prototype* statement to define the function ports. Alternatively, you can use the Include statement to include the function prototype stored in the corresponding .inc files. You can then create an object declaration (instance declaration) or an inline reference to a boolean function instance.

contains a function prototype for the current project file using the *Create Standard Include File command from the File menu.*

At the stage of compiling the text file of the project, the compiler looks for files with the extension .inc in the following order:

• first the search in the directory of this project is performed;

• user libraries defined by the User Libraries command in the *Parameters* menu;

• *\maxp lus2\max2 lib\mega_ lpm and \maxp lus2\max2 inc directories created during installation.*

Save and check in the project from the"File menu", or perform a full recompile of the project to restore the project hierarchy tree displayed in the Project Hierarchy Display window.

The following rules must be followed when using the Include operator:

• in a running program file names are context-sensitive, in the MAX+PLUS II documentation file names can be entered in both uppercase and lowercase letters, however, if the Include statement is used, the file names must exactly repeat their

original names, the names of *Altera* macros and megafunctions are only with lowercase letters;

• The *include* statement must be outside of other AHDL sections;

• *Include* statements can be used any number of times in a single project text file.

Files with the .inc extension can only contain the following instructions:

• Function Prototype;

• Define;

• Parameters;

• Constant.

Constant operator

The "Constant" operator allows you to enter the name of a number or an arithmetic expression. The following examples demonstrate the use of the Constant operator:

CONSTANT UPPER_LIMIT =130;

CONSTANT BAR = 1 + 2 DIV 3 + LOG2(256);

CONSTANT FOO= 1;

CONSTANT FOO_PLUS_ONE = FOO + 1

The *constant o*perator has the following characteristics:

• the *Constant* operator begins with the keyword CONSTANT, followed by a symbolic name, then the symbol (=), and finally a number (including its base, if necessary) or an arithmetic expression;

• the *Constant* operator ends with (;);

• After. the way the constant is defined allows it to be used throughout the project's text file, in the above example in the *Logic* section, the UPPER_LIMIT constant can be used to represent the decimal number 130;

• constants can be defined using arithmetic expressions, these expressions can contain predefined constants;

• the compiler evaluates the arithmetic expressions used in the *Constant* operator and simplifies them to numerical values without creating logic circuits.

The following rules must be followed when using the Constant operator:

• the constant can be used only after its definition;

• names of constants must be unique;

• the name of the constant cannot contain spaces, to separate words in the name of the constant, use the underscore character;

• the *Constant* operator can be used any number of times within one text file of the project;

• the *Constant* operator must be outside other sections of the AHDL language;

• the constants used to define other constants must be defined in advance;

• the use of circular links is prohibited, the following example illustrates the use of impermissible circular links:

• CONSTANT FOO = BAR;

• CONSTANT BAR = FOO;

Define operator

The *Define* statement allows you to define an evaluated function, which is a mathematical function that returns values calculated from optional input arguments.

The following example describes a MAX evaluation function that determines the presence of at least one port in a subproject partition:

```
DEFINE MAX(a,b) = (a > b) 7 a : b;

SUBDESIGN

(

dataa[MAX(WIDTH,0)..0]: INPUT;

datab[MAX(WIDTH.0)..0]: OUTPUT;

)

BEGIN datab[] = dataaf];

END;
```

The *Define* operator has the following characteristics:

• The *Define* statement begins with the keyword DEFINE, followed by a symbolic name and a list of one or more arguments enclosed in parentheses;

• arguments are separated by commas, the symbol (=) separates the list of arguments from the arithmetic expression;

• the operator ends with (;);

• once defined evaluation function can later be used in the entire text file of the project;

• Predefined evaluation functions can be used to define an evaluation function, for example, the following evaluation function MIN_ARRAY_BOUND is calculated from the value of the evaluation function MAX:

```
DEFINE MAX(a,b) = (a > b) ? a : b;
```

DEFINE MIN_ARRAY_BOUND (x) = MAX(0, x) + 1;

The following rules must be followed when using the *Define* operator:

• the evaluation function can be used only after its definition; .

• names of evaluation functions must be unique;

• names of evaluation functions should not contain spaces, to separate words in the name of the evaluation function and improve its perception, use underscores;

• the *Define* operator can be used any number of times in one text file of the project;

• the *Define* statement must be placed outside other AHDL sections.

Function Prototype operator

Prototype operators perform the same function as symbols in graphic design files. Both include a brief description of the function, its name, and input, output, and bidirectional ports. State machine ports can be used for import and export functions for state machines.

The mega- and macro-function input ports do not have default values as in the MAX+PLUS II graphics editor files. Therefore, input values for unused ports must be specified explicitly. Also, in the *Subdesign* section, you can specify default values for bidirectional ports.

Before creating a mega- or macro-function object, make sure that a corresponding project file exists that describes its logical operation. The function *prototype operator* is then used to describe the function ports and create an instance of the boolean function by declaring the object or reference to be replaced.

The function prototype operator has the following characteristics:

• the keyword FUNCTION is followed by the name of the function;

• the list of input ports follows the function name;

• keyword WIDTH, and the list of parameters is followed by a list in parentheses, names separated by commas;

• The RETURNS keyword

• when importing and exporting state machines, the *function prototype statement used by the file must use the machine port* (specified with the MACHINE keyword) to specify which inputs and outputs are state machines, for example:

FUNCTION ss_def (clock, reset, count)

RETURNS (MACHINE ss_out);

• the function prototype operator ends with (;);

• the *function prototype statement* must be outside other AHDL sections and precede the instance of the logical function created by the superseded object or reference declaration.

For a primitive instance, it is also worth using the mechanism of declaring a replaced object or reference. However, unlike mega and macro functions, the primitive logic is defined, and there is no need to define the primitive logic in a separate project file. Also, there is no need to use the *function prototype operator* unless you want to change the primitive's port traversal order.

The following example illustrates a function prototype that exists by default for JKFF primitives.

FUNCTION JKFF (j, k, elk, clrn, pm)

RETURNS (q);

This example shows a modified function prototype for the JKFF primitive:

FUNCTION JKFF (k, j, elk, clrn, pm)

RETURNS (q);

An alternative to using the Function Prototype statement in the project file is to use the Include statement to combine the .inc files that contain the function prototypes you use. Additionally, MAX+PLUS II has a *Create Default Include File* command on the File menu that automatically creates a file with an .inc extension that contains the function prototype for the current project file.

macro functions are stored in files with the extension .inc in the \maxplus2 \ m ax2lib\ mega_lpm and \maxplus2\max2inc directories, respectively.

Options operator

The *Options* operator is intended to provide a LiT 0 option value that indicates, relative to a group, whether the bit with the lowest number is the Most Significant Bit (MSB), the Least Significant Bit (LSB), or a weight index that depends on the position of that bit with the description of the group. Using this option avoids generating warning messages if the least-numbered bit in the group is not used as the least significant bit, which is typical. When describing a group of dimensions defined by a range of numbers, the left number of the represented range is always the index of the most significant bit; accordingly, the correct number of the represented range is always the index of the least weighted bit. If the specified range of numbers is presented in ascending order and the parameter B IT0=M8B is not set, a warning will be generated. If BIT 0= MSB is selected and the specified range is in descending order, a warning message will also be generated. If the option BIT0= ANY, it is possible to define group sizes by ranges of numbers presented in both ascending and descending order without generating warnings.

The *Options s*tatement begins with the OPTIONS keyword, followed by the VIT0 option and its setting. The option operator ends with (;);

The following example illustrates the use of the *Options* operator:

OPTIONSBIT0 = MSB;

In the given example, the bit with the lowest number in the group is defined as the most significant bit (MSB). Other possible options LSB - the smallest; weight, and any is the weight depending on the position of the least-numbered bit in the group description.

The Options statement at the beginning of a text project file causes the bit order settings in the groups to apply to the entire project file. If the current project file is a top-level project file, the settings in the Options statement apply to all subprojects contained in that top-level project. If the current project file is not a top-level project file, then the action of the *Set Options* statement applies only to that project file.

Assert operator

The *Assert operator* allows you to check the validity of arbitration expressions using parameters, numbers, evaluation functions, as well as port statuses (if the port is in use or not).

The following example illustrates the use of the *Assert s*tatement:

ASSERT (WIDTH>0)

REPORT "Width (%) positive integer" WIDTH

SEVERITY ERROR

HELP_ID INTVALUE;

The ASSERT keyword is followed by an arithmetic expression enclosed in parentheses. When the expression evaluates to false, the message line after the

REPORT keyword is sent to the text editor. In the absence of a conditional expression, the message string is displayed unconditionally.

The REPORT keyword is followed by a message string and additional parameters represented by variables. The message string is enclosed in quotation marks and may contain % characters, which are replaced by the values of the corresponding variables. If the REPORT keyword is not used and the value of the arbitration expression is false, the following message appears:

<validity>: line <1 and number >, file <filename>: Assertion failed

Optional variables included in the message consist of one or more parameters, evaluation functions, or arithmetic expressions. The variables contained in the message are separated by commas. The values of the variables are substituted in the order of occurrence of % characters in the message. In the example above, the value of the WIDTH variable replaces the % character in the message string.

The optional keyword SEVERITY is followed by a severity level of ERROR, WARNING, or INFO. The default severity level is ERROR.

The HELP_ID key and the *hint string* are supported by some *Altera* features and are reserved for Altera's internal use.

The Assert statement ends with a character (;).

Assert statements can be used in a *Logic* section or outside of other AHDL sections.

Subdesign operator

The *Subdesign section* defines the input, output, and bidirectional ports of this project.

The following example illustrates the use of the Subproject section:

SUBDESIGN top

foo, bar, clk1, clk2 : INPUT =VCC;

aO,al,a2,a3,a4 : OUTPUT;

b[7 .0) : BIDIR

The *Subdesign* section has the following characteristics:

• the SUBDESIGN keyword is followed by the name of the project, the name of the subproject must match the name of the text file of the project, in this example it is called top;

• list of signals in round brackets;

signals are represented by symbolic names indicating their type (for example, INPUT);

signal names are separated by commas. The name is followed by a colon, then the signal type and the symbol (;);

• possible port types IN, OUT, BIDIR, MACHINE IN or MACHINE OUT. In the above example, signals *foo, bar , clk1 and clk2 and signals a0, al, a2, aZ and a4 are outputs. Bus b [7..0] is bidirectional;*

• the MACHINE INPUT and MACHINE OUTPUT keywords are used to import and export state machines between project text files or other project files, but the MACHINE INPUT and MACHINE OUTPUT port types cannot be used in top-level project text files;

• after specifying the port type, you can optionally specify a default value of GND or VCC (otherwise default values are not accepted), in the above example VCC is the default value for input signals unless they are used in higher-level files hierarchies (assignments made in the file hierarchy have higher priority)

In the top-level file, the INPUT, OUTPUT, or BIDIR ports are device outputs. In lower-level files, all port types are entry and exit points for that file, but not for the device as a whole.

Variable section

The *variables* section is used to describe and/or generate the variables used in the *logic* section. AHDL variables are similar to variables used in high-level languages and define internal logic.

The following example illustrates the use of *the Variable* section:

```
VARIABLE
a, b, c : NODE;
temp : halfadd;
ts_node: TRI STATE_NODE;
IF DEVICE_FAMILY = "FLEX8000" GENERATE
adder : flex_adder;
d, e : NODE,
ELSE GENERATE
cadder : pterm_adder;
  f, g : NODE;
END GENERATE;
```

The *variables* section can contain the following operators and constructs:

• description of objects;

• description of nodes; .

• description of registers;

• description of finite automata;

• description of pseudo-names of finite automata.

If Generate statements , which can be used to create objects, nodes, registers, state machines, and state machine aliases.

The *variables s*ection has the following functions:

• the section begins with the keyword VARIABLE;

• user-defined character variable names are separated by commas, and acceptable variable types are NODE, TRI STATE_NODE, <primitive>, <megafunction>, <macrofunctiton> or <state machine declaration> in the above example, internal variables *a, b* and *c* are of type NODE; temp is an instance of the halfadd macro function, and ts_node is a TRI_STATE_NODE type object;

• each variable definition line ends with a sign (;).

In a file with the extension . suitable for the current project may contain compiler-reserved names that include a tilde (~). The tilde character is reserved for compiler-generated names only; it is prohibited to use to indicate outputs, nodes and groups (buses).

Case operator

The *Case* operator specifies a list of alternatives that can be activated depending on the value of the variable, group, or expression following the Case keyword.

The following example illustrates the use of the Case operator:

```
CASE f[].q IS

WHEN H"00" =>

addr[] =0;

 s = a & b;

WHEN H"01" =>

cont[].d = cont[].q+1;

WHEN H"02", H"03", H"04" =>

t[3..0].d = addr[4..1];

WHEN OTHERS => f[].d = f[].q;

END CASE;
```

Case Operator has the following functions:

• logical expressions, a group or a state machine are placed between the keywords CASE and IS in the above example it is f[].q;

• the Case operator ends with the keywords END CASE, followed by the symbol (;);

• the content of the Case statement is a list of one or more unique alternatives resulting from the WHEN keyword, each alternative being preceded by the WHEN keyword;

• each alternative is one or more constant values separated by commas and the symbol (=>);

• if the value of the logical expression after the CASE keyword matches any alternative in the option, then all operators after the corresponding symbol (*>) are activated, in the example above, if f[].q is h"01 ", then the expression count[ ].d = number t[].q+1;

• if the value of the logical expression after the CASE keyword is not equal to any of the alternatives, then the alternative option after the WHEN OTHERS keywords in the above example will be activated if the value of f[].q is not equal to H "00", H "0l" or H "CF", then the expression f[].d = f[].q is activated;

• the Defaults operator defines the default values in cases where the WHEN OTHERS keywords are not used;

• if the Case statement was used in defining the transitions of the state machine, then you cannot use the WHEN OTHERS keywords to escape from invalid states, if the states of the state machine are defined using n-dimensional code and the machine has 2n states, then it is acceptable to use the WHEN keywords OTHERS; :

• each alternative variable must end with a character (;).

Defaults operator

The *Defaults* statement allows you to define default values that are used in truth tables, as well as in If then and Case statements. Since active high signals are automatically assumed to be GND, the default operator is only necessary when using active low signals.

The default values assigned to variables should not be confused with the default values assigned to ports in the Subdesign section.

The following example shows the use of the Defaults operator;

```
BEGIN

DEFAULTS

A = VCC;

END DEFAULTS;
```

```
IF y& z THEN
a = GND; % low %
END IF; END;
```

The Defaults operator has the following characteristics:

• default values are placed between the DEFAULTS and END DEFAULTS keywords. The operator ends with a symbol (;);

• the content of the Defaults statement consists of one or more logical expressions assigned to constants or variables, in the above example, the variable a is assigned the default value VCC;

• the default operator is activated if any variable from the list of default operators is found to be undefined in any of the operators, in the above example, the variable a is undefined if y and z have the logical zero value; this activates the expression (a = VCC) in the Default statement.

The following rules must be followed when using the default operator:

• in the *Logic* section, it is allowed to use no more than one Default operator, and when using it, it must be located directly after the BEGIN keyword;

• if several assignments are made to the same variable in the default operator, all assignments except the first are ignored;

• The Default operator cannot be used to assign the value of X to (any) variable.

If Then statement

The *If Then* statement contains a list of statements that will be executed if the logical expression between the IF and THEN keywords evaluates to true.

An example of the IfThen operator:

```
IFa [] == b [] THEN

C[8..1] = H"77";

Addr[3..l] = f[3..l].q;

f[].d = addr[] +1;

ELSIF g3 $ g4 THEN

T[].d = addr[];

ELSE

d = VCC;

ENDIF;
```

The *If Then* operator has the following characteristics:

• between the IF and THEN keywords there is a logical expression, depending on the value of which the list of operators after the THEN keyword will be executed or not, each operator in this list ends with a symbol (;);

• between the keywords ELSEIF and THEN there is an additional logical expression, and after the keyword THEN — a list of operators that are executed depending on the value of the Boolean expression. These additional keywords and operators can be repeated multiple times;

• operators after the THEN keyword are activated if the corresponding logical expression is true, the following ELSEIF THEN constructs are ignored;

• the ELSEIF keyword followed by one or more statements has the same meaning as the WHEN OTHERS keyword in the Case statement, if none of the boolean expressions is true, then the statements following the ELSE keyword are executed,

as in the example shown above, if none of the logical expressions is true, d = VCC is executed, using the ELSE keyword is optional;

• the values of the logical expressions following the keywords IF and ELSEIF are calculated sequentially;

• The IF Then statement ends with the ENDIF keywords followed by the (;) character.

The If then statement can generate logic circuits that are too complex for the compiler. If the if then statement contains complex Boolean expressions, then including the inverse of each of those expressions will likely result in even more complex Boolean expressions. For example, if a and b are compound expressions, the conversion of these expressions can be even more complicated.

If operator:	Interpretation by the compiler:
IF a THEN	IF a THEN
c = d;	c = d;
END IF;	
ELSIF bTHEN	IF !a& !b THEN
c = e;	c = e;
END IF;	
ELSE	IF !a& !bT HEN
c = f;	c = f;
END IF;	END IF; .

Unlike the If Then statement, which can only evaluate the values of Boolean expressions, the IfGenerate statements can evaluate the values of sets of arithmetic expressions. The main difference between the IfThen and IfGenerate operators is that in the first case, the value of a boolean expression is calculated by hardware, and in the second case, the value of a set of arithmetic expressions is calculated at the compilation stage.

If Generate statement

The *If Generate statement* contains a list of statements that are activated when an arithmetic expression evaluates to a positive value.

The If Generate statement has the following characteristics:

• between the Generate keywords is an arithmetic expression whose value can be evaluated, the GENERATE keyword displays a list of operators, each of which ends with a root (;), the operators are activated if the arithmetic expression evaluates to a true value;

• the ELSE GENERATE keywords are accompanied by one or more operators that are activated if the arithmetic expression evaluates to a false value;

• the If Generate operator ends with the keywords END GENERATE, followed by the symbol (;);

• The If Generate statement can be used in the *Logic and Variable* section

Unlike IfThen statements, which can only evaluate the values of Boolean expressions, IfGenerate statements can evaluate the values of sets of arithmetic expressions. The main difference between If then and If Generate statements is that in the former case, the value of the Boolean expression is evaluated in hardware, while in the latter case, the value of the set of arithmetic expressions is evaluated at compile time.

The If Generate statement is especially often used with For Generate statements to handle special situations, such as the least significant bit in a multistage multiplier. This statement can also be used to test parameter values, as shown in the last example.

For Generate operator

The following example illustrates the use of the For Generate iterative statement:

```
CONSTANT NUM_OF_ADDERS = 8;

SUBDESIGN gent

 (

A[NUM_OF_ADDERS..1], b[NUM_OF ADDERS..1];

cin : INPUT;

c(NUM_OF ADDERS.. l], cout : OUTPUT;

)

VARIABLE

carry_out[(NUM_OF_ADDERS+l)..l]: NODE;
```

```
BEGIN

carry_out[1] = cin;

For and IN 1 TO NUM_OF_ADDERS GENERATE

c[i] = a[i] $ b[i] $ carry_out[1]

carry_out[1] [i+l] = a[i] & b[ij"# carry out[i] & (a[i] $ b[i]);

END GENERATE;

cout = carry_out (NUM_OF_ADDERS+l ]);

END;
```

The For Generate statement has the following characteristics:

• between the keywords FOR and GENERATE there are the following parameters:

• a temporary variable that has a symbolic name, the variable is only used in the For Generate statement and ceases to exist after the compiler has processed the statement, in the above example the variable i. This name cannot be used as a constant, parameter, or node name in this project;

• the IN keyword is followed by a range delimited by two arithmetic expressions, the arithmetic expressions are separated by the TO keyword, in the above example the arithmetic expression is equal to 1 and NUM_OF_ADDRESS, within the ranges there can be expressions consisting only of constants and parameters; the use of variables is not allowed;

• the GENERATE keyword is followed by one or more logical operators, each of which ends with (;);

• The If Generate statement ends with the END GENERATE keywords followed by (;)

3.5. Development of projects of digital devices in AHDL

AHDL templates allow you to enter easily AHDL syntax structures, increasing the speed and accuracy of entering designs.

To insert an AHDL template at the current position:

1. Open the AHDLT template dialog using the Template menu command.

2. Select a name in the TemplateSection window.

3. Click OK.

After entering the template in . tdf , you must replace all variables in the template. Each AHDL keyword is capitalized, and each variable name begins with two underscores (_ _) to identify them.

A generated text output file

You can create one or more text design output files (TextDesignOutputFiles (.tdo)) containing the AHDL equivalent of fully optimized logic for the device used in the project. In addition, the compiler also creates one or more assignment and configuration output files (Assignment&ConfigurationOutputFiles (.aco)), save the . tdo as a text file of the edited project, define it as a project using the ProjectName or ProjectSetProjectto current file menu commands, and recompile the project.

You need to create a project file:

1.Enable GenerateAHDL. do File in the Process menu command.

2. Run the assembly or one of the commands in the menu "File" : "Project". Save and Build or Design Save, compile and simulate in any MAX + PLUSII program.

Numbers are used to represent constant values in logical and arithmetic expressions, equations, and parameter values. *AHDL* supports all combinations of decimal, binary, octal, and hexadecimal numbers.

File Dec. _ Tdf below describes an address decoder that generates an active high signal when the address is 370 Hex.

```
SUBDESIGN dec

(

address [15..0]: INPUT;

chip_enable : OUTPUT;

)

BEGIN

chip_enable = (address [15..0] ==H”0370”);

END;
```

In this simple example, the decimal numbers 15 and 0 are used to define the address bus bits. The hexadecimal number H "0370" defines the decrypted address.

Constants and calculation functions are especially useful when the same number, string, or arithmetic expression is repeated multiple times in a file: if it changes, you only need to change one statement. In AHDL, constants are implemented using the Constant operator, and functions are calculated using the Define operator.

AHDL contains USED , CEIL , and FLOOR functions.

the file dec 2.tdf below has the same functions as dec . tdf , but uses the IO_ADDRESS constant instead of the H number "0370" .

```
ConstantIO_ADDRESS= H"0370";
SUBDESIGN dec
(
A [15..0]: INPUT;
Ce : OUTPUT;
)
BEGIN
ce = (a [15..0] == IO_ADDRESS);
END;
```

Combinational logic

Combinational logic is implemented in AHDL using logic equation expressions, mega truth tables, and macro functions. Examples of combinational functions include decoders, multiplexers, and adders.

Implementation of expressions and logical equations

Logical expressions are a set of nodes, numbers, constants, separated by operators and/or additionally grouped by parentheses. A logical equation sets a node or bus equal to the value of the expression.

The following tdf demonstrates two simple boolean expressions representing two logical blocks.

```
SUBDESIGN bool
(
A0, a1, b: INPUT;
Out1, out2 : OUTPUT;
)
BEGIN
Out1 = a1&! a0;
Out2 = out1#b;
END;
```

In this file out1 is the logical AND of input a1 and inverts a 0, and output out2 is the logical OR of output1 and b. The order in which they are passed in the file does not matter.

You can name logical operators in the project report file to make it easier to enter resource assignments and interpret equations.

bfile 3.tdf is *identical to logical file 1.tdf*, but uses named operators. The name of the operator is separated from the operator by a colon: the name can contain up to 32 characters.

```
SUBDESIGN 22
(
a0, a1, b: INPUT;
out1, out2 : OUTPUT;
)
BEGIN
out1 = a1tiger:&! a0;
out2 = out1 panther: # b;
END;
```

Node declaration

The node that appears with the node declaration in the Variable section can be used to store the value of an intermediate expression. Node declarations are particularly useful when logical expressions are reused.

b 2.tdf file contains the same logic as the bool file. tdf , but has only one output.

```
SUBDESIGN b2
(
a0, a1, b: INPUT;
out: OUTPUT;
)
Variable
a_equals_2 : Node;
BEGIN
a_equals_2 = a1&!a0;
out2 = a_equals_2 # b;
END;
```

This file declares the *node a_equals_2* and binds it to the *expression a1&!* and *0*. Using nodes can save device resources when a node is used in multiple expressions.

Assignment to type Node to combine signals using AND and OR functions.

Bus definition

A bus can contain up to 256 bits, is interpreted as a set of nodes, and works as a whole. The bus name can be specified as a single-range name, a double-range name, or a name in sequential format.

In logic equations a bus can be equal to a mule, another bus, a single node, VCC , GND , 1 or 0. In each of these cases, the values of the bus are different. The Option operator can be used to specify whether the least significant bit will be the most significant bit (MSB), the least significant bit (LSB), or something else.

When a bus is defined, [] brackets are a shorthand way of specifying the entire range. For example, a[4..1] can also be specified as a[]; b[5..4][3..2] can be represented as b[][].

batch file 1.tdf shows logical expressions that define multiple buses.

```
OPTIONS BIT0 = MSB;
CONSTANT MAX_WIDTH = 1+2+3-3-1;
% MAX_WIDTH = 2%
SUBDESIGN group
(
a[1..2], use_exp_in[1+2-2.. MAX_WIDTH]: INPUT;
d[1..2], use_exp_out[1+2*2-4.. MAX_WIDTH]: OUTPUT;
dual_range[5..4][3..2]: OUTPUT;
)
BEGIN
D[] = a[] + B"10";
use_exp_out[] = use_exp_in[];
dual_range[][] = VCC;
END;
```

In this example the OPTIONS operator is used to specify that the rightmost bit of the bus is the MSB, and a decimal 1 is added to the a[bus. The use _ exp _ in and use _ exp _ out buses show how constants and arithmetic expressions can be used to limit ranges tires

The following examples illustrate the use of tires:

- when a tire is compared with a tire of the same size, each element on the right side is equal to each element on the left side in the corresponding position;

- when the bus is tied to VCC or GND, all bus bits are tied to these values;

- when the bus is set to 1, only the least significant bit of the bus is connected to the VCC value. Other bus bits are connected to GND;

- when aligning buses of different sizes, the number of bus bits on the left side of the equation must be exactly divisible by the number of bus bits on the right side of the equation.

For example, the equation

and [4..1] = b[2..1] is correct.

In this equation, the bits are represented as follows:

$a4 = b2$

$a3 = b1$

$a2 = b2$

$a1 = b1$

Implementation of conditional logic

If Then and Case operators are used to implement conditional logic. The If then statement evaluates one or more logical expressions and describes the behavior for different values of the expression. A Case operator is a list of alternatives available for each value of an expression. They evaluate the expressions and then choose a course of action based on the values of the expressions.

Conditional logic implemented using the If statement. Do not confuse this and Case with the logic generated by the *If GENERATE* statement. This logic is optional, it is a condition.

File, priority . The following tdf shows a priority encoder that converts the highest priority active input level to the value of an expression.

```
SUBDESIGNpr33
(
low, middle, high : INPUT;
highest_level[1..0]: OUTPUT;
)
BEGIN
IF high THEN
highest_level[] = 3;
ELSIF middle THEN
highest_level[] = 2;
ELSIF low THEN
highest_level[] = 1;
ELSE
highest_level[] = 0;
END IF;
END;
```

In this example, *the high, medium, and low* inputs are evaluated to determine whether their levels are equal to *VCC*. The *If statement* activates the equation that follows the active *If or ELSE* fields and if the input signal is highhigh then top_level [ ] is 3.

If more than one input is activated, the *If then operator* evaluates the priority of the input data in the order of passing through the If and ELSIF areas (the first area has the highest priority).

the equation after the ELSE keyword will be called.

File, . tdf describes a 2-4-bit decoder below. It converts a 2-bit code to a unary code.

```
SUBDESIGNdec
(
code[1..0] : INPUT;
out[3..0] : OUTPUT;
)
BEGIN
CASE code[] IS
      WHEN 0 => out[] = B"0001";
      WHEN 1 => out[] = B"0010";
WHEN 2 => out[] = B"0100";
      WHEN 3 => out[] = B"1000";
END CASE;
END;
```

In this example, the input bus code is 0, 1, 2, or 3. In the CASE statement , => is followed by the active equation . For example, if *code []* is 1, then output 1 is set to B "0010". Since all the values of the expressions are different, only one WHEN field can be activated at a time .

If the If Then and Case operators *are similar. In some cases, you can use one of two operators to get the same result.*

But there is an important difference between them:

- any kind of logical expression can be used in an If then statement, any expression after the If or ELSIF fields may not be associated with other expressions in the statement. *In contrast, only one logical expression is compared to the constant in each* WHEN field in the *Case statement*;
- using ELSIF can lead to overly complex logic for the compiler, since each subsequent ELSIF statement must still check whether previous If/ ELSIF statements are false.

Creation of decryptors

You can use the truth table functions or pin_compare or lpm_decode to create a decoder in AHDL .

File segment. *The tdf* below is a combinational light emitting diode (LED) decoder, the LEDs display hexadecimal numbers.

```
SUBDESIGN segment
(
I[ 3..0]: INPUT;
a , b, c, d, e, f, g: OUTPUT;
)
BEGIN
TABLE
i[ 3..0] => a, b, c, d, e, f, g;
H"0" =>1, 1, 1, 1, 1, 1, 0;
H"1 " = >0, 1, 1, 0, 0, 0, 0;
H"2 " = >1, 1, 0, 0, 1, 1, 0;
H"3 " = >1, 1, 1, 1, 0, 0, 1;
H"4 " = >0, 1, 1, 0, 0, 1, 1;
H"5 " = >1, 0, 1, 1, 0, 1, 1;
H"6 " = >1, 0, 1, 1, 1, 1, 1;
H"7 " = >1, 1, 1, 0, 0, 0, 0;
H"8 " = >1, 1, 1, 1, 1, 1, 1;
H"9 " = >1, 1, 1, 1, 0, 1, 1;
H"A" => 1, 1, 1, 0, 1, 1, 1;
H"B" => 0, 0, 1, 1, 1, 1, 1;
H"C" => 1, 0, 0, 1, 1, 1, 0;
H"D" => 0, 1, 1, 1, 1, 0, 1;
H"E" => 1, 0, 0, 1, 1, 1, 1;
H"F" => 1, 0, 0, 0, 1, 1, 1;
END TABLE;
END;
```

In this example, the output set p for all 16 possible input sets i[3..0] is described in the statement *TruthTable* .

The decode3 *.tdf* file is an address decoder for *implementing a 16-bit microprocessor system.*

```
SUBDESIGN decode3
(
Addr[ 15..0], m/io : INPUT;
rom , ram, print, sp[2..1]: OUTPUT;
)
BEGIN
TABLE
m/io , addr[15..0] =>rom, ram, print, sp[];
1, B"00XXXXXXXXXXXX" =>1, 0, 0 , B "00";
1, B"100XXXXXXXXXXX " = > 0, 1, 0, B"00";
0, B"0000001010101110" => 0, 0, 1, B"00";
0, B"0000001011011110" => 0, 0, 0, B"01";
0, B"0000011101110000" => 0, 0, 0, B"10";
END TABLE;
END;
```

This example has thousands of input data sets and describing them in a *Truth Table* statement is impractical. Instead, you can use the logic level *X* to indicate that the output is independent of the corresponding input. For example, in the first line of the TABLE statement, the output signal rom must be high for all input data sets, address [15.. 0], starting at 00. Therefore, only the intersection of the input data set should be carefully defined, and the X character should be used for other inputs .

When using X symbols, ensure that the bit combinations in the truth table do not overlap. AHDL assumes that only one condition in a truth table can be true at any time.

Use default values for variables

You can define a default value for the node or bus being used if its value is not specified elsewhere in the file. AHDL allows you to assign a node or bus value more than once in a single file. If these assignments conflict, the default values are used to resolve the conflicts. If no default value is defined, it is assigned the value GND .

The default value is specified using the Defaults statement for variables used in *the Truth Table , If Then , and Case statements* .

Do not confuse the default variable values with the default port values assigned in the Subproject section .

default 1.tdf file below evaluates the input data and selects one of five ASCII codes based on the input data.

```
SUB DESIGN 1
(
I[ 3..0]: INPUT;
ascii_ code[ 7..0]: OUTPUT;
)
BEGIN
DEFAULTS
ascii_code [ ] = I'00111111'; %ASCII code "?"%
END DEFAULTS;
TABLE
I[ 3..0] => ascii_code[];
B "1000"=>B"01100001";
B "0100"=>B"01100010";
B "0010"=>B"01100011";
B "0001"=>B"01100100";
END TABLE;
END;
```

When the input set matches one of the sets listed on the left side of *the truth table statement,* the output is set according to the combination on the right side. If there is no match, the output defaults to B'00111111'.

The default 2.tdf file illustrates how conflicts occur when a single node is assigned more than one value and how AHDL resolves these conflicts.

```
SUBDESIGN 2
    (
    a , b, c : INPUT;
    select_a, select_b, select_c : INPUT;
    wire_or, wire_ and : OUTPUT;
    )
    BEGIN
    DEFAULTS
    wire_or = GND;
    wire_and = VCC;
    END DEFAULTS;
    IF select_a THEN
    wire_or = a;
    wire_and = a;
    END IF;
    IF select_b THEN
    wire_or = b;
    wire_and = b;
    END IF;
    IF select_c THEN
    wire_or = c;
    wire_and = c;
    END IF;
    END;
```

In this example *, wire_or assigns a , b , or c , depending* on *the signals* Select_a *, Select_b , and Select_c* . If neither of these signals is equal to *VCC* , then *the _ or wire* is set to GND .

If multiple *select _ a* , *select _ b* , or *select _ c signals* are *VCC* , then *the signal is either* a logical OR of the corresponding input values.

The signal *wire _ and* signal work the same way, except that they default to *VCC* when none of *the selected* signals is equal to *VCC* , and are logically "ANDed" by their respective inputs when more than one signal is *VCC* .

Active-low level logic

The active low signal becomes active when its value is equal to GND . Active low signals can be useful for controlling memory, peripherals, and microprocessor chips.

Daisy file . *tdf* _ is a module of the arbiter scheme. It accepts bus access requests from itself and from the next module in the chain. The bus is accessed by the module with the highest priority that requested it.

```
SUBDESIGN daisy
(
local_ request : INPUT;
local_ grant : OUTPUT;
 request _in : INPUT;
 request _out : OUTPUT; % to higher priority %
 grant _ in : INPUT;
grant _ out : OUTPUT ; % to lower priority %
)
BEGIN
DEFAULTS
local_grant = VCC;
 request _out = VCC;
% should be equal to default %
grant_out = VCC;
END DEFAULTS;
IF request _in ==GND # local_request == GND THEN
 request _out = GND;
END IF;
IF grant _ in ==GND THEN
IF local_request == GND THEN
local_grant = GND;
ELSIF request _in == GND THEN
grant_out = GND;
END IF;
END IF;
END;
```

All signals in this file are active low. *Altera* recommends choosing a node naming scheme that clearly indicates the names of the active holes, for example, with an " n " at the beginning.

If Operators *If Then* the activity of the modules is determined, that is, whether the signal is equal to GND . If the signal is active, the equations after the corresponding *If Then statement are activated* .

Implementation of bidirectional outputs

MAX+PLUS II allows you to configure the input/output pins as bidirectional. Bidirectional outputs can be defined using the BIDIR port , which is connected to the output of the TRI primitive . The signal between the pin and the TRI primitive is bidirectional and can be used to drive other logic in your design.

The bidirectional I/O signal driven by the TRI primitive is used as the *d input of* the D flip-flop (DFF).

It is also possible to connect bidirectional output from a lower-level TDF file to a higher-level output. The bidirectional output port of a subproject must connect to the bidirectional output port of the top level of the hierarchy. A function prototype for a low-level TDF file must contain bidirectional output in the RETUTNS clause . *Bidir* file . *The following tdf contains four instances* of the bus_reg function mentioned above 2. *tdf* .

```
FUNCTION bus_ reg2( clk, oe)
RETURNS ( io );
SUBDESIGN bidir
(
clk , oe : INPUT;
io[ 3..0] : BIDIR;
)
BEGIN
Io0 = bus_reg2 ( clk, oe);
Io1 = bus_reg2 ( clk, oe);
Io2 = bus_reg2 ( clk, oe);
Io3 = bus_reg2 ( clk, oe);
END;
```

Implementation of tristable buses

TRI primitives that are managed by OUTPUT or BIDIR ports , have an output enable input (OUTPUTEnable), which puts the output in a high impedance state.

Create a tristable bus by connecting TRI primitives and OUTPUT or BIDIR ports together using the TRI_STATE_NODE node . The control scheme should not allow more than one output at a time.

The files *tri _ bustdf* implements the tristable bus using the TRI _ S TATE_NODE node created in the *Node declaration* .

```
SUBDESIGN tri_bus
( in[ 3..1], oe[3..1] : INPUT;
out1 : OUTPUT; )
VARIABLE
 tnode : TRI _ NODE;
BEGIN
tnode = TRI(in1, oe1);
tnode = TRI(in2, oe2);
tnode = TRI(in3, oe3);
out 1 =    tnode ;
END ;
```

In this example multiple node assignments link signals together. To implement three stable buses, TRI _ STATE _ NODE type is required instead of NODE type, , signals are connected with an " AND " wire or "OR" wire for NODE type, and for TRI _ STATE _ NODE type signals are connected with the same node. However, if only one variable is assigned to a TRI _ STATE _ NODE node , it is interpreted as a normal NODE variable .

3.6. Sequential logic

Sequential logic in AHDL can be implemented using state machines and registers or using a library of parametric modules (LPM). State machines are especially convenient for implementing sequential logic. Other examples are counters and controllers.

Register declaration

Registers store data values and synchronize them using a clock signal . You can declare an instance of a registry using a registry declaration in the Variable section (you can also implement a registry using function references in the *Logic section*). AHDL offers some primitive registers and also supports LPM register functions .

Once a register is declared, it can be connected to other logic in the TDF file using its ports. The instance port is used in the following format:

"instance name" "port name"

The tdf file uses *the Register declaration* to create a byte register that captures the value of the *d inputs* on the leading edge *of Clock* when the input load is high.

```
SUBDESIGN reg
(
clk , load, d[7..0] : INPUT;
q[ 7..0] : OUTPUT;
)
VARIABLE
ff [7..0] : DFFE;
BEGIN
Ff[ ].clk = clk;
Ff[ ].ena = load;
Ff[ ].d = d[];
Q[ ] = ff[].q;
END;
```

The registers appear in the *VARIABLE section* as DFFE. The first logical equation in the *" Logic " section.* connect *clk input* with ports *clk* triggers *ff* [7..0]. The second equation connects the load input to the sync enable ports. The third equation connects the data inputs *d* [7..0] to the flip-flop input ports *ff* [7..0]. The fourth equation connects the outputs to the output ports of the flip-flops. All four equations are estimated together.

You can also declare the T , JK , and SR triggers in the *Variable section* and then use them in the *Logic section* .

File *lpm _ rej . tdf* below contains a reference to an implementation of an instance of the function *lpm _ dff* , which has the same functions as the *reg file . tdf*

```
INCLUDE “lpm_dff.inc”
SUBDESIGN lpm_reg
(
clk , load, d[7..0] : INPUT;
q[ 7..0] : OUTPUT;
)
BEGIN
Q[ ] = lpm_dff (.clock = clk, enable = load, data[]=d[])
   WITH( LPM_WIDTH=8)
   RETURNS( .q[]);
END;
```

Declaration of registry results

TDF file registry output data by declaring the output ports as triggers in the *Variable section*. *The_ tdf* file *below has the* same functionality as the *bur _ reg file . tdf* , but has a registered output.

```
SUBDESIGN out
(
clk , load, d[7..0] : INPUT;
q[ 7..0] : OUTPUT;
)
VARIABLE
Q[ 7..0] : DFFE; %also declared as registered%
BEGIN
Q[ ].clk = clk;
Q[ ].ena = load;
Q [ ] = d [];
END ;
```

After assigning a value to the register outputs in the *Logic section* , the value from *the d* inputs is sent to the register. The register outputs do not change until the rising edge *of the clock signal appears* . To specify a register's clock input, use the register-name .c lk construct in *the Logic section* .

Implement a global clock signal *using* the GLOBAL primitive with the *Global boolean Signal parameter in the Individual Logic Option* dialog box or using *automati cGlobal Clock* from *the global Design Logic Synthesis dialog box* ("Assign" menu).

In the above file each DFFF trigger declared in the *Variable section* requests an output of the same name, so you can refer to the outputs of *q* triggers without using port *q* . In a high-level TDF file , the output ports are synchronized with the output pins.

Counter development

The counter can be defined using D triggers (DFF and DFFF) and the *If Then statement or using the lpm_counter* function .

The file cou . *tdf* implements a 16-bit adder with a load that can be reset to zero.

```
SUBDESIGN cou
(
clk , load, ena, clr, d[15..0]: INPUT;
q[ 15..0] : OUTPUT;
)
VARIABLE
Count[ 15..0] : DFF;
BEGIN
Count[ ].clk = clk;
Count[ ].clrn = !clr;
IF load THEN
Count[ ].d = d[];
ELSIF ena THEN
Count[ ].d = count[].q + 1;
ELSE
Count[ ].d = count[].q;
END IF;
END;
```

In this file *there are* 16 triggers with names from 0 to 15 in section variables . Operator *If* defines the value that is loaded into triggers during the implementation of the same function as file *cou. tdf.*

```
INCLUDE “lpm_counter.inc”;
SUBDESIGN lpm_cnt
( clk , load, ena, clr, d[15..0]: INPUT;
Q[ 15..0] : OUTPUT;)
VARIABLE
my_cntr: lpm_counter WITH (LPM_WITDH=16);
BEGIN
my_cntr.clock = clk;
my_cntr.aload = load;
my_cntr.cnt_en = ena;
my_cntr.aclr = clr;
my_ cntr.data[ ] = d[];
q[ ] = my_cntr.q[];
END;
```

Finite state machine

In AHDL finite state machines can also be easily implemented as truth tables in Boolean equations . The language is designed in such a way that you can assign values to states yourself or let the MAX + PLUSII compiler do it itself.

The compiler uses advanced heuristics to automatically assign states, which minimizes the logical resources needed to implement state machines. To do this, simply draw a state diagram and build a table of the following states. The compiler will automatically perform the following actions:

● assigns bits by selecting a T or D trigger (TFF or DFF) for each bit;

● assigns values to states;

● uses the technique of complex logical synthesis to obtain excitation equations .

To define a state machine in AHDL , add the following to your TDF file :

● announcement of the finite state machine (Variables section);

● Boolean control equations (logical section);

● transitions between states in *the table* or list of *cases* (Logic section).

It is also possible to import and export state machines between TDF files and other project files by defining inputs and outputs as machine ports in the *Subproject section* .

Implementation of finite state machines

You can create a state machine by declaring its name, states, and bits in the state machine declaration in the *variables section.*

A simple file . *The tdf* below has the same functionality as the D trigger (DFF)

```
SUBDESIGN simple
(
clk, reset, d: INPUT;
q : OUTPUT;
)
VARIABLE
ss : MACHINE WITH STATES (s0, s1);
BEGIN
```

```
ss.clk = clk;

ss.reset = reset;

CASE ss IS

With s0 =>

q = GND;

IF d THEN

Ss = s1;

END IF;

WHEN s1 =>

q = VCC;

IF !d THEN

ss = s0;

END IF;

END CASE;

END;
```

A finite state machine named *ss is declared* in the *Variable section of the file* . The automaton states are defined as *s 0* and *s 1* , and the state bits are not declared.

The transitions of the finite state machine determine the conditions for the change of state. To specify transitions of a state machine, it is necessary to conditionally assign a state within the framework of one behavioral structure. *Table* or *register* operators are recommended for this purpose . For example, in *simple . tdf* transitions from each state are defined by the *Case operator* .

State output can also be defined using an *If Then* or *Case statement.* In *Case's testimony* , these assignments are used in WHEN clauses . For example, in *simple . tdf* output *q* is assigned the value GND when the state machine *ss* is at *s 0* and VCC when the machine is at *s 1* . Output values can also be defined in truth tables.

3.7. Implementation of hierarchical projects

A TDF written in AHDL can be mixed with other files in the project hierarchy. Low-level files can be files provided by Altera or user-defined mega and macro functions.

Use of non-parametric functions

MAX + PLUSII contains libraries of primitives and non-parametric macro functions. All MAX + PLUSII logic functions can be used to create hierarchical structures. Mega and macro functions are automatically installed in subdirectories of the \ *maxplus 2\ max 2 lib directory* created during installation. Primitive logic is built into AHDL .

There are two ways to use (i.e. inserting an instance) of non-parametric function in AHDL :

● declare a variable for the function, i.e. the instance name, in the *Variable section* of the instance declaration and use the instance ports in the Logic section ;

● use a logic function reference in the LogicTDF section of the file.

Inputs and outputs of mega- and macro-functions must be specified using the function operator (Function prototype). Prototype functions are not required for primitives. MAX + PLUS II contains included files (Include files) from the

prototype of all MAX + PLUSII mega and macro functions *in the \ maxplus 2\ max 2 lib \ mega _ lpm* and *\ maxplus 2\ max 2 inc directories,* respectively . Using the *Include statement* , you can transfer the contents of an *Include file* to a TDF file to declare prototype MAX + PLUSII mega and macro functions .

The tdf macro file contains a 4-bit counter connected to a 4-by - 16 decoder. These functions are created using *instance declarations* in the *Variable section* .

```
INCLUDE “4count”;

INCLUDE “16dmux”;

SUBDESIGN macro

( clk : INPUT;

Out[15..0]: OUTPUT; )

VARIABLE

        counter: 4count;

        decoder: 16dmux;

BEGIN

        counter.clk = clk;

        counter.dnup = GND;

        decoder.(d,c,b,a) = counter.(qd,qc,qb,qa);

        out[15..0] = decoder.q[15..0];

END;
```

File uses operators *Include* for import prototypes functions for two macro functions : 4count and 16dmux. In the *variables* section *a counter* variable *r is* declared as 4 count function instance, and the *variable decodev* is declared as a 16 dmux function instance. The function`s incoming ports in the format < instance name _ >and < port name > are defined in the left part of the logical equations in " Logic " section , a the outgoing ports – in the right part.

The link to *4 count . inc* uses port-element communication during *16 dmux* link . *Inc* uses port name communication. The input ports of both macro functions are defined to the right of the link, and the input ports are defined to the left.

There are equivalent links for different types of port communication in the comments. In a reference, ports to the right of the equal sign (=) can be listed using a positional reference or port name. Ports on the left side of the equation always use positional communication. When using positional communication, the order of the ports is important because there is a one-to-one correspondence between the order of the ports in the function prototype and the ports defined in the" Logic " section .

RETURNS offer is optional for the link. RETURNS can be used to enumerate a subset of the results of the functions used in the instance.

Primitives and macro functions always have default values for unbound inputs, and vice versa, mega functions do not necessarily have them.

Use of parametric functions

MAX + PLUSII includes parametric megafunctions and functions from the Library of Parametric Modules (LPM). For example, parameters are used to specify the port width, regardless of whether the RAM block is implemented as synchronous or asynchronous. Parametric functions can contain other subprojects, which in turn can be parametric or nonparametric. Parameters can be used with some non-

parametric macro functions. All Boolean functions can be used to create hierarchical structures. Mega- and macro-functions are automatically installed in subdirectories \ *maxplus 2\ max 2 lib* , created during installation, primitive logic is built into the AHDL language.

Parametric functions can be accessed via a function link or *instance declaration* in the same way as non-parametric functions, but with a few extra steps:

● an instance of a logical function must contain a WITH clause, which is based on a WITH clause in the function prototype that lists the parameters used by the instance. The parametric value must be bound somewhere in the project, if the instance itself does not contain some or all of the required parameter values, the compiler searches for them in the order of the parameter values;

● since parametric functions do not necessarily have initial values for unconnected inputs, ensure that all necessary ports are connected. On the other hand, primitives and macro functions always have initial values for unattached inputs.

The file *lpm _ add .1 tdf* below *implements an 8-bit adder* by using a reference to *the parametric megafunction lpm _ add _ sub.*

```
INCLUDE “lpm_add_sub.inc”

SUBDESIGN lpm_add1

( a[8..1], b[8..1]: INPUT;

C[8..1] : OUTPUT;

carry_out : OUTPUT; )

BEGIN

%Instance of megafunction with port connection by position%

(c[],carry_out) lpm_add_sub(GND, a[], b[], GND)
```

```
WITH (LPM_WIDTH = 8,

LPM_REPRESENTATION = “unsigned”);

%Equivalent instance with relationship by name%

-- (c[],carry_out) lpm_add_sub(.data a[] = a[], data b[] = b[],

-- cin = GND, add_sub = GND)

-- WITH (LPM_WIDTH = 8,

LPM_ REPRESENTATION = “unsigned”);

END ;
```

CHAPTER IV. WORKING WITH A GRAPHIC EDITOR

By selecting the graphic editor, open a new file. Double-click in the editor window to call the library. First, it is better to work with the mf library which contains standard elements of the 74XX series. The diagram of any library element can be viewed by double-clicking on the selected element. In addition, you can view the directory by performing the following operations:

- select the desired element by clicking the "mouse";
- click on it with the right "mouse" key;
- to chooseinmenu Edit Ports/Parameters option ;
- select Help on.

This reference contains a prototype text editor function and a truth table for the selected item.

When working with a mega function, the following operations must be performed:

- call the desired symbol from the library of megafunctions;
- click inside the symbol with the right "mouse" key;
- set the necessary parameters, cancel redundant entries.

The Altera graphic editor is similar to the PaintBrush editor. But many operations have their own characteristics.

Bus lines are drawn through the Options/LineStyle command with the selection of the type of connecting line. Bus labeling is done in Latin in pointer mode (slanted arrow in the left toolbar), but it is better to do it in text mode by placing the marker in the input/output text or above the line being labeled. For example, rg[12..1], where 12 and 1 are, respectively, the highest and lowest indexes of connections. The designation of a separate link in the bus consists of the name of the bus and its index, for example rg[1]. Lines coming from inputs/outputs can be left unlabeled as they are automatically assigned input/output names.

Inputs and outputs are defined using primitives that are invoked by double-clicking. They must also be named.

A dot on a bus or connection is placed using the tool ruler on the left.

Connections are better made with the help of a tool in the form of a right angle.

Primitives can be used to input GND and VCC ("ground" and power). Primitives are called by double-clicking the left mouse button.

The rotation of the element is performed by the Rotate command, which is selected from the menu that appears after clicking the right mouse button (context menu). The easiest way is to execute the rotation command from the Edit option.

Printing schemes from the graphic editor:

- reduce the scale of the image to the edge to determine the required size of the sheet and the way the image is arranged;
- in the File option select Size and set the required sheet size. This can be done by sorting through the proposed formats;
- check if this sheet contains the entire scheme;
- enter the File option and select Print Setup to set portrait or landscape orientation;
- click on the printer icon.

Creating a library item:

- create a .gdf file and broadcast it;
- click " mouse " for command File/Create Default Symbol.

Implementation of a hierarchical project in the graphic editor:

- display all blocks (subroutines) as *.gdf files;
- compile each block (subroutine);
- use the File/Create Default Symbol command to add each block (subroutine) to your subroutine library as a symbol;
- insert this symbol as a standard element in the main block (program);
- display all links to this element in the main block.

Working with a text editor

This editor creates files with the *.tdf extension. AHDL and VHDL languages can be used.

To view the include files, you need to click on the folder icon, select the maxplus2max2inc directory under the All files key, and view files with the *.inc extension. Their contents can be seen in the maxplus2 max2libmf directory. To see the contents of the function (include), you need to click on the Templates option.

Creating an Include- file:

- create a .tdf file with the TITLE header and broadcast it;
- click with the " mouse " on File/Create Default Include File command ;
- read this file and specify input/output names.

Viewing the include file from the standard library:

- click on Help;
- to choose option Searh for Help on┘ ;
- find primitives name (4count, 8count, etc.);
- click on the "Show" button;

- view contents

Viewing an Include file from your own library:

- open the folder with the desired Include file;
- call your Include file;
- view contents

ettings on the FPGA series MAX7000S:

- enter in the Assign/Device option ;
- set type PLIS MAX7000S in in the Device Family window;
- only then give the name to the project and start the broadcast.

Creating a combined project

Rational design is associated with the use of graphic and text editors at the same time. Standard nodes (counters, registers, adders, etc.) are conveniently created in a graphic editor. Microprogram automata (MPA), arbitrary logical functions, operations with conditions, loops, etc. should be synthesized in a text editor. With such a design, time is saved, and the scheme turns out to be quite transparent. The procedure for solving the problem in this case can be as follows:

- divide the scheme into standard nodes and non-standard blocks;
- create project(s) describing all non-standard blocks in a text editor;
- broadcast the text file(s) with the local name of the project and create a symbol corresponding to this non-standard block;
- in the graphic editor create a file with a global name to include all non-standard blocks in the form of symbols and all standard nodes;

- broadcast the received file.

Compiling and setting up

Specify the name of the project (the name of the main file with the extension *.gdf) in the File/Project option before compiling. Compilation is done through the MAX+plusII/Compiler option or through the main toolbar. If the compiler window appears, press the Start key.

If there are errors in the compiled file, corresponding messages appear. By highlighting the desired message and clicking the Location button, you can localize the error.

To cancel the global inputs of clocking, reset and crystal selection, you need to select the Assign option in the menu, and in it Global Project Logic Synthesis and remove all "checkmarks" in the Automatic Global, MAX Device Synthesis Options and other sections. Make sure the Global Project Synthesis Style window is set to NORMAL. In this case, global inputs can be used as normal.

To determine the scope of the project, the following operations must be performed:

- click the "mouse" on the "pyramid";
- open rpt option;
- view the message for the %LCS Utilized option.

Simulation

Open the file with the extension *.scf by clicking on the corresponding icon of the main toolbar. Double-click to open the input/output variables input menu.

To set the simulation time interval, you need to enter the File/EndTime option and set the desired time (us, ms, s) or scale (1, 10, 100, 1000).

To set the clock frequency, you need to use the left toolbar (the icon with the "C" symbol) after clicking on the clock frequency chart.

To set the values of the input signals, you need to set the duration of the signal by clicking and dragging with the help of a vertical ruler, and then its value using the icon with the symbol "1" on the left toolbar.

If the step task does not work when setting the frequency, then click on the 3rd icon on the right on the horizontal toolbar, and then repeat the frequency task again on the vertical ruler.

To start the simulation package (simulator), click the simulator button on the main toolbar.

Assignment of resources

To specify pin numbering the following operations should be performed:

- enter the Assign menu;
- choose Pin/Location/Chip option ;
- in the window, write down the name of the node and the output number of the FPLIS.

You can view the results of your work on resource allocation by calling the .acf file, where the project name is.

FPGA programming

After assigning resources, FPGA programming is possible. Programming is performed in the following order:

- with the PC turned off, connect the ByteBlaster cable to the printer connector of the PC and the programmable device;

- turn on the PC, start MAX+PLUS II;
- supply power (usually +5) to the programmable device;
- enter the File menu option and specify the name of the project;
- select the MAX+plus II/Programmer option and the Program button or simply click the "mouse" on the programming icon;
- ensure successful programming;
- turn off the power supply of the programmable device;
- exit MAX+PLUS II;
- turn off the PC;
- disconnect the ByteBlaster cable.

The procedure for developing a new project in the automated system MAX+PLUS II design

The procedure for developing a new project from concept to the completion can be simplified as follows:

1. creation of a new project file or a hierarchical structure of several project files using various project development editors in the MAX+PLUS II system, i.e. graphic, text and signal editors;

2. assignment of the top-level project file name (Top of hierarchy) as the project name (Project name);

3. assignment of programming logic integrated circuits for project implementation. The user can assign a specific device himself or delegate this action to the compiler for the purpose of evaluating the available resources;

4. opening the compiler window and launching it with the "Start" button to start compiling the project. If the user wishes, he can connect the Timing SNF Extractor module to create a layout file used when testing timing parameters;

5. in case of successful compilation – testing and time analysis, for which the following actions must be performed:

5.1. to perform time analysis, open the “Timing Analyzer” window, select the analysis mode and press the “Start” button;

5.2. for testing, you must first create a test channel file (*.scf) using a signal editor, or in a test vector file (*.vec) using a text editor;

6. programming or downloading the configuration of the synthesized device is done by starting the programmer and then inserting the device into the programming adapter of the MPU (Master Programming Unit) programmer or by connecting the MasterBlaster, ByteBlaster device or the FLEX download cable (FLEX Download Cable) to the device, which programmed in the system;

7. selecting the “Program” button to program the device with EPROM memory (MAX, EPC) or selecting the Configure button to download the configuration of the device with SRAM type memory (FLEX).

When working with MAX+PLUS II you should distinguish between project files, support files and projects.

A project file is a graphic, text, or signal file created using a graphics or signal editor for the MAX+PLUS II Environment Editor. This file contains the project logic and is processed by the compiler. The compiler automatically processes the following project files:

1) graphic files of the project (*.gdf);

2) text files of the project in the AHDL language (*.tdf);

3) project signal files (*.wdf);

4) project files in VHDL language (*.vhd);

5) project files in the Verilog language (*.v);

6) OrCAD schematic files (*.sch);

7) input EDIF files (*.edf);

8) Xilinix Netlist format files (*.xnf);

9) Altera project files (*.adf);

10) digital files (*.smf).

Additional files are files related to a MAX+PLUS II project, but they are not part of the project tree. Most of such files do not contain a description of the logical

functions of the project. Some of them are created automatically by applications to the MAX+PLUS II system, others by the user.

A project consists of all files in the design hierarchy, including support and input files. The project name is the top-level file name without the extension. The MAX+PLUS II system compiles, tests, hourly analyzes, and programs the entire project at once, although the user may be editing files for that project in another project at the same time.

For each project, you should create a separate subdirectory in the working directory of the system (\max2work).

When the MAX+PLUS II system is started, its main window (Main Window) (fig. 4.1) opens automatically, covering all system applications. The name of the last project with which the user worked is written in the uppermost line. The following two rows are typical for Windows: the main menu bar and the toolbar, on the left of which are the usual Windows tools (New, Open, Save, Print, Cut, Copy, Paste, Undo), and on the right are package-specific tools using which launch the main applications of the package.

Fig. 4.1. The main window of the MAX+PLUS II automated design system

The system components should be launched through the MAX+PLUS II menu window (fig.4.2), which contains a submenu for calling the main applications: hierarchy overview, graphic editor, symbol editor, text editor, signal editor, level planner, compiler, simulator, analyzer of time parameters, programmer and message generator, the functional purpose of which has already been described in the previous section.

In the hierarchical structure of the project it is allowed to mix the use of files with the extensions .gdf, .tdf, .vhd, .v, .edf, .sch. at any level. However, files with the extension .wdf, .xnf, .adf, .smf must either be at the lowest hierarchical level of the project or be a single file.

All MAX+PLUS II applications provide the ability to enter, edit, and delete assigned resource types, devices, and parameters that control project compilation, logical synthesis, and partitioning using commands from the “Assign” menu. In fig. 4.3 presents the Assign menu commands. The user can perform assignments for the current project regardless of whether any project file or application window is open.

Fig. 4.2. MAX+PLUS II menu window Fig. 4.3. Project assignment menu Assign

The MAX+PLUS II system stores information for the project in the file with extension acf . Change of assignments made in the level planner window are also stored in format _ acf. In addition , the user has the possibility to edit the acf file of the project in a text editor .

The following functions is common to all of MAX+PLUS II applications: assignment of device, resources and probes , saving previous ones versions , global device options in the project , global project parameters , global requirements for hours project parameters , global logical synthesis of the project.

There is a resource part of the Altera device, such as a contact or logical element which performs specific, defined by the user tasks. Management compilation of the project and its time parameters is carried out using diverse assignments. There are the following types of assignments.

Clique assignment sets what exactly logical functions should stay together. Grouping logical functions in clique guarantees that they are implemented in the same block logical structure of the device or the same row.

Chip assignment sets what logical functions must be implemented in the same device in case of division the project into parts (several devices).

Pin assignment assigns input or output of a logical function such as a primitive or megafunction, a specific contact or a horizontal (vertical) row of FPGA outputs.

Location assignment (assignment cell) sets placing logical function (node) in a concrete logical element. In the fields of this window you can set the output number , logical center or block, as well as using the “ Change ” and “ Delete ” buttons to change appointments.

Probe assignment provides an easy to remember unique name of input or output of a logical function.

Connected pin assignment sets external combination of two or more pins in the diagram user This information is useful also in mode testing hours parameters schemes and when testing several composed projects.

Local routing assignment assigns the coefficient distribution by output node of logical element that is in the same logical elements block or in the neighboring one, adjacent with the chosen node using local connections. Local routing is also carried out between the node placed in the logic block elements on the periphery of the device and the output contact with which it is connected. Local routing assignment is carried out using Assign / Local routing command.

Device assignment assigns the type of FPGA in which the project will be implemented. If the project tconsists of several devices, then this function assigns chips to specific devices. to The option Auto can be also choosen to provide the compiler the right to choose a device from the given homeland devices . You can use the automatic device selection process to manage by setting the range and number of devices in the homeland. If a project is very large for implementation in one device, you can set the type and number of additional devices. The “ Assign / Device ” command is used to select the device.

Logic option assignment controls the synthesis of individual logical functions during compilation using the style of logical synthesis and separate options of a logic synthesizer. Altera provides a large number of logical options as well as ready styles, each of them is a collection of settings for logical options, united by one synthesis style name. A user may use ready styles and or create new ones. Synthesis styles allow to adjust synthesis options on certain homeland devices , taking into account the architecture of homeland. The command "Assign / Logic Options " is used for synthesis styles.

Timing assignment controls logical synthesis and adjustment of individual logical functions to obtain necessary values for time delay. A user also can also cut out connection between paths for a particular signal and others cells or project blocks. Timing assignment of block parameters occurs by Assign /Timing Requirements command.

You can introduce global timing requirements for the project by setting general characteristics for delay using the command “Assign / Global Project Timing Requirements".

Assign / Global Project Logic Synthesis command is used for assignment of global parameters of the project logical synthesis.

4. 2. The procedure for compiling the created project in the MAX+PLUS II automated design system

First the compiler retrieves information about hierarchical relationships between project files and checks the project for common design input errors. It creates an organizational map of the project and then, by combining all the project files turns them into a non-hierarchical database that it can handle efficiently.

The compiler uses various means of increasing the efficiency of the project and minimizing the use of device resources. If the project is too large to be implemented in one programmable logic integrated circuit, the compiler can automatically break it into parts for implementation in several devices of the same family of programmable logic integrated circuits, while minimizing the number of connections between devices. The reporting file (.rpt) will show how the project will be implemented: in one or more devices.

The compiler can automatically compile the project. There are opportunities to specify the processing of the project in accordance with the exact instructions of the developer. For example, it is possible to set the style of logical synthesis of the project and other parameters of logical synthesis within the limits of the entire project. In addition, it is convenient to set hourly requirements within the framework of the entire project, to specify precisely the division of a large project into parts for implementation in several devices, and to choose options for device parameters that will be used for the entire project as a whole. The user is able to choose the number of outputs and logic elements that will remain unused during a continuous compilation to reserve them for subsequent project modifications.

Compilation can be started from any MAX+PLUS II application from the compiler window. The compiler automatically processes all input files of the current project.

The compilation process can be seen in the compiler window (fig. 4.4) as follows:

1) the sand time is emptied and flipped indicating the activity of the compiler;

2) the rectangles of the compiler modules are highlighted one by one;

3) an icon of the output file generated by this module appears under the rectangle of the compiler module;

4) the percentage of compilation completion gradually increases (up to 100%);

5) the Stop button of the compiler during disassembly and assembly turns into a Stop/Show Status button, which the user can select to open a dialog box that displays the current status of the disassembly and assembly of the project;

6) when any errors or possible problems are detected in the compilation process, a message handler window is automatically opened, which displays a list of error messages, warnings and informational messages and also provides immediate help on correcting the error. In addition, the user can define the message sources in the project files or its peer assignment plan.

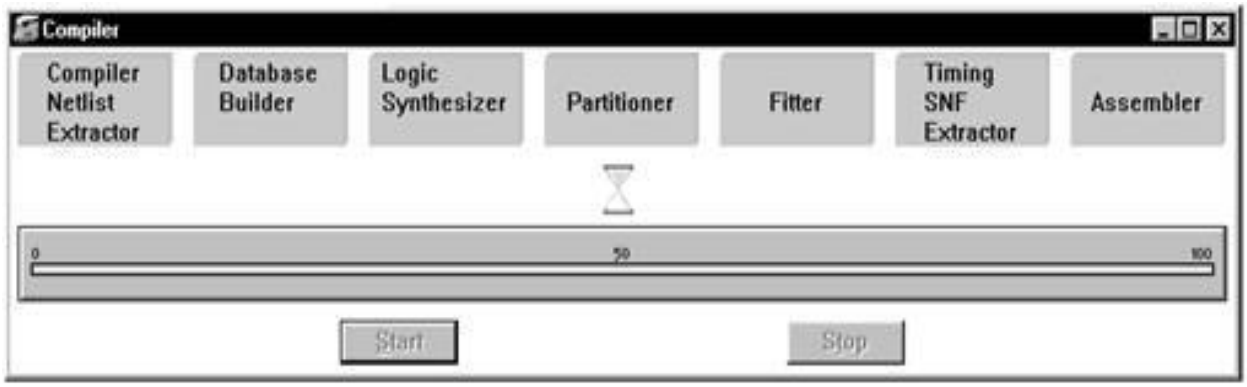


Fig. 4.4. Compilation process of the project

The MAX+PLUS II automatic design system compiler processes the design using the following modules and utilities:

1) Compiler Netlist Extractor which includes programs for reading EDIF, VHDL, Verilog, XNF formats;

2) Database Builder; 3) Logic Synthesizer;

4) Partitioner;

5) Fitter;

6) Functional SNF Extractor;

7) Timing SNF Extractor;
8) Linked SNF Extractor;
9) EDIF Netlist Writer;
10) Verilog Netlist Writer;
11) VHDL Netlist Writer;
12) Assembler;
13) Design Doctor Utility.

The Compiler Netlist Extractor converts each project file into one or more binary files with the extension.cnf (compiler netlist file). Since the compiler substitutes the values of all parameters used in parameterized functions, the contents of the cnf file may change during successive compilations if the parameter values change. This module also creates a file of hierarchical interconnections, with the extension.hif (hierarchy interconnect file). This file documents the hierarchical relationships between project files, as well as the information needed to display the hierarchy tree in the “Hierarchy Display” window. In addition, this module creates

a node database file with the extension.ndb (node database) which contains the names of project nodes for the resource assignment database.

The Database Builder module uses the hierarchical relations file to compose the cnf- files created by the compiler that contain the project description. Based on the data about the hierarchical structure of the project, this module copies each cnf - file to one database without a hierarchical structure. Thus, this database stores the electrical connections of the project.

When creating a database, the module examines the logical completeness and consistency of the project, as well as checks border connections and the presence of syntactic errors. At this stage of compilation most errors are detected and can be easily corrected. Each compiler module sequentially processes and updates this database.

The first time the compiler processes a project, all project files are compiled. The user has the opportunity to choose "smart recompile" to create an extended project database, which allows to speed up subsequent compilations. It is possible to choose between recompiling only those files that have been edited since the last compilation or full recompilation using the total recompile option.

The Logic Synthesizer module uses a number of algorithms that reduce the use of resources and remove duplicated logic, thereby ensuring the effective use of the structure of the logical element for the architecture of the entire homeland of devices. In addition, the logic synthesizer searches for logic for unconnected nodes. If it finds such a node, it removes the primitives related to such a node.

If the project does not fit in one device during the installation, the Partitioner module divides the database into several FPGAs of the same parent, while trying to divide the project into the minimum number of devices.

Using the database updated by the partitioning module, the Fitter module matches the project requirements with the known resources of one or more devices.

It assigns the position of the logical element to each logical function that implements it and chooses the appropriate ways of interconnections and assignment of outputs.

Extractor for functional testing (Functional SNF Extractor) creates a file for functional testing with the extension.snf. The compiler generates this file before synthesizing the project, it contains all nodes present in the initial project files.

Timing SNF Extractor, if the project compiles without errors, creates a file for testing timing parameters containing data about the timing parameters of the project. The file extension is also.snf.

The Linked SNF Extractor creates a file (.snf) for testing the layout of multiple projects (at the board level). Such a file combines information from snf- files of two types: for testing time parameters and for functional testing, which were synthesized for these several projects separately.

A program for recording the source file in the EDIF format (EDIF Netlist Writer). The MAX+PLUS II compiler can interact with most standard automatic design system software that can read standard EDIF 200 or EDIF 300 format files. Edo.

A program for writing a source file in Verilog (Verilog Netlist Writer). An optional Verilog writer module generates output files with a.vo extension containing information about the functions and their time parameters obtained after the synthesis.

A program for recording the output file in VHDL format (VHDL Netlist Writer). The optional VHDL writer compiler module generates one or several output files (.vho) in VHDL with 1987 or 1993 syntax.

The assembler module converts the logic, pin and device assignments made by the trace module into a software image for the device in a form of one or more binary programmer object files (.pof) and SRAM object files (.sof).

The Design Doctor Utility checks the logic of each design file to identify elements that may cause reliability problems at the system level. These problems are revealed only after starting the device "in iron". There is an option to choose one of the three previous project processing rules with different levels.

4.3. General information about the hardware description language AHDL

The hardware description language AHDL (Altera Hardware Description Language) was developed by the Altera company and is intended for the description of combinational and sequential logic devices, group operations, digital automata taking into account the features of the Altera FPGA. It fully integrates with the MAX+PLUS II automatic design system. Hardware description files written in AHDL have the extension .tdf (Text design file). To create a tdf- file you can use both MAX+PLUS II system and any other text editor. The project made in the form of a tdf-file is compiled and used to create a programming file or download an Altera FPGA.

Operators and elements of the AHDL language are quite powerful and universal means of describing the algorithms of the functioning of digital devices. The hardware description language AHDL makes it possible to create hierarchical projects within the framework of one of these languages or to use both tdf-files written in the AHDL language and other types of textual hardware descriptions in a hierarchical project. To create AHDL projects, you can usually use any text editor, but the text editor of the MAX+PLUS II system provides a number of additional options for entering, compiling and verifying the project.

Files created in the AHDL language are easily integrated into the hierarchical structure of the project. The MAX+PLUS II system allows you to automatically create a symbol of a component which operation algorithm is described by a tdf-file, and then insert it into a schematic design file (gdf- file). In addition, the user can enter his own functions to about 300 macro functions developed by Altera.

Altera supplies files with the extension .inc (include design file) for all functions included in the macro library of the MAX+PLUS II system.

The designer can use text editor programs or AHDL language operators when allocating device resources. In addition, the developer can check the syntax and do a full compilation. Any errors are automatically recorded by the message handler and information about their presence appears in the text editor window, which optimizes the device development time.

4.4. Implementation of basic microelectronics devices in MAX+PLUS II integrated environment

The basic structural unit for building combinational logic circuits is a logic element (valve). The role of such a structural unit is played by a trigger in case of sequential logic circuits. In this section different types of triggers will be considered.

Conventional notation of the RS-trigger is given in fig. 7.1.1: An RS flip-flop has two inputs R and S and two outputs Q 1 and Q 2. The outputs are always in opposite (coplanar) states in flip-flops. In other words, if at the input of Q_1 we have a logical one, then at the output of Q_2 there will be a level of logical zero, and vice versa. The R and S inputs of the flip-flop in question are called the set input 1 and the set input 0, respectively.

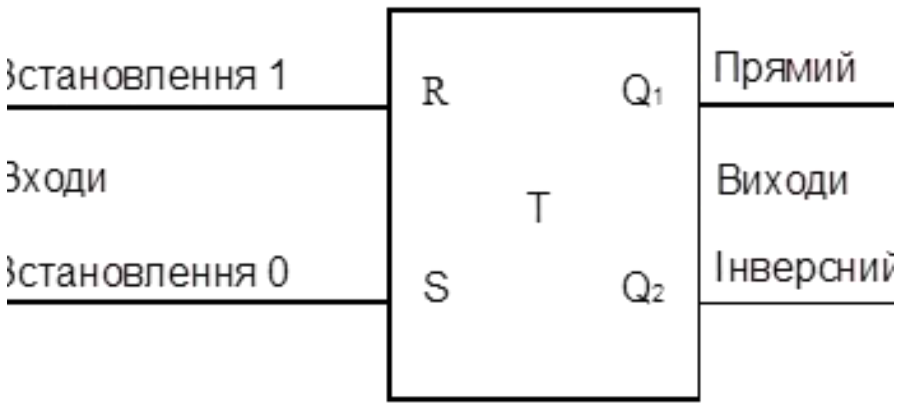


Figure 4.5. Symbol of the RS-trigger

The principle of operation of the RS-trigger is illustrated by its truth table (Table 4.5).

Table 4.1. RS flip-flop truth table

Mode of operation	Input		Output		
	S	R	Q_1	Q_2	Effect on output Q 1
Forbidden state	0	0	1	1	Forbidden - is not used
Setting 1	0	1	1	0	To set Q_1 in 1
Setting 0	1	0	0	1	To set Q_1 in 0
Save	1	1	Q_1	Q_2	Depends on the previous state

When a logical zero level (R=S=0) trigger is applied to both inputs, a logical unit ($Q_1 = Q_2 = 1$) is set at both outputs. This is a forbidden trigger condition; it is not used. According to another row of the truth table, a logical 1 is set at the output of Q 1. In this case, it is said that the flip-flop is set to state 1. According to the third row, when S=1 and R=0, the signal is reset (clearing the output of Q_1) at the input of Q_1 to the level of logic 0. This means that the trigger is set to state 0. The fourth row of the truth table corresponds to R=S=1. In this case, the trigger is at rest: the previous complementary signal levels are stored at the outputs Q_1 and Q_2. This is a saving mode.

It can be seen from the table 4.1 that setting the trigger to state 1 (setting 1 at the output Q_1) initiates a logical 0 at the input S. Similarly, setting the trigger to state 0 (setting 0 at the output Q_1) of the RS-trigger state is due to the appearance of a 0 at one of its inputs, then, more likely, a more accurate image of this scheme would be a conventional graphic image shown in fig. 4.6.

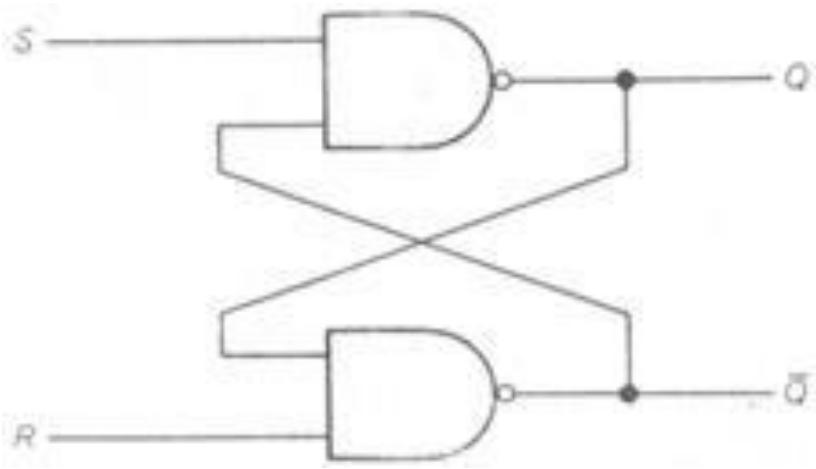


Fig. 4.6. RS-trigger built on AND-NOT logic elements

Particular attention should be paid to the inverting coils at the R and S inputs. They show that the active signal level for setting the flip-flop to the 1 and 0 states is a logic 0 level on one of the inputs. An RS flip-flop is often called an RS latch or a split- inputs flip-flop.

The principle of operation of the synchronous RS-trigger is illustrated by its truth table (Table 4.2).

Table 4.2. Truth table of the synchronous RS trigger

Mode of operation	Input			Output		
	CLK	S	R	Q_1	Q_2	Effect on output Q 1
Saving	____	0	0	Unchanged		Unchanged
Setting 0	____	0	1	0	1	To set Q_1 in 0
Setting1	____	1	0	1	0	To set Q_1 in 1
Forbidden state	____	1	1	1	1	Forbidden - is not used

Only the top three rows of the truth table describe the real modes of operation of the RS-trigger. The bottom line corresponds to the forbidden state and is never used. It can be seen from the table that the state of the outputs of the synchronous RS-trigger

can change only at the time of arrival of clock pulses. In this case, it is said that the trigger works synchronously: its switching process is in synchronism with the clock pulses.

An important role in many digital circuits is played by another characteristic of the RS-trigger - the presence of memory. Indeed, if the trigger is set to 1 or 0, it remains in this state even with some changes in the input signals.

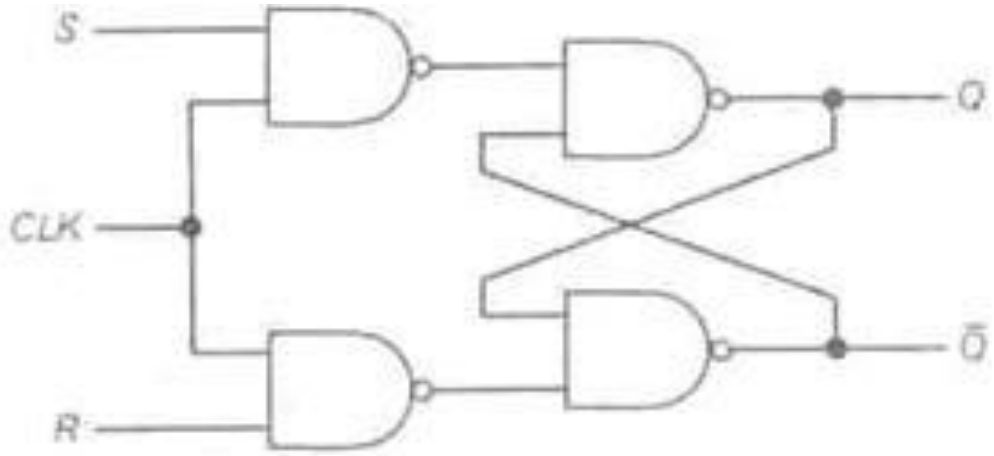


Fig. 4.7. Synchronous RS-trigger built on AND-NO logic elements

To get a synchronous RS- trigger you need to introduce two additional AND-NOT logic elements into the circuit of a regular RS- trigger as shown in fig. 4.7.

Conventional graphic designation of the D-trigger is presented in figure 7.1.6. This flip-flop has only one data input D, as well as a synchronizing input CLK. The D flip-flop is often called a delay trigger. The word "delay" denotes what happens to the data (information) that comes to the input D..

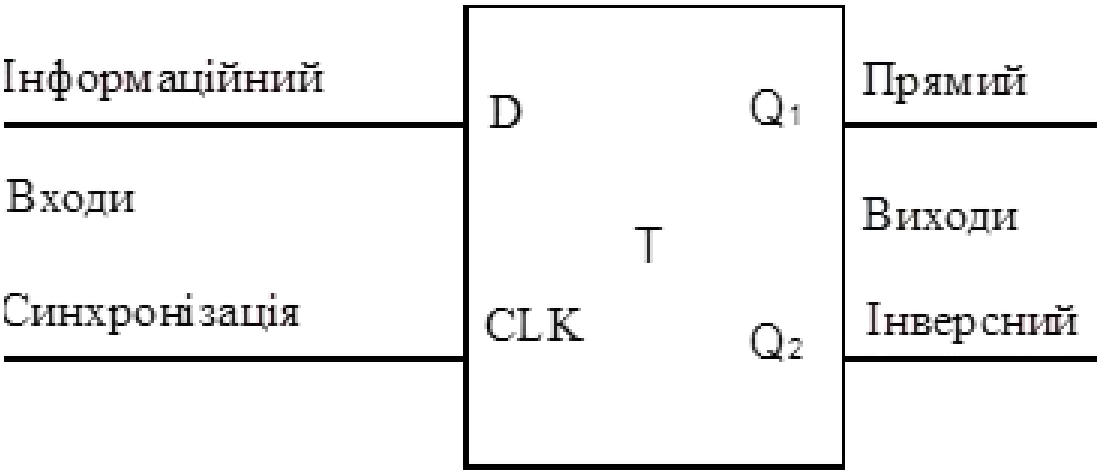


Fig. 4.8. Conventional graphic designation of the D-trigger

It should be noted that the signal at the output Q in clock n+1 repeats the signal that was at the input D in the previous clock n.

A D- trigger can be obtained from a clocked RS flip-flop by adding an inverter to the latter, as shown in fig. 4.8.

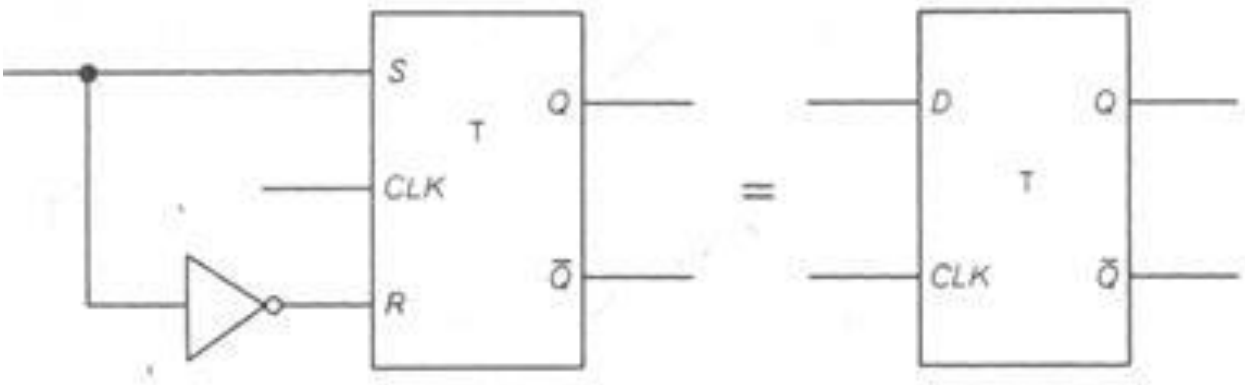


Fig. 4.9. D-trigger scheme

The conventional designation of a typical mass-produced D-trigger is shown in fig. 4.9. It has two additional inputs - pre-set (PS) and clear (CLR). A logic 0 at the PS input initiates the setting of a logical 1 at the Q output. A logic 0 at the CLR input initiates a Q clear output.

The PS and CLR inputs block the action of the D and CLK inputs in active states; when unlocked, the D and CLK inputs act similarly to the usual D-trigger shown in fig. 4.8.

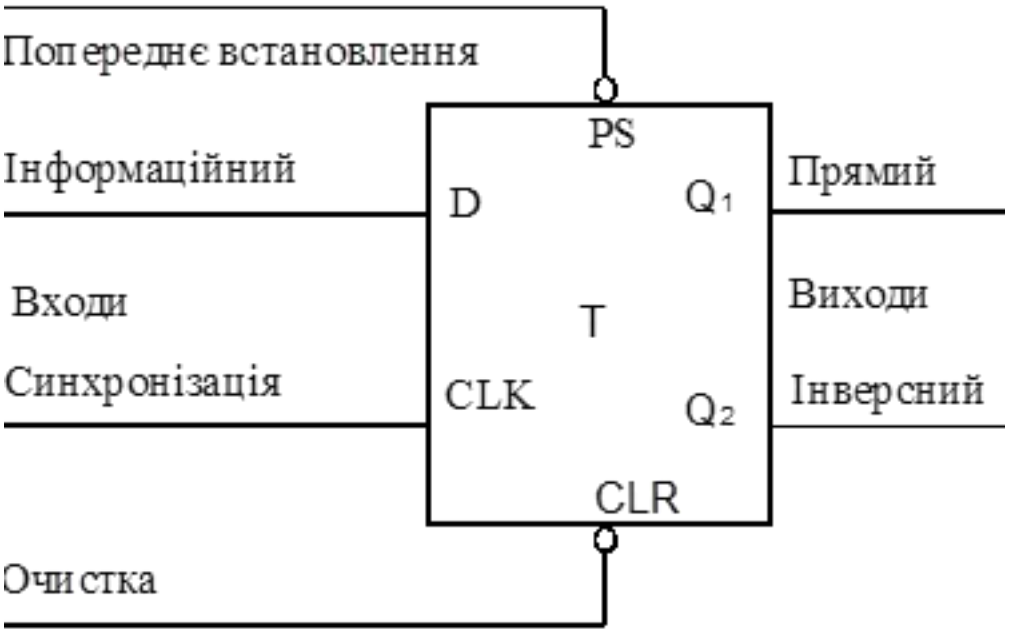


Figure 4.10. Conventional graphic designation of a serial integral D-trigger

The JK trigger is a universal trigger that has the characteristics of all other types of triggers. Conventional graphic designation of the JK-trigger is given in fig. 4.11. The JK flip-flop has two information inputs: J and K and a CLK synchronization input and, like all flip-flops two complementary outputs Q 1 and Q 2 . The validity table for the JK-trigger is given in table. 4.3. When both inputs J and K are applied to a logic 0 level, the flip-flop is blocked and the state of its outputs does not change. In this case, the trigger is in a save mode.

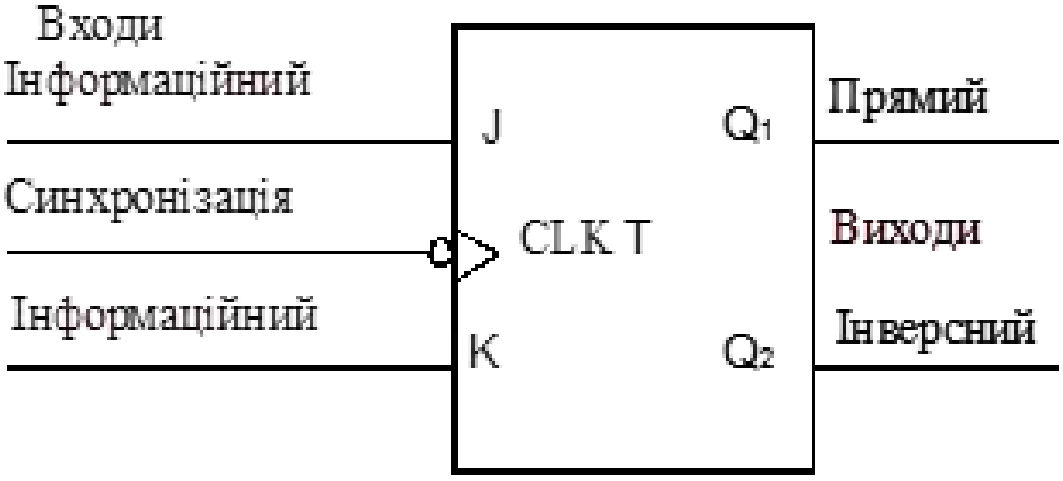


Fig. 4.11. Conventional graphic designation of the JK-trigger

Table 4.3. JK- trigger truth table

Mode of operation	Input			Output		
	CLK	J	K	Q1 _	Q_2	Effect on output Q_1
Save	____	0	0	Unchanged _		Unchanged - _ locking
Setting 0	____	0	1	0	1	Reset or clearing Q_1 in 0
Setting 1	____	1	0	1	0	To set Q_1 in 1
Switching	____	1	1	Switching		Change the status to the opposite

Lines 2 and 3 of the truth table describe the modes corresponding to setting the trigger from state 0 to 1. Line 4 illustrates a very important mode of operation of the JK-trigger- the switching. If both the J and K inputs are set to logic 1, then subsequent clock pulses will cause the signal levels at the trigger outputs to flip from 1 to 0, from 0 to 1, etc. This work is similar to switching the toggle switch, that is where the name of the mode comes from.

Conventional graphic designation of the JK trigger, which is a part of the integrated circuit is shown in fig.4.12. This trigger has two additional asynchronous inputs (a preset input and a clear input).

The synchronous inputs are the information inputs J and K and the synchronizing input CLK.

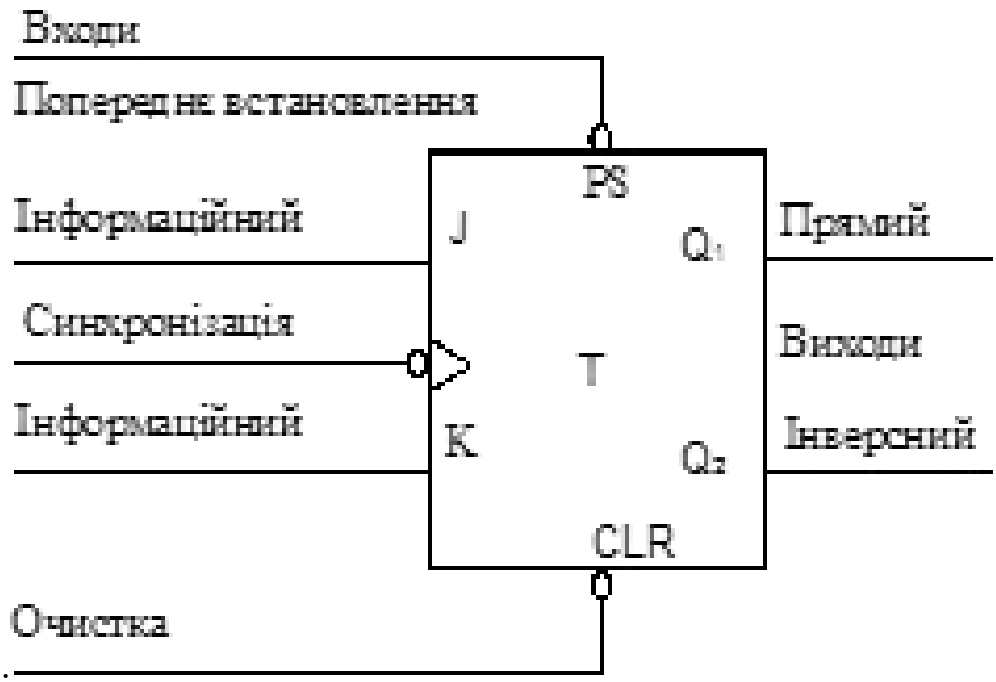


Fig. 4.12. Conventional graphic designation of a serial integral JK- trigger

Trigger primitives must be used when implementing triggers using the AHDL language.

All trigger primitives used in describing the operation of the equipment are given in the table 4.4.

Table 4.4. Primitives triggers in AHDL

Primitive	The prototype of the primitive
DFF	FUNCTION DFF (D, CLK, CLRN, PRN) RETURNS (Q)
DFFE	FUNCTION DFFE (D, CLK, CLRN, PRN, ENA) RETURNS (Q)
TFF	FUNCTION TFF (T, CLK, CLRN, PRN) RETURNS (Q)
TFFE	FUNCTION TFFE (T, CLK, CLRN, PRN, ENA) RETURNS (Q)

JKFF	FUNCTION JKFF (J, K, CLK, CLRN, PRN) RETURNS (Q)
JKFFE	FUNCTION JKFFE (J, K, CLK, CLRN, PRN, ENA) RETURNS (Q)
SRFF	FUNCTION SRFF (S, R, CLK, CLRN, PRN) RETURNS (Q)
SRFFE	FUNCTION SRFFE (S, R, CLK, CLRN, PRN, ENA) RETURNS (Q)
LATCH	FUNCTION LATCH (D, ENA) RETURNS (Q)

Triggers outputs:

D, T, J, K, S, R – information inputs;

CLK – clock signal input (active 0-1 drop);

CLRN – asynchronous trigger reset input (active level – logical zero);

PRN – asynchronous trigger setting input (active level – logic zero);

ENA – work permit input (active level – logical unit).

The program for implementing triggers using the AHDL language in the MAX+PLUS II integrated environment looks like this: Title "triggers";

```
Subdesign triggers
(
D, T, J,K,S,R,CLK,CLRN,PRN,ENA : input;
Q1, Q2, Q3, Q4, Q5, Q6, Q7, Q8, Q9: output;
)
```

Begin

Q1 = DFF (D, CLK, CLRN, PRN); Q2 = DFFE (D, CLK, CLRN, PRN, ENA);

Q3 = TFF (T, CLK, CLRN, PRN); Q4 = TFFE (T, CLK, CLRN, PRN, ENA);

Q5 = JKFF (J, K, CLK, CLRN, PRN);

Q6 = JKFFE (J, K, CLK, CLRN, PRN, ENA);

Q7 = SRFF (S, R, CLK, CLRN, PRN);

Q8 = SRFFE (S, R, CLK, CLRN, PRN, ENA);

Q9 = LATCH (D, ENA);

End;

The signal editor window of the Triggers project is shown in fig. 4.13.

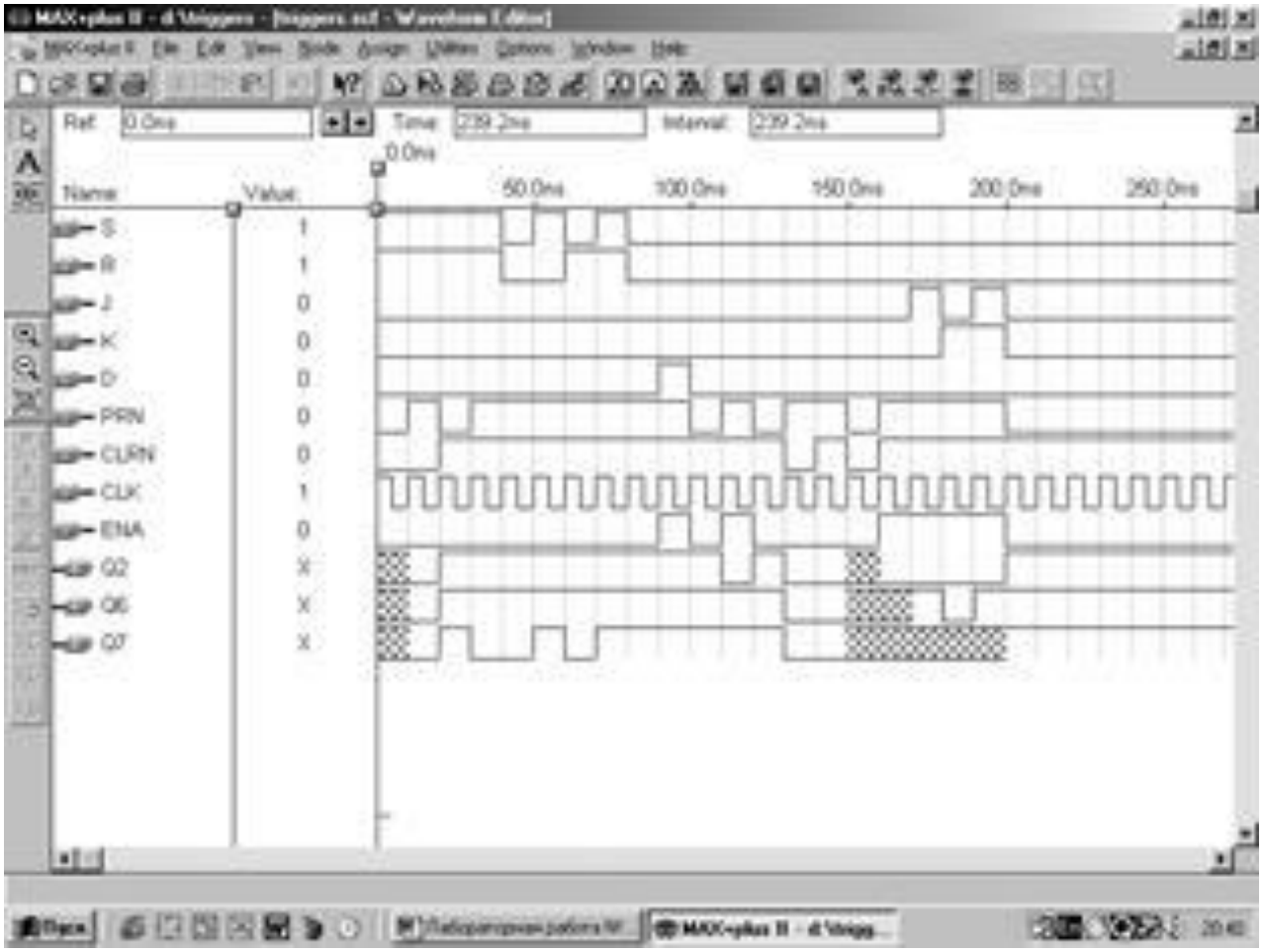


Fig. 4.13. Results of testing RS-, D-, JK-triggers

4.5. Theoretical information about registers

The diagram of one of the typical shift registers is shown in fig. 4.14. This register is implemented on 4 D-triggers. Such a register is called a 4-bit shift register, because it allows storing 4 binary bits of A, B, C, D data.

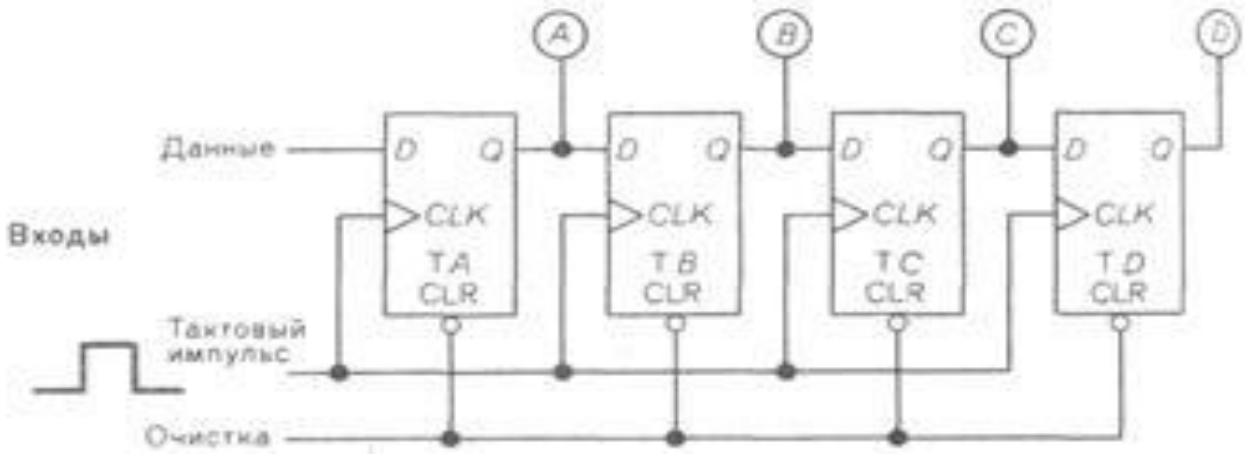


Fig. 4.14. 4-bit serial shift register

We have the opportunity to observe the operation of this device using the table 4.4. and fig. 4.14. First, let's clear the register (set logical zero levels on its outputs A, B, C, D). To do this, apply a logic 0 to the CLR clear input. Row 1 of the table 4.4. corresponds to the received state of the shift register. The register outputs remain in the state 0000 before the arrival of the clock pulse. Let's apply the first pulse to the CLK synchronizing input; the indicator will show the number 1000, since a logic 1 on the clock pulse from the information input of the trigger TA is transferred to its logic output Q, bit A, and the previously entered units are shifted one position (digit) to the right. In the same way, when a logical 0 is applied to the information input, this zero will be entered into bit A at each clock pulse, and the previously entered units and zeros will be shifted to the right. Before the arrival of clock pulse 9, the information input is set to 1, and before the arrival of pulse 10 this input returns to 0. In the hour of operation of clock pulses 9-13, the unit entered into the register on

pulse 9 will shift to the right on the indicator. Line 15 in the table 4.5. shows that on pulse 13 this one leaves the rightmost bit of the shift register and is lost.

Table 4.5. Operation of a 4-bit shift register

Input				Output			
Number of lines	Cleaning	Data	Clock pulse number	TA A	TB B	TC C	TD D
1	0	0	0	0	0	0	0
2	1	1	0	0	0	0	0
3	1	1	1	1	0	0	0
4	1	1	2	1	1	0	0
5	1	1	3	1	1	1	0
6	1	0	4	0	1	1	1
7	1	0	5	0	0	1	1
8	1	0	6	0	0	0	1
9	1	0	7	0	0	0	0
10	1	0	8	0	0	0	0
11	1	1	9	1	0	0	0
12	1	0	10	0	1	0	0
13	1	0	11	0	0	1	0
14	1	0	12	0	0	0	1
15	1	0	13	0	0	0	0

We remind you that a D-trigger is also called a delayed trigger. It simply transfers the information signal from input D to output Q with a delay of one clock cycle.

The device, the diagram of which is shown in fig. 4.14. is called a sequential shift register. The term "sequential" reflects the fact that data is entered into this register bit by bit. For example, to enter the binary combination 0111 into the register you need to go through the entire sequence of states from line 1 to line 6 in the table. 4.5. The sequential loading of the 4-bit combination 0111 into the serial register takes place in 5 cycles (line 2 can be excluded).

Another method of loading the register is parallel (or extended) loading, in which all information bits are entered into the register "on command" of one clock pulse at the same time.

The shift register in fig. 4.14. can be transformed into a 5-bit by adding another D-trigger to the circuit. Shift registers are more often 4-, 5- or 8-bit. They can use not only a D-trigger, but also other types of triggers (for example, a JK-trigger or synchronous RS-triggers).

The sequential shift register, the operation of which is described above, has two significant disadvantages: it allows inputting only one bit of information on each clock pulse and, in addition, the rightmost bit is lost every time it is shifted to the right.

A diagram of a 4-bit parallel ring register is shown in fig. 4.15. Inputs A, B, C, D in this device are informative.

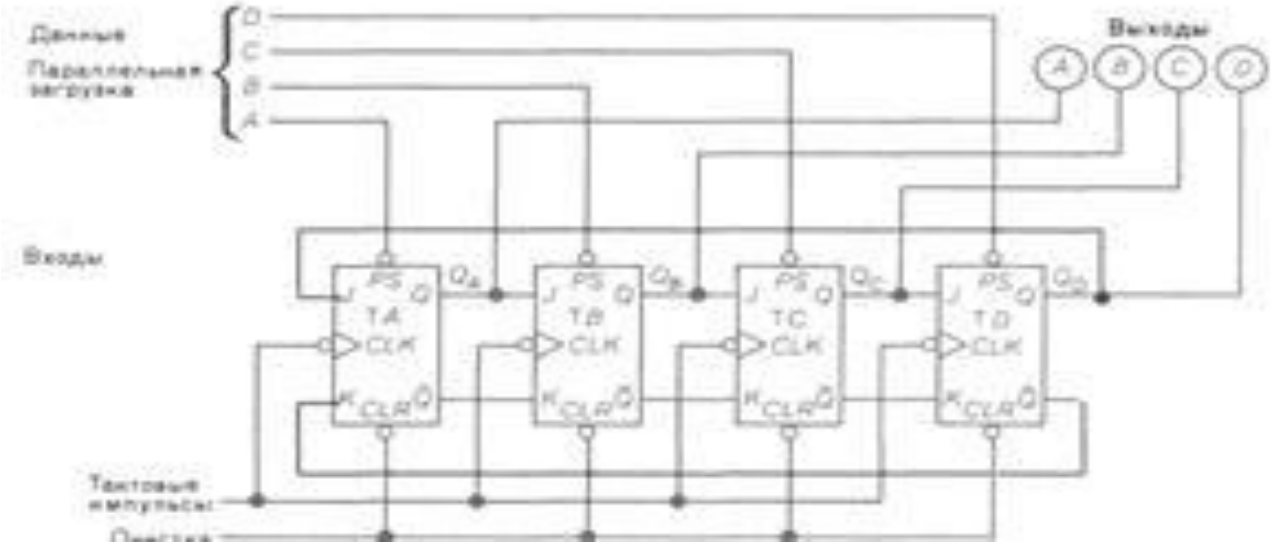


Figure 4.15. 4-bit parallel ring shift register

This system can be equipped with another useful characteristic - the possibility of circular movement of information, when the data from the input of the device is returned to its input and is not lost.

This shift register uses four JK s. It is necessary to pay attention to the feedback of the output of the TD trigger and the inputs J and K of the TA trigger. Due to this feedback circuit, information is entered into the register, which is usually lost at the output of the TD trigger, which will circulate through the shift register. The signal for clearing the register (setting its outputs to the state 0000) is the level of logic zero at the CLR input.

The parallel load inputs A, B, C, D are connected to the PRN trigger presetting inputs, allowing a logic unit level to be set on any output (A, B, C, D). If a logical 0 is applied to one of these inputs, then the corresponding output will be a logical 1. Applying clock pulses to the CLK inputs of all JK triggers causes the information in the register to be shifted to the right. The data is transferred from the TD trigger to the TA trigger (circular movement of information).

Table 4.6. Operation of a 4-bit parallel ring shift register

Input	Output

Line number	Cleaning	Parallel download of data				Clock pulse number	TA	TB	TC	TD
		A	B	C	D		AND	B	C	D
1	1	1	1	1	1	0	1	1	1	0
2	0	1	1	1	1	0	0	0	0	0
3	1	1	0	1	1	0	0	1	0	0
4	1	1	1	1	1	1	0	0	1	0
5	1	1	1	1	1	2	0	0	0	1
6	1	1	1	1	1	3	1	0	0	0
7	1	1	1	1	1	4	0	1	0	0
8	1	1	1	1	1	5	0	0	1	0
9	0	1	1	1	1	0	0	0	0	
10	1	1	0	0	1	0	1	1	0	
11	1	1	1	1	1	6	0	0	1	1
12	1	1	1	1	1	7	1	0	0	1
13	1	1	1	1	1	8	1	1	0	0
14	1	1	1	1	1	9	0	1	1	0
15	1	1	1	1	1	10	0	0	1	1

Table 4.6. helps to understand the principle of operation of the parallel shift register. When the power is turned on, any binary combination is set on the register outputs, such as, for example, as in line 1 of the table. Applying a logic 0 to the CLR inputs of the triggers initiates a register clearing (line 2). Next (line 3) the binary combination 0100 is loaded into the register. Successive clock pulses cause a shift of the entered information to the right (lines 4-8). Pay attention to lines 5 and 6: the unit from the rightmost flip-flop TD is transferred to the leftmost trigger TA. In this case, we can talk about the circular movement of the unit in the register.

Next (line 9), the cleaning of the registry is re-initiated using the CLR input. The new binary combination 0110 (line 10) is loaded. Supplying 5 clock pulses (lines 11-15) results in a circular shift of information 5 positions to the right. It should be noted that 4 clock pulses are required to return the data to the initial state. If the feedback chain is broken in the shift register in fig. 4.15, then we will get a normal parallel shift register; the possibility of circular movement of data will be excluded.

The program for implementing a 4-bit sequential shift register using the AHDL language in the MAX+PLUS II integrated environment looks like this:

```
Title "register1";
Parameters
(WIDTH = 4); -  setting the bit rate of the register
Assert (WIDTH > 0) - checking register bit rate for difference from 0
Report "Value of WIDTH parameter must be greater than %" WIDTH
Severity Error;
Subdesign register1
(
D_INPUT, SET, RESET: input = GND; - incoming signals
CLK: input; - synchronization input
ENABLE :   input =  VCC; - input permission
Q_OUTPUT: output; - output signals
)
Variable
FF [WIDTH..1]: DFFE; -declaration of variable FF, which belongs to DFFE class
Begin
FF []. clk = CLK;
FF[].prn = !SET;
FF[].clrn = !RESET;
FF[].ena = ENABLE;
```

```
FF[].d = (FF[WIDTH-1..1].q, D_INPUT);
Q_OUTPUT = FF[WIDTH] .q;
End;
```

Program for implementation of 4- bit parallel ring shift register using AHDL language in MAX+PLUS II integrated environment looks like this:

```
Subdesign register2
        Parameters
                (WIDTH = 4); - setting the bit rate of the register
                Assert (WIDTH > 0) – checking trigger bit rate (greater than zero)
        Report "Value of WIDTH parameter must be greater than %" WIDTH
        Severity Error;
(
I [WIDTH..1]: input = VCC; - incoming signals ( data )
        CLK: input;
        RESET: input;
        O [WIDTH..1]: output; -output signals ( data )
)
Variable
        FF [WIDTH..1]: JKFF; -declaration of variable FF belonging to JKFF class
Begin
        FF [WIDTH..1].j = (FF[WIDTH-1..1].q, FF[WIDTH].q);
        FF [WIDTH..1].k = (!FF[WIDTH-1..1].q, !FF[WIDTH].q);
        FF [WIDTH..1].clk = CLK;
        FF [WIDTH..1].clrn = !RESET;
        FF [WIDTH..1].prn = I[WIDTH..1];
        O [WIDTH..1] = FF[WIDTH..1].q;
End;
```

The signal editor window of the Register1 project is shown in fig. 4.16.

Fig. 4.16. Test results of a 4-bit sequential shift register

The signal editor window of the Register2 project is shown in fig. 4.17.

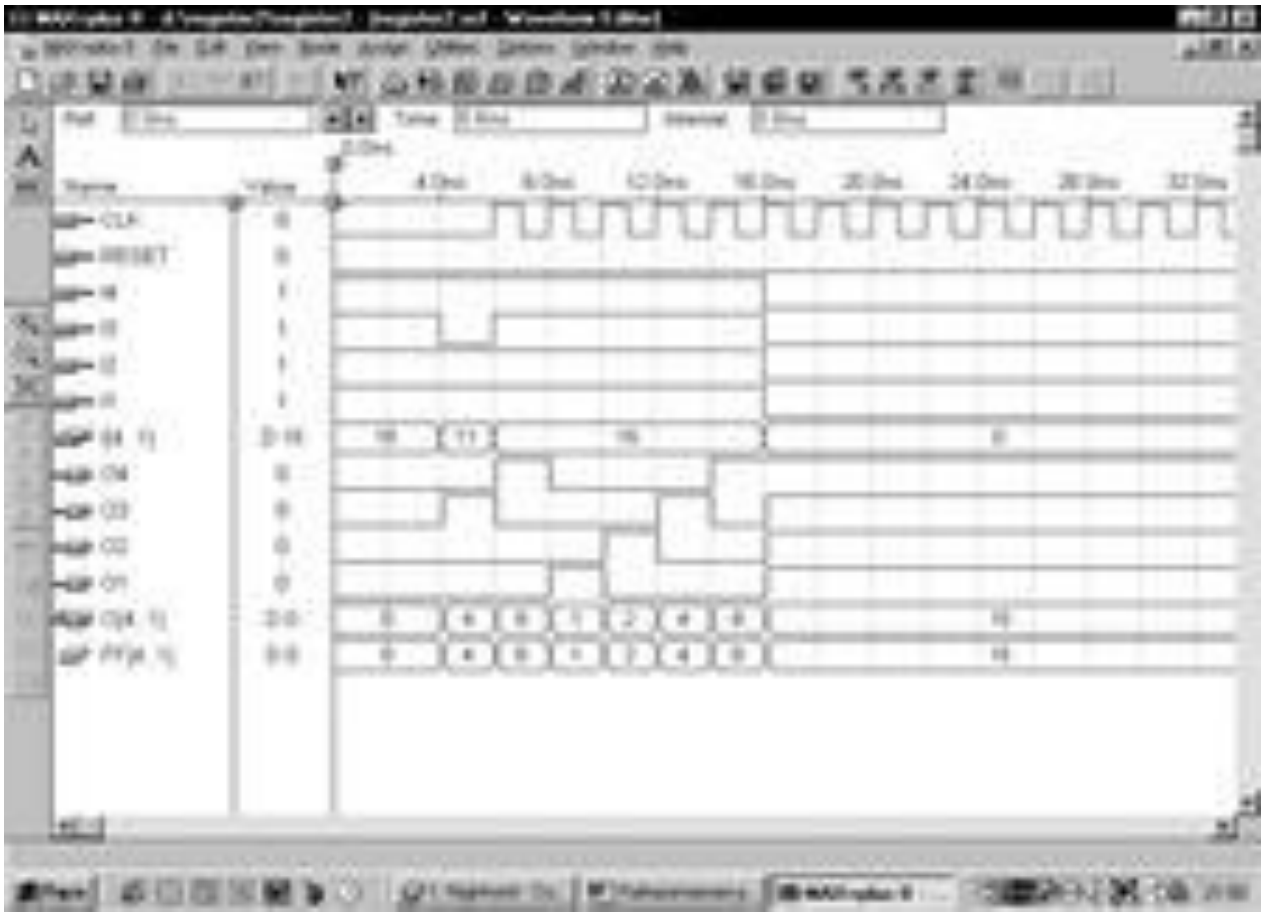


Fig. 4.17. Test results of a 4-bit parallel ring shift register

4.6. Theoretical information about counters

The procedures for binary and decimal numbers are illustrated in the table 4.7. We can count from 0000 to 1111 (0 to 15 in decimal system) using only four binary digits (T4, T3, T2, T1). Column T1 of the table corresponds to the binary digit of the unit or the least significant digit. The term " least significant digit " is usually used. Column T4 corresponds to the binary digit of eights or the most significant digit. The term " most significant digit " is usually used. Note, that the numbers in the units' column change frequently. If you want a counter that counts from 0000 to 1111 in binary system, it must have 16 different output states. Such a counter is called a module 16 counter. The counter module is the number of different states through which the counter passes during one complete counting cycle.

Table 4.7. The counting sequence for the counter according to module 16

A binary number				Decimal number
T4	T3	T2	T1	
8	4	2	1	
0	0	0	0	0
0	0	0	1	1
0	0	1	0	2
0	0	1	1	3
0	1	0	0	4
0	1	0	1	5
0	1	1	0	6
0	1	1	1	7
1	0	0	0	8
1	0	0	1	9
1	0	1	0	10

1	0	1	1	11
1	1	0	0	12
1	1	0	1	13
1	1	1	0	14
1	1	1	1	15

The functional diagram of the module 16 counter, assembled from four JK triggers is shown in fig. 4.18. Skin JK-trigger works in switching mode (J = K = 1). Let the state of the counter outputs correspond to the binary number 0000 at the initial moment of the hour - the cleaning counter. When the clock pulse 1 arrives at the synchronizing input CLK of the trigger T1, this trigger is switched, when the pulse cut -off is passed, and the number 0001 appears on the indicator. Returning to the table. 4.7. we see that the digits (1 or 0) in column T1 (ones) change at each counting step. That is, the T1 trigger is switched with the arrival of a new skin clock pulse. As can be seen from the T2 column, the T2 flip-flop switches half as often as the T1 flip-flop. In general, the skin most significant digit in the table. 4.7. switches twice as often as the previous one.

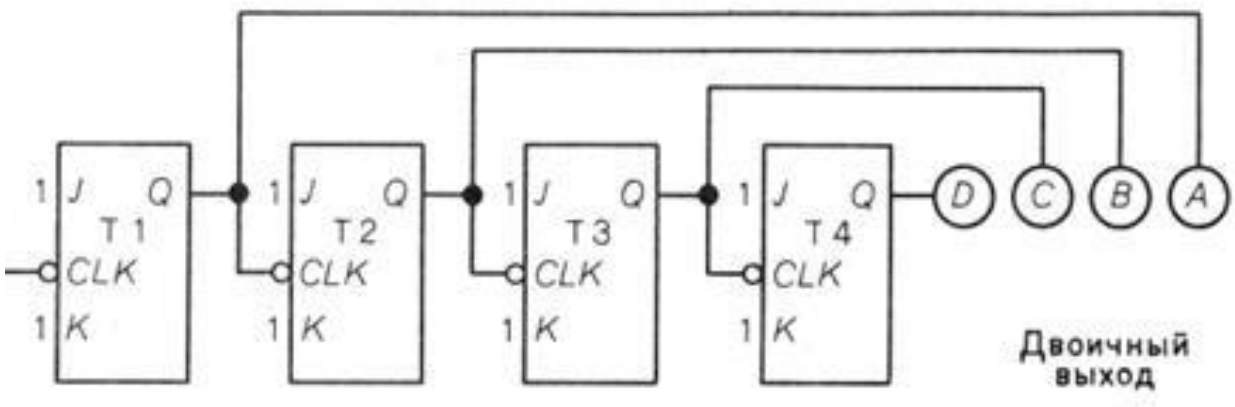


Fig. 4.18. Logical circuit of the counter according to module 16

The operation of the counter according to module 16 is illustrated by the hourly diagrams in fig. 4.18. The top diagram corresponds to the synchronizing input. Diagrams for outputs Q of registers T4, T3, T2, T1 are given below.

Since each trigger affects only the next trigger, it takes several hours to switch all the triggers.

We see that the change of states sequentially passes through the chain of triggers. Therefore, the counter that we are considering is called a pass-through counter.

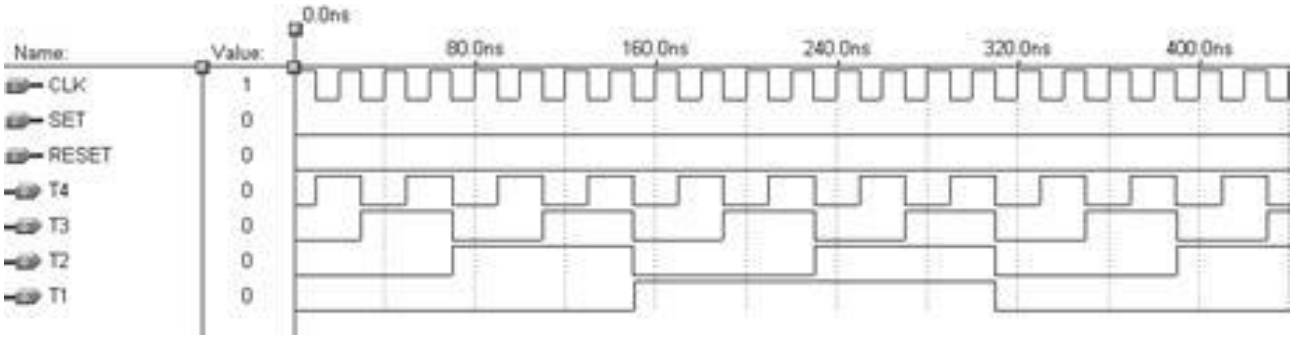


Fig. 4.19. Modulo 16 counter time diagram

The counter, the functional scheme of which is shown in fig. 4.19., can be called not only a pass-through counter, but also a modulo 16 counter, a 4-bit counter or an asynchronous counter. The skin of the names characterizes the considered scheme from one side.

The definitions of " pass-through counter " and "asynchronous" mean that the triggers do not launch at the same hour. The name "module 16 counter" reflects the number of different states that the counter "goes through" in one complete counting cycle. The definition "4-bit" indicates the number of binary digits at the output of the counter.

The counter by module 10 counts from 0000 to 1001 (from 0 to 9 in the decimal system), that is, the first 10 combinations in the table. 4.7. We can see that this requires four binary digits: the ones digit, the twos digit, the foursdigit and the eights digit.

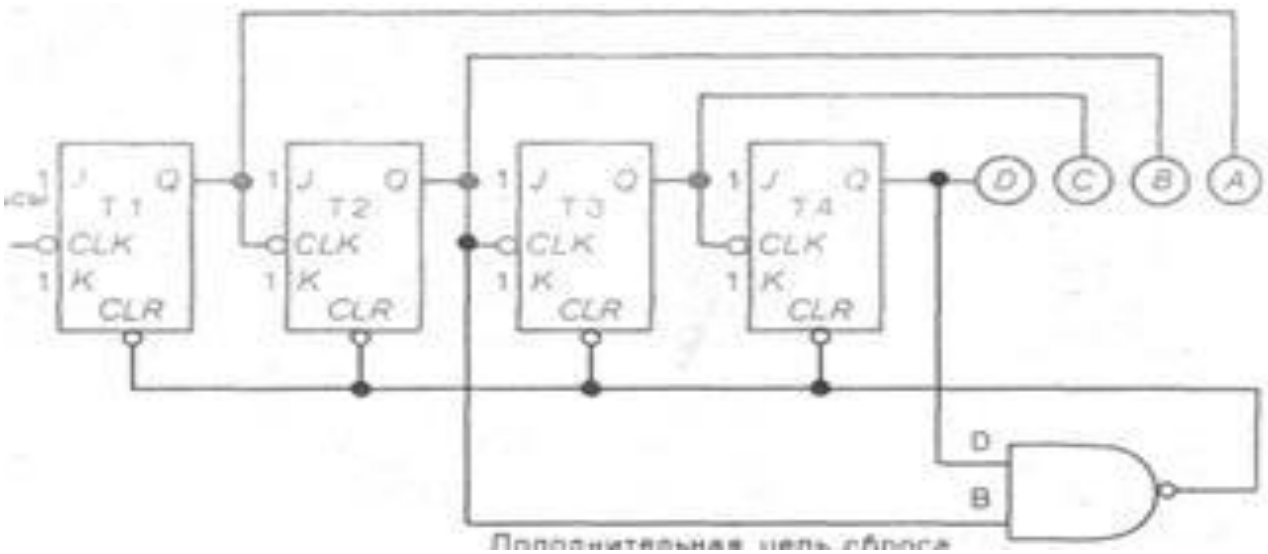


Fig. 4.20. The logical circuit of the counter module 10

Such a counter can be implemented on four triggers connected according to the circuit of the asynchronous counter described above. You need to introduce additionally a logic element AND-NO in the circuit to set all triggers to the zero state, clear the counter, with the arrival of the tenth impulse (that is, with the arrival of the first impulse after the counter has counted up to 1001 - 9 in the decimal system).

The principle of using such a logical element becomes clear if we consider which binary number comes after 1001. It can be seen from the table. 4.7. that this number is 1010 (10 in the decimal system). When presenting a logical 1, contained in twos and eights digits of the binary number 1010, at the input of the AND-NOT logic element, this element sets all triggers to the state 0. The counter will start counting from 0000 to 1001. Thus, the AND-NOT logic element ensures the counter setting to the state 0000. Similar use of the AND-NOT logical element allows you to create counters with some other values of the module. The figure 4.20. shows the functional circuit of an asynchronous counter by module 10. This counter can also be called a decade (decimal) counter.

Counters counting in the forward direction (0, 1, 2,..) were described above. However, there is a need to count backwards (9, 8, 7, 6,..) in some digital systems.

Counters that count from larger numbers to smaller ones are called subtraction counters or reverse counters.

The circuit of the asynchronous subtraction counter according to module 8 is shown in fig. 4.21, the corresponding counting sequence of numbers is given in the table. 4.7.

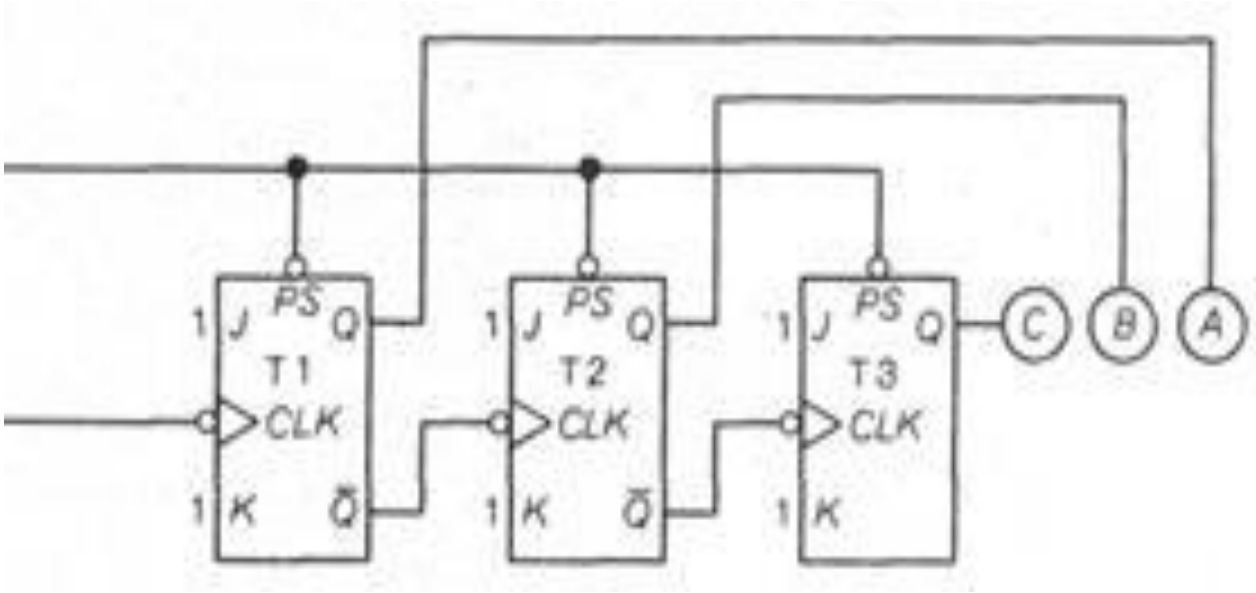


Fig. 4.21. Logical circuit of an asynchronous 3- digit subtraction counter

Table 4.8. Counting sequence for a 3- digit subtraction counter

clock number	Binary counting sequence			Decimal numbers
	T3	T2	T1	
0	1	1	1	7
1	1	1	0	6
2	1	0	1	5
3	1	0	0	4
4	0	1	1	3

5	0	1	0	2
6	0	0	1	1
7	0	0	0	0
8	1	1	1	7
9	1	1	0	6

It should be noted that the circuit of the subtraction counter resembles the circuit of the direct action counter in fig. 4.18. The only difference is the method of transfer from trigger T1 to trigger T2 and from trigger T2 to trigger T3. In a direct-action counter the synchronizing input of the skin flip-flop is connected to the inverse Q output of the previous flip-flop. Note that before starting the countdown in the countdown counter, it is provided for its preliminary setting to state 111 (decimal number 7) using the presetting input (PRN). Trigger T3 is a binary counter of the ones digit (column T1). Trigger T2 is a binary counter (column T2). Trigger T3 is a counter of the digit of fours (column T3).

The program for implementing a 4-bit asynchronous pass-through counter with modulo 16 using the AHDL language in the MAX+PLUS II integrated environment looks like this:

```
title "counter1";
        parameters
                (WIDTH = 4); - setting counter capacity
        assert (WIDTH > 0) - checking counter capacity
        report "Value of WIDTH parameter must be greater than %" WIDTH
        severity error;
        subdesign counter1
(
```

```
        CLK: input; - counter synchronization impulses (CLK)
        RESET: input; - counter triggers reset impulses (CLRN)
         SET: input; - impulses preset (PRN) counter triggers
        O [WIDTH..1]: output; - counter output
)
variable
        TRIGGER [WIDTH..1]: JKFF;
begin
        TRIGGER [WIDTH..1].j = vcc;
        TRIGGER [WIDTH..1].k = vcc;
        TRIGGER [WIDTH..1].clrn = !RESET;
        TRIGGER [WIDTH..1].prn = !SET;
        TRIGGER [WIDTH].clk = !CLK;
        TRIGGER [WIDTH-1..1].clk = !TRIGGER [WIDTH..2].q;
        O [WIDTH..1] = TRIGGER [1..WIDTH].q;
end;
```

The program for implementing an asynchronous counter by module 10 using the AHDL language in the MAX+PLUS II integrated environment looks like this:

```
title "counter2";
        parameters
                (WIDTH = 4); - setting counter capacity
                assert (WIDTH > 0) - checking counter capacity
                report "Value of WIDTH parameter must be greater than %" WIDTH
        severity error;
        subdesign counter2
(
```

```
        CLK: input; counter synchronization impulses (CLK)
         SET: input; - preset pulses (PRN) counter triggers
        O [WIDTH..1]: output; - counter output
)
variable
        TRIGGER [WIDTH..1]: JKFF;
begin
        TRIGGER [WIDTH..1].j = vcc;
        TRIGGER [WIDTH..1].k = vcc;
        TRIGGER [WIDTH..1].prn = !SET;
        TRIGGER [WIDTH].clk = !CLK;
        TRIGGER [WIDTH-1..1].clk = !TRIGGER [WIDTH..2].q;
        TRIGGER [WIDTH..1].clrn = (TRIGGER [3].q! & TRIGGER [1].q);
        O [WIDTH..1] = TRIGGER [1..WIDTH].q;
end;
```

Program for implementation of 3- bit counter subtraction using AHDL languages in integrated environment MAX+PLUS II looks like this:

```
title "counter3";
        parameters
                (WIDTH = 3); - setting counter capacity
                assert (WIDTH > 0) - checking counter capacity
                report "Value of WIDTH parameter must be greater than %" WIDTH
                severity Error;
        subdesign counter3
(
        CLK: input; - synchronization impulses of the counter(CLK)
        RESET: input; - counter triggers reset impulses (CLRN)
        SET: input; - preset pulses (PRN) of counter triggers
        O[WIDTH..1]: output; - counter output
```

```
)
variable
        TRIGGER[WIDTH..1]: JKFF;
begin
        TRIGGER [WIDTH..1].j = vcc;
        TRIGGER [WIDTH..1].k = vcc;
        TRIGGER [WIDTH..1].clrn = !RESET;
        TRIGGER [WIDTH..1].prn = !SET;
        TRIGGER [WIDTH..1].clk = (CLK, !TRIGGER [WIDTH..2].q);
        O [WIDTH..1] = !TRIGGER [1..WIDTH].q;
end;
```

The program for implementing a 3-digit universal counter using the AHDL language in the MAX+PLUS II integrated environment looks like this:

```
        title "counter4";
        parameters
                (WIDTH = 3); - setting counter capacity
                assert (WIDTH > 0) - checking counter capacity
                report "Value of WIDTH parameter must be greater than %" WIDTH
                severity Error;
        subdesign counter4
(
        CLK: input; - synchronization impulses of the counter(CLK)
        SET: input; - preset pulses of counter triggers(PRN)
        RESET: input; - reset pulses of counter triggers(CLRN)
        FWC: input;
        BWC: input;
        O[WIDTH..1]: output; - counter output
)
variable
```

```
        TRIGGER [WIDTH..1]: JKFF;
begin
        TRIGGER [].j = vcc;
        TRIGGER [].k = vcc;
        TRIGGER [].prn = !SET;
        TRIGGER [].clrn = !RESET;
        TRIGGER[WIDTH].clk = !CLK;
        TRIGGER[WIDTH-1..1].clk = !((TRIGGER[WIDTH..2].q & !FWC) &
        !(!TRIGGER[WIDTH..2].q & !BWC));
        O[WIDTH..1] = TRIGGER[1..WIDTH].q;
end;
```

Note:

- Values of the inputs corresponding to the setting the counter counting in
- increasing direction:
- FWC = 1;
- BWC = 0.
- Values of the inputs corresponding to the setting of the counter count in
- descending direction:
- FWC = 0;
- BWC = 1.

The signal editor window of the counter1 project is shown in fig. 4.22.

Figure 4.22. Test results of a 4- digit asynchronous counter with pass-through modulo 16

The signal editor window of the counter2 project is shown in fig. 4.23.

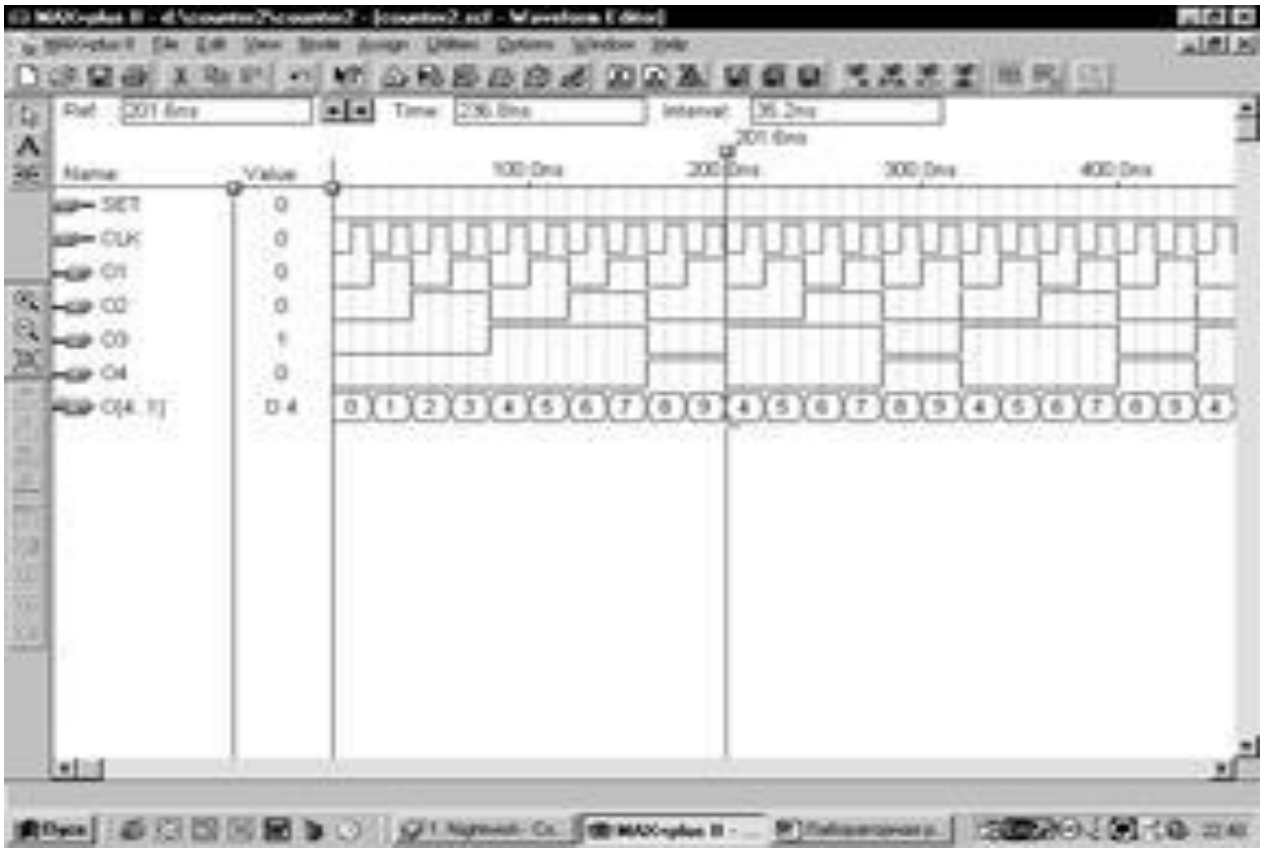


Fig. 4.23. The results of testing the asynchronous counter according to module 10

The signal editor window of the counter3 project is shown in fig. 4.24.

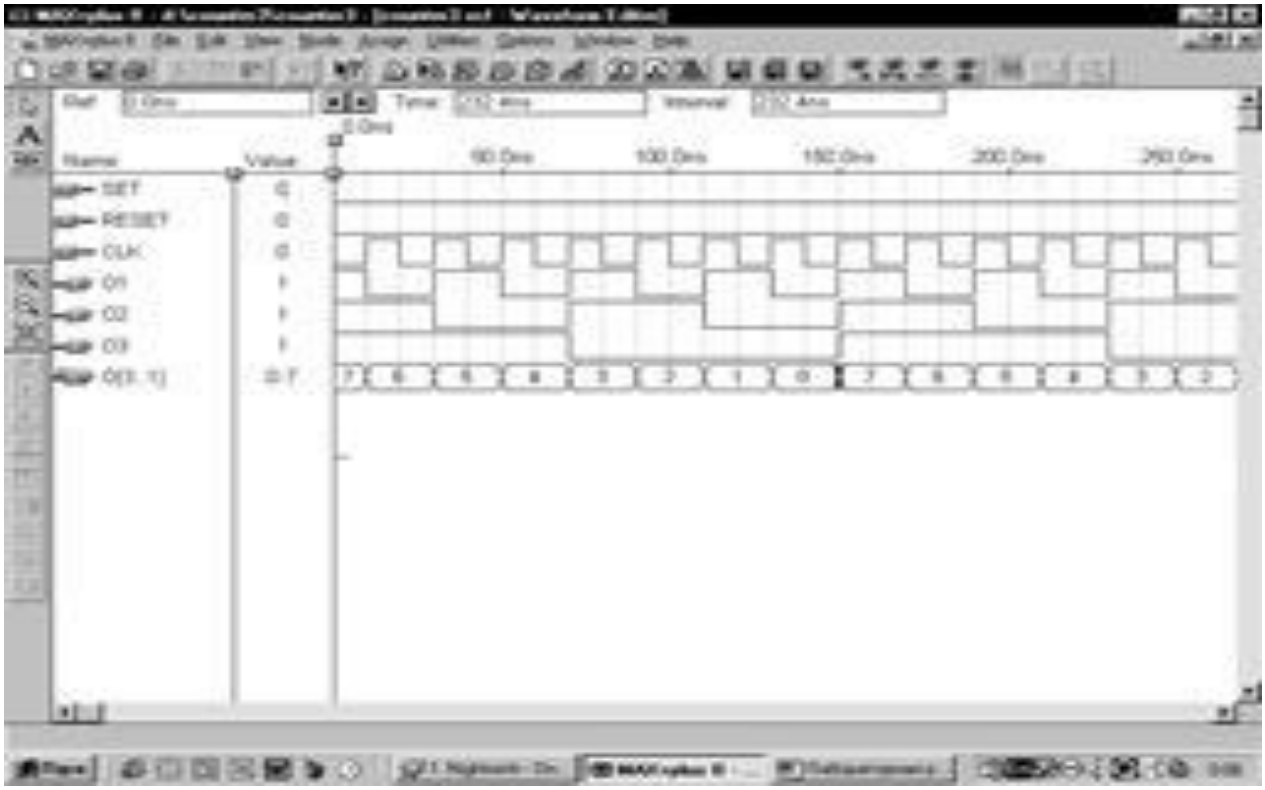


Fig. 4.24. Test results of a 3- digit synchronous counter

The signal editor window of the counter4 project is shown in fig. 4.25.

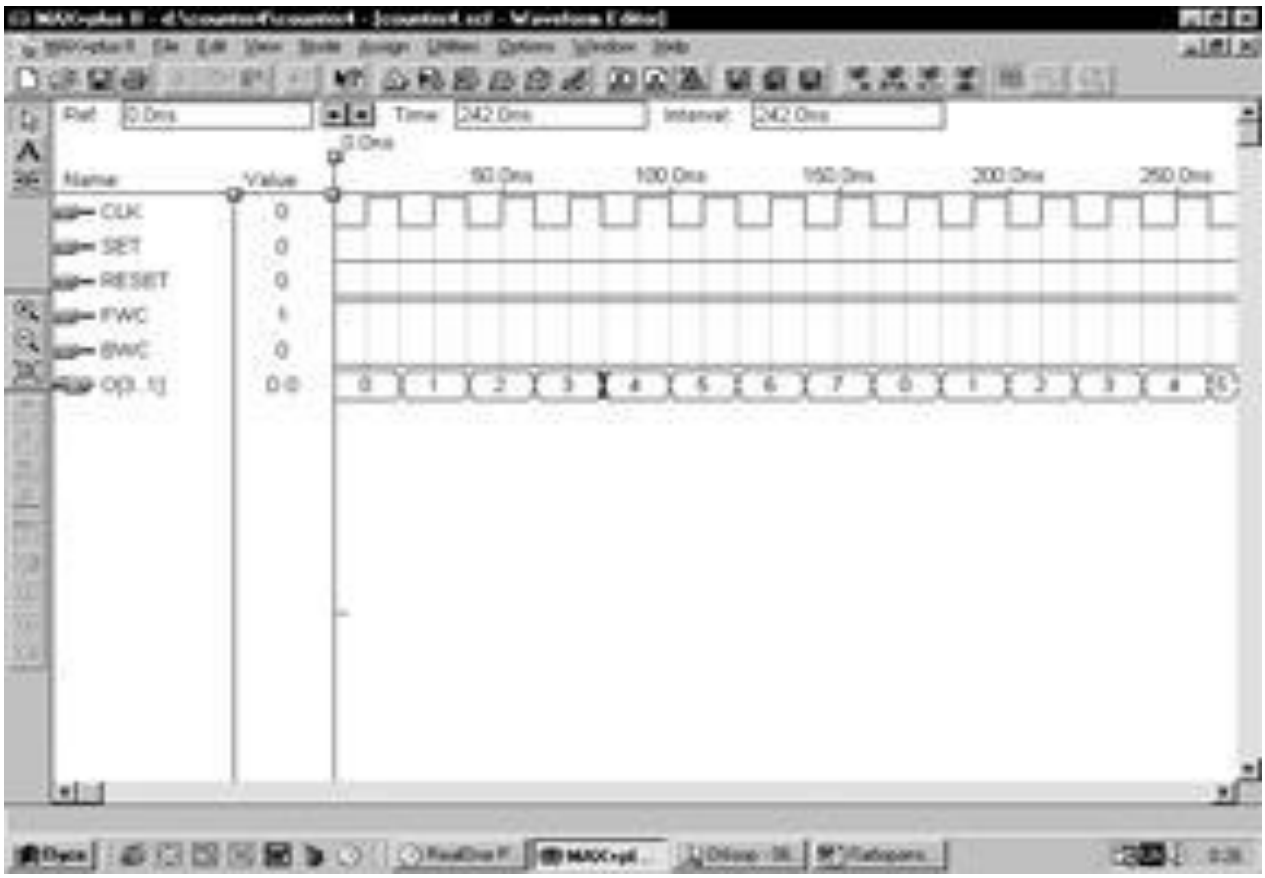


Fig. 4.25. Test results of a 3-digit universal counter

4.7. Theoretical information about multiplexers, demultiplexers, encoders and decoders

A multiplexer is a combinational logical device designed to control the transmission of data from several sources to one output channel.

A typical use of multiplexers is the transmission of data from several spatially scattered information sources to the input of the receiver. Suppose the ambient temperature is measured in several rooms and the results of these measurements should be displayed on a computer. Moreover, since the temperature changes slowly, it is not necessary to measure it constantly to obtain sufficient accuracy. It is enough to have measurements carried out at some fixed intervals of an hour. The main thing is that the intervals between the two measurements should be significantly less than a constant hour, which characterizes the temperature change in the room being

monitored. It is this function, that is, the connection of different sources of information to one receiver according to a given command, that the multiplexer performs. It transforms information scattered in space into an image.

According to its purpose, the multiplexer should have one output and two groups of inputs: informational and addressable. The code applied to the address inputs determines which of the information inputs is currently connected to the output pin. Since an n-bit binary code can take 2^n values, then if the number of address inputs of the multiplexer is n, then the number of its information inputs should be $2^{n.}$.

The truth table showing the operation of a multiplexer with two addressable inputs looks like this (Table 7.4.1).

Table 4.9. Truth table for a multiplexer with two addressable inputs

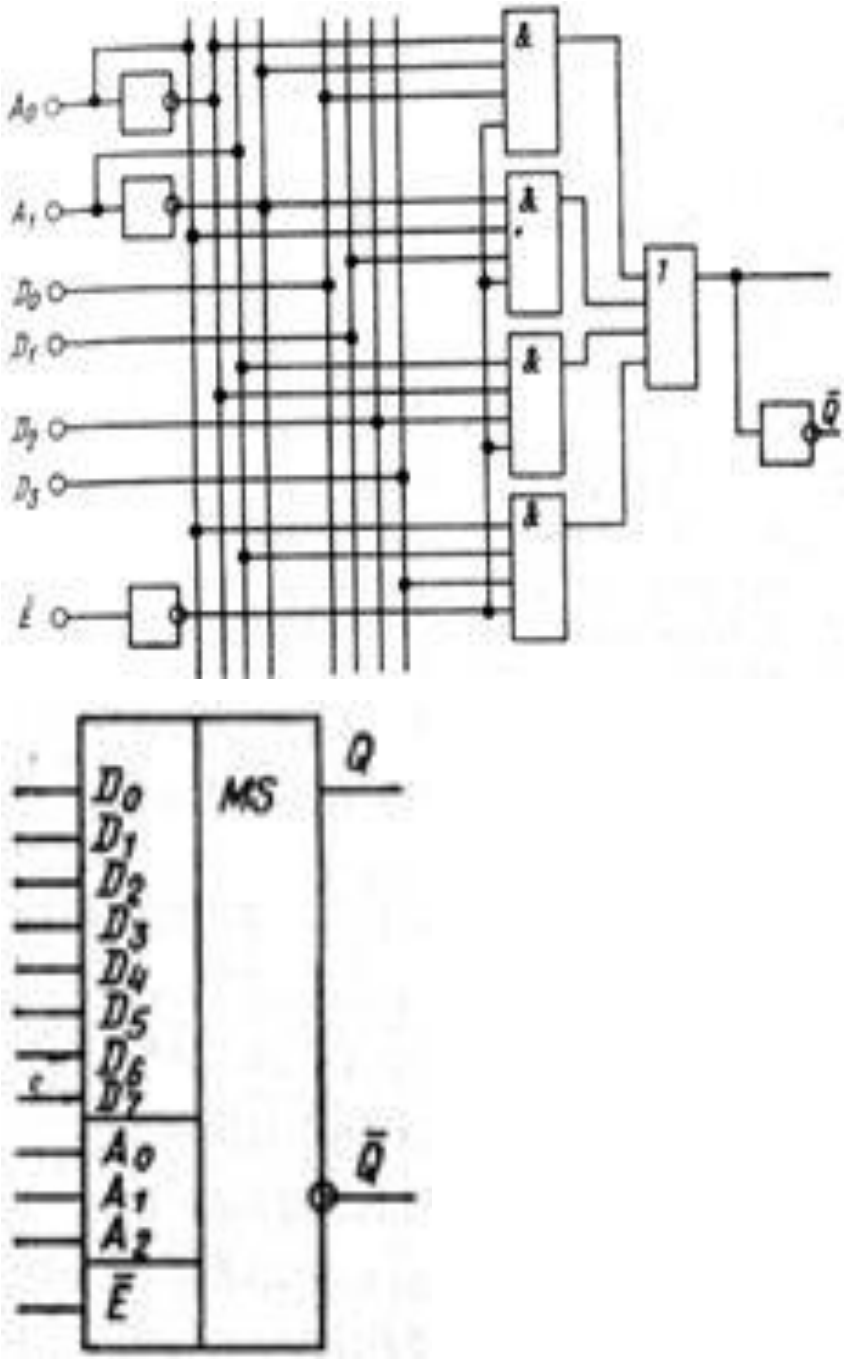
E	A_1	A_0	Q	$\overline{\cap}$
1	h	x	0	1
0	0	0	D_0	$\overline{D}_{\wedge}$
0	0	1	D_1	$\overline{D}.$
0	1	0	D_2	$\overline{D}_{\backslash}$
0	1	1	D_3	$\overline{D}_{\backslash}$

It is taken into account in this table that the multiplexer usually has an additional inverse output $\overline{Q}$ and input of operation permission E (in programs in the AHDL language, the input of operation permission based on the example of primitive triggers has the name ENA). If an active logic signal (E=1) is applied to the enable input E, the output signal of the multiplexer is constant and does not depend on its input signals.

The logic algebra function describing the operation of the multiplexer has the form:

$$O = D \cdot \overline{A} \cdot \overline{A} \cdot \overline{E} + D \cdot \overline{A} \cdot A \cdot \overline{E} + D \cdot A \cdot \overline{A} \cdot \overline{E} + D \cdot A \cdot A \cdot \overline{E}$$

The logic circuit of the multiplexer corresponding to the given function of the algebra logic and the conventional symbol lof the multiplexer on the example of IC (integrated circuit) 555KP7 is shown in fig. 4.27. a, b.



a) b)

Fig. 4.27. The logical circuit of the multiplexer (a) and its conventional graphic symbol (b)

When transmitting information from several sources on a regular channel with time distribution, not only multiplexers are needed, but also reverse-purpose devices that distribute information received from one channel between several receivers. This task is solved by demultiplexers.

The multiplexer can be described in two ways in the MAX+PLUS II integrated environment using the AHDL language:

1) a truth table;

2) at the behavioral level.

The description of the device using the truth table is the simplest because it requires the designer to know only the truth table of the multiplexer. The volume of the resulting program is significantly smaller in size compared to the volume of the description program at the behavioral level, but the architecture (logical circuit) of the device itself remains unknown to the designer.

The specialist chooses the description method based on the technical specification, the given scope of the program, the number of elementary gates on the microcircuit and his own experience.

The examples of the description of the multiplexer, which has two addressable, four informational inputs and one operation permission input are given both using a truth table and at the behavioral level.

A demultiplexer is a combinational logical device designed to control data transmission from one source of information to several output channels. A demultiplexer, in general, has one information input, n address inputs and 2^n outputs according to the definition. The truth table describing the operation of the demultiplexer with two address inputs and the operation enable input E looks like this (Table 4.10):

Table 4.10. Truth table for a demultiplexer with two addressable inputs

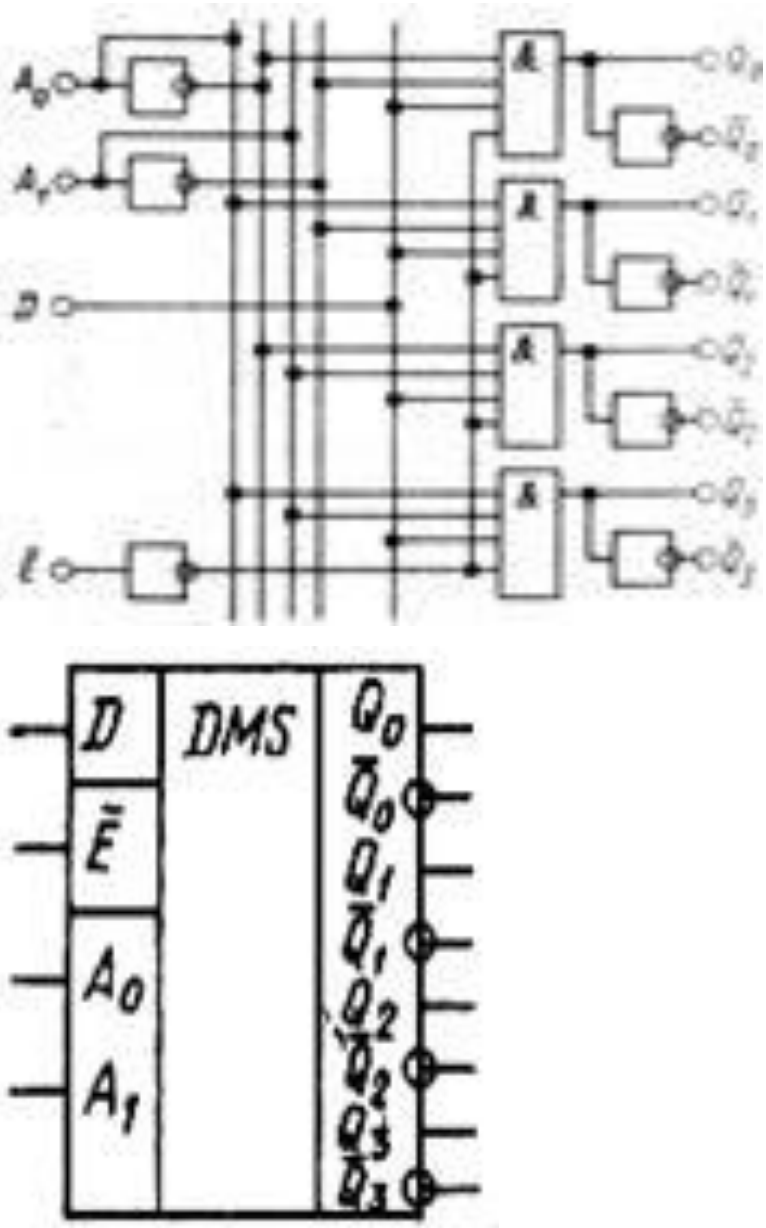
E	A_1	A_0	Q_0	Q1 _	Q_2	Q3 _
1	x	X	0	0	0	0

0	0	0	D	0	0	0
0	0	1	0	D	0	0
0	1	0	0	0	D	0
0	1	1	0	0	0	D

The following logical algebra function corresponds to this table:

$$Q_0 = D\overline{A}_1\overline{A}_0\overline{E} = \overline{D} \downarrow A_1 \downarrow A_0 \downarrow E,$$
$$Q_1 = D\overline{A}_1 A_0\overline{E} = \overline{D} \downarrow A_1 \downarrow \overline{A}_0 \downarrow E,$$
$$Q_2 = D A_1\overline{A}_0\overline{E} = \overline{D} \downarrow \overline{A}_1 \downarrow A_0 \downarrow E,$$
$$Q_3 = D A_1 A_0\overline{E} = \overline{D} \downarrow \overline{A}_1 \downarrow \overline{A}_0 \downarrow E. \quad (2)$$

The figure 4.28.a) shows a logical circuit of the demultiplexer that satisfies the functions of the logic algebra (2), and the figure 4.28 b) shows its conventional graphic image.



o

Fig. 4.28. The logic circuit of the demultiplexer (a) and its conventional graphic image (b)

A combinational logical device for converting numbers from the decimal system to binary is called an encoder or coder. Encoder inputs are sequentially assigned the values of decimal numbers, so the application of an active logical signal to one of the inputs is perceived by the encoder as the application of a corresponding decimal number. This signal is converted to the output of the encoder in a binary code. According to the above, if an encoder has n outputs, then its inputs should be

no more than 2^n. An encoder that has 2^n inputs and n outputs is called complete. If the number of inputs of the encoder is less than 2^n, then it is called incomplete.

Let's consider the operation of the encoder using the example of a converter of decimal numbers from 0 to 9 into a binary-decimal code. The truth table corresponding to this case looks like this: (Table 4.11).

Since the number of inputs of this device is less than $2^n = 16$, we have an incomplete encoder. Using the table for Q_3, Q_2, Q_1, Q_0, you can write the following expressions:

$$Q_3 = x_8 + x_9;$$

$$Q_2 = x_4 + x_5 + x_6 + x_7;$$

$$Q_1 = x_2 + x_3 + x_6 + x_7;$$

$$Q0 = x1 + x3 + x5 + x7 + x9$$

Table 4.11. Truth table for of the decimal numbers from 0 to 9 to binary-decimal code

x_9	x_8	x_7	x_6	x_5	x_4	x_3	x_2	x_1	x_0	Q3 _	Q_2	Q1 _	Q_0
0	0	0	0	0	0	0	0	0	1	0	0	0	0
0	0	0	0	0	0	0	0	1	0	0	0	0	1
0	0	0	0	0	0	0	1	0	0	0	0	1	0
0	0	0	0	0	0	1	0	0	0	0	0	1	1
0	0	0	0	0	1	0	0	0	0	0	1	0	0
0	0	0	0	1	0	0	0	0	0	0	1	0	1
0	0	0	1	0	0	0	0	0	0	0	1	1	0
0	0	1	0	0	0	0	0	0	0	0	1	1	1
0	1	0	0	0	0	0	0	0	0	1	0	0	0
1	0	0	0	0	0	0	0	0	0	1	0	0	1

The resulting system (3) characterizes the operation of the encoder. The logical diagram of the device corresponding to system (3) is shown in fig. 4.29.

It is easy to see that in this type of encoder, the signal applied to the $x_{0\ input}$ is not used. Therefore, the absence of a signal at any input $x_0 x_1$ is interpreted by the circuit as an obvious zero signal.

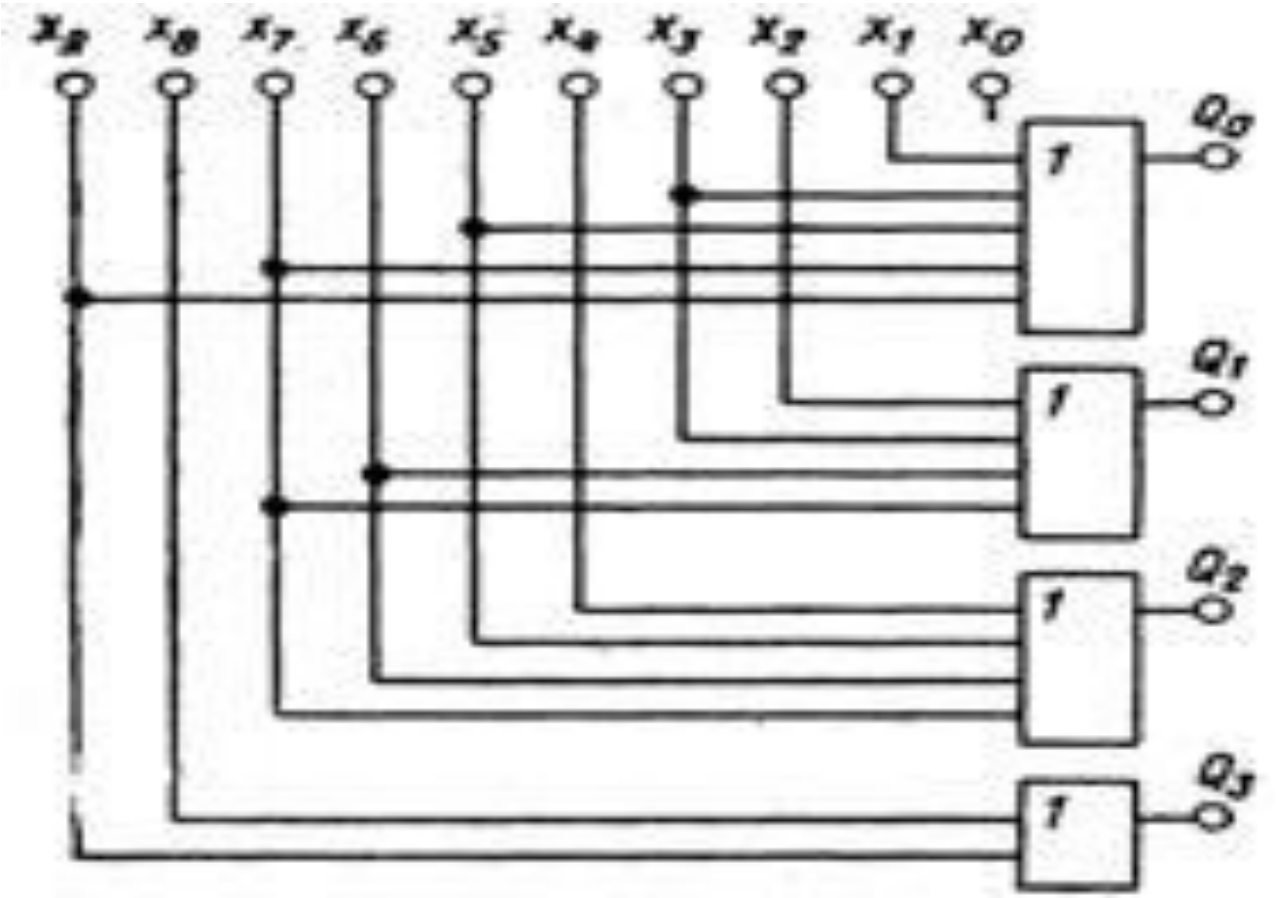


Fig. 4.29. The decimal encoder logic circuit

The main direction of using the encryptors in digital systems is the input of initial information from the keyboard.

When any key is pressed, a "logical unit" signal is applied to the corresponding input of the encoder, which is then converted into a binary-decimal code. The option of entering information is shown in fig. 4.30.

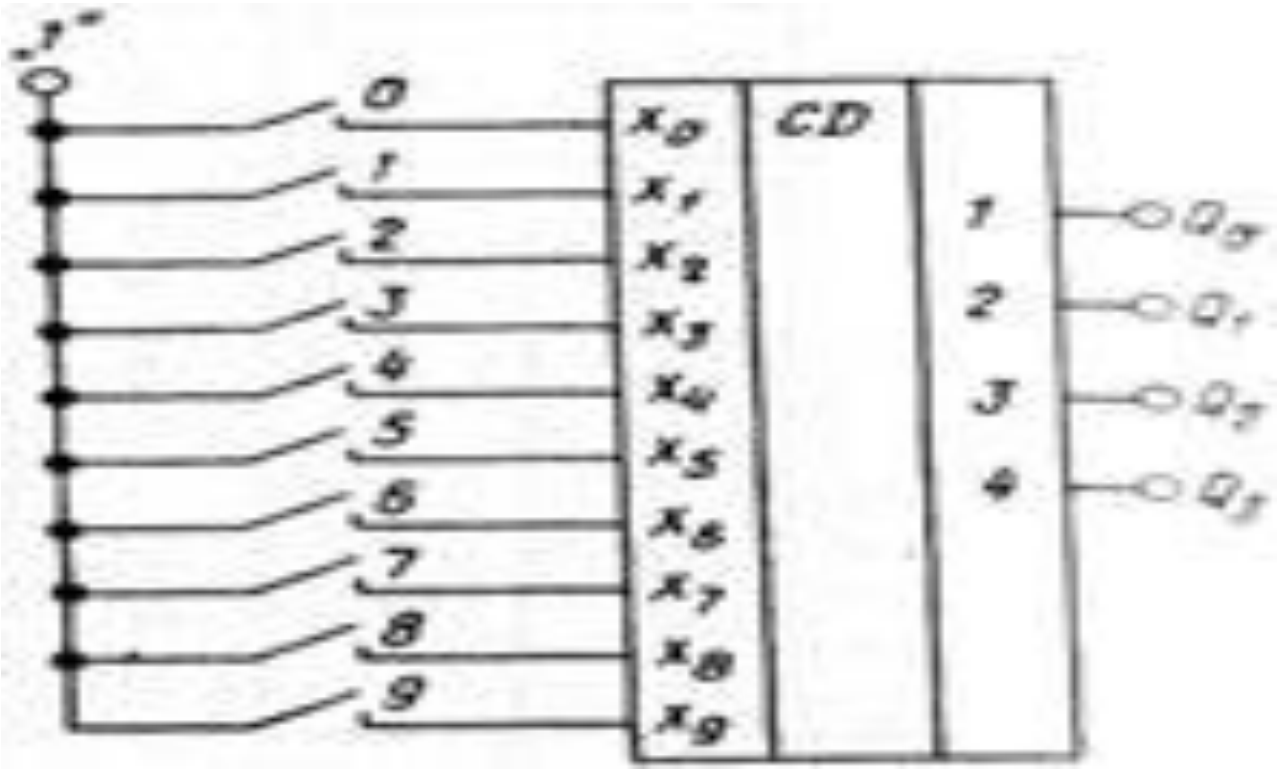


Fig. 4.30 Keyboard input device

The encoder can be described in two ways in the MAX+PLUS II integrated environment using the AHDL language:

- a truth table emulated using the CASE operator;
- at the behavioral level.

The description of the device emulated using the CASE operator is the simplest, because it requires the designer to know only the encoder's truth table. The volume of the resulting program is significantly smaller in size compared to the volume of the description program at the behavioral level, but the architecture (logical circuit) of the device itself remains unknown to the designer.

The specialist chooses the method of description based on the technical specifications, the given scope of the program, the number of elementary gates on the microcircuit and his own experience.

A decoder is a combinational logical device for converting numbers from the binary system to the decimal one. According to the definition, the decoder belongs to the class of code converters. It is clear that each binary number is matched with a signal generated at the output of the device. Thus, the decryptor performs the reverse

operation of the encoder. If the number of address inputs of the decoder is related to the number of its outputs m by the ratio $m = 2^n$, then the decoder is called complete. In the opposite case, if $m < 2^n$, the decoder is called incomplete.

The behavior of the decoder is described by a truth table, similar to the truth table of the encoder (see system 3), but in this table the input and output signals are reversed. According to this table, since the output signal is equal to 1 only on one, single set of input variables, that is, for one constitution unit, the decoder operation algorithm is described by such a system of equations:

$$x_0 = \overline{Q}_3 \overline{Q}_2 \overline{Q}_1 \overline{Q}_0;$$

$$x_1 = \overline{Q}_3 \overline{Q}_2 \overline{Q}_1 Q_0;$$

$$x_2 = \overline{Q}_3 \overline{Q}_2 Q_1 \overline{Q}_0;$$

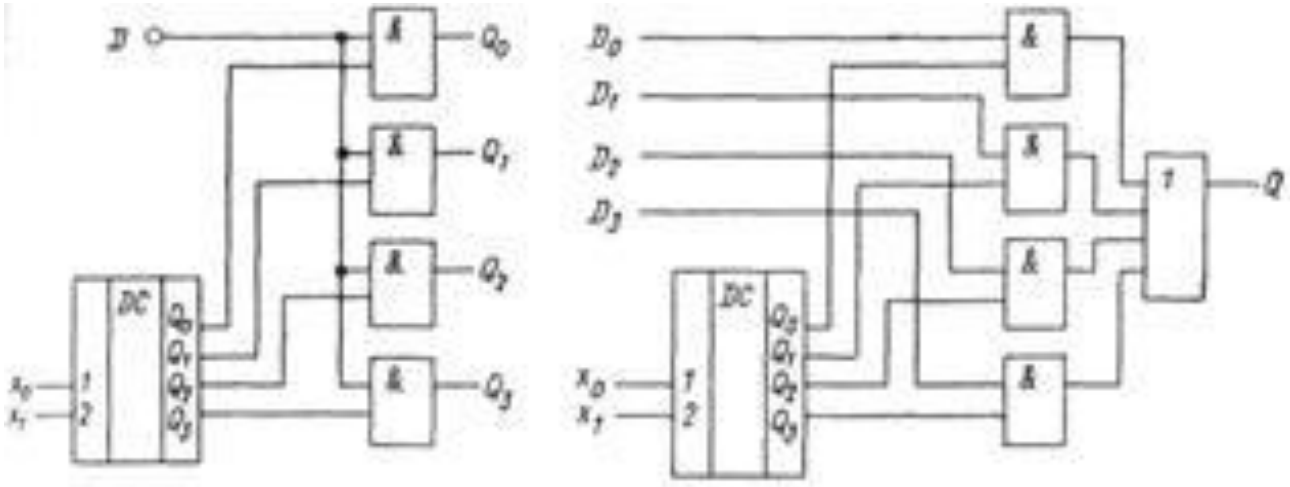
etc, where Q_i is the value of the logical variable at the i-th input of the device.

In general, the system (4) has the form:

$$x_i = (Q_3 Q_2 Q_1 Q_0)_i$$

where, x_i- is the signal at the i-th output of the encoder; $(Q_3 Q_2 Q_1 Q_0)_i$ is the constituent of the unit corresponding to the binary code of the i-th decimal digit.

It is not difficult to notice that the function of the algebraic logic of the decoder differs from the function of the algebraic logic of the demultiplexer only by the presence in the latter of an additional multiplier that corresponds to the value of the signal at the information input D. Therefore, when D = 1, the demultiplexer functions as a decoder. The inverse conversion of the decoder into a demultiplexer requires the introduction of two auxiliary logical elements AND, which perform a logical multiplication operation between the common signal of the information input D and the corresponding logical result of multiplying the address signals ($Q_3 Q_2 Q_1 Q_0$). A multiplexer circuit can be also built using the decoder. To do this, the circuit from fig.4.31.a, must be supplemented with four output logical OR elements (fig.4.31,b).



o b)

Fig. 4.31. Implementation of demultiplexer (a) and multiplexer (b) using a decoder

When developing integrated circuits, several decoder logic structures are used. Their main difference lies in the speed of operation and the number of elementary logic elements used.

The fastest and at the same time the most complex is the decoder that directly implements the system of algebraic functions of logic (4). Such a decoder is called single-stage or parallel. Its block diagram is similar to that of a demultiplexer under the condition D = 1.

Assuming that some conventional unit of hardware is required to implement the processing of one input logic signal, the number of units of these hardware for an n-bit decoder is determined by the expression:

$$N_1 = n2^{n.}$$

A conventional graphic designation of the decoder is given in fig. 4.32. It corresponds to the integrated circuit of the binary-decimal decoder type 564ID1.

If the main design requirement is the simplicity of the system solution, other structural diagrams of decoders are used. However, the simplification of the structure is achieved at the expense of a decrease in speed of performance.

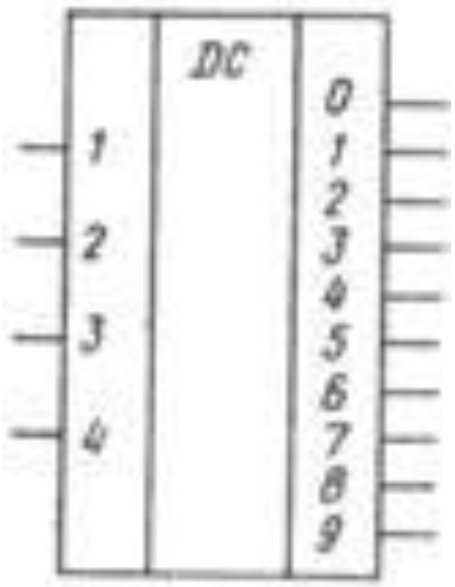


Fig. 4.32. Conventional graphic designation of the decoder

Decoder chips often have an enable input E (gate input). The presence of this input allows on the basis of ready-made integrated circuits, to create decoder tree structures if it is necessary to increase the bit rate of the input code.

The program for implementing a 10-by-4 encoder (the description of the truth table of the encode) using the AHDL language in the MAX+PLUS II integrated environment looks like this:

```
Subdesign cipher1
        (
XIP [9. .0]: input; - incoming signals
QOP [3.. 0]: output; -  output signals
        )
Begin
        Table
                XIP [] => QOP[];
                b"0000000001" => b"0000";
                b"0000000010" => b"0001";
                b"0000000100" => b"0010";
```

```
                b"0000001000" => b"0011";
                b"0000010000" => b"0100";
                b"0000100000" => b"0101";
                b"0001000000" => b"0110";
                b"0010000000" => b"0111";
                b"0100000000" => b"1000";
                b"1000000000" => b"1001";
        End table;
End;
```

The program for implementing a 10-by-4 encoder (description at the behavioral level of the encoder) using the AHDL language in the MAX+PLUS II integrated environment looks like this:

```
        Subdesign cipher2
                (
            XIP [9..0]: input; - input signals
            QOP [3..0]: output; - output signals
                )
        Begin
            QOP [3] = XIP [8] + XIP[9];
            QOP [2]= XIP[4] + XIP[5] + XIP[6] + XIP[7];
            QOP 1]= XIP[2] + XIP[3] + XIP[6] + XIP[7];
            QOP [0]= XIP[1] + XIP[3] + XIP[5]+ XIP[7] + XIP[9];
        End;
```

The program for implementing a 3-bit decoder with inverse outputs using the AHDL language in the MAX+PLUS II integrated environment looks like this:

```
        subdesign decipherer1
        (
```

```
        XIP [3..1]: input; - incoming signals
        QOP [7..0]: output; - output signals
)
begin
        case XIP [] is
                when 0 => QOP = b"11111110";
                when 1 => QOP = b"11111101";
                when 2 => QOP = b"11111011";
                when 3 => QOP = b"11110111";
                when 4 => QOP = b"11101111";
                when 5 => QOP = b"11011111";
                when 6 => QOP = b"10111111";
                when 7 => QOP = b"01111111";
        end case;
end;
```

The program for implementing a multiplexer with 2 address inputs, 4 information inputs and an enable input (description by an emulated multiplexer truth table) using the AHDL language in the MAX+PLUS II integrated environment looks like this:

```
subdesign multiplexer1
(
        INFIN [4..1]: input; - information inputs
        ADRIN [2..1]: input; - addressed inputs
         ENA: input; - enable input (input gate)
        Q: output; - Multiplexer output
)
begin
        if ENA == 0 then - truth table emulation
```

```
            case ADRIN [2..1] is
                when 0 => Q = INFIN [1];
                when 1 => Q = INFIN [2];
                when 2 => Q = INFIN [3];
                when 3 => Q = INFIN [4];
            end case;
        end if;
    end;
```

Note: The AHDL compiler does not allow the presence of
- of variables (parameters) in validity tables even if
- variables (parameters) have previously been assigned fixed
- the meaning. Therefore, according to the logic of the truth table, based on
- to the CASE selection operator, a sequence for checking is formed
- value of input signals of the system.

The program for implementing a multiplexer with 2 address inputs, 4 information inputs, and an enable input (description at the behavioral level of the multiplexer) using the AHDL language in the MAX+PLUS II integrated environment looks like this:

```
subdesign multiplexer2
    (
        INFIN [4..1]: input; - information inputs
        ADRIN [2..1]: input; - address inputs
        ENA: input; - enable input (gate input)
        Q: output; - Multiplexer output
)
begin
    Q = INFIN [1] & !ADRIN[2] & !ADRIN[1] & !ENA #
    INFIN [2] & !ADRIN [2] & ADRIN[1] & !ENA #
```

```
        INFIN [3] & ADRIN [2] & !ADRIN[1] & !ENA #
        INFIN [4] & ADRIN [2] & ADRIN [1] & !ENA;
end;
```

Note: Q is a logical algebra function that describes the operation of the multiplexer.

The program for implementing a demultiplexer with 3 address inputs, 1 information input, and an enable input using the AHDL language in the MAX+PLUS II integrated environment looks like this:

```
subdesign demultiplexer1
        (
                ADRIN [3..1]: input; - address input
                INFIN: input; - information input
                ENA: input; - enable input (gate input)
                Q [7..0]: output; - Demultiplexer output
        )
begin
        if ENA == 0 then
                case ADRIN [] is
                        when 0 => Q [0] = INFIN;
                        when 1 => Q [1] = INFIN;
                        when 2 => Q [2] = INFIN;
                        when 3 => Q [3] = INFIN;
                        when 4 => Q [4] = INFIN;
                        when 5 => Q [5] = INFIN;
                        when 6 => Q [6] = INFIN;
                        when 7 => Q [7] = INFIN;
                        end case;
        end if;
```

end;

The window of the signal editor of the Shifrator1 project is given in fig. 4.33.

Fig. 4.33. 10 by 4 encoder testing results

The signal editor window of the "decipherer1" project is shown in fig. 4.34.

Fig. 4.34. Test results of a full 3-bit decoder with inverse outputs

The signal editor window of the multiplexer2 project is given in fig. 4.35

Fig. 4.35. Test results of a multiplexer with 2 address inputs, 4 information inputs and an operation enable input

The signal editor window of the “demultiplexer1” project is shown in fig. 4.36.

Fig. 4.36. Test results of a demultiplexer with 3 address inputs, 1 information input and an enable input

4.8. Theoretical information about adders and subtractors

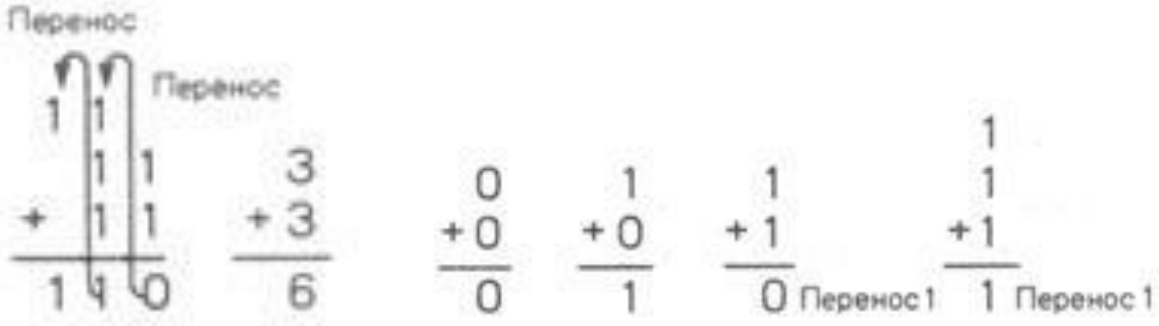
The leftmost digit of a binary number, such as 101011, is called the most significant digit (MSD), and the rightmost digit is the least significant digit (LSD). We remind you that the digits of the given binary number in order of increasing precedence have weight (from right to left) 1, 2, 4, 8, 16, 32.

Since there are only two digits (0 and 1) in binary numbers, the addition table is quite simple. It is shown in fig. 7.5.1. As in the case of adding decimal numbers, the first three results do not raise questions. As for the last problem (1+1), when adding decimal numbers in this case, the answer would be number 2. Thus, in binary addition, 1+1=0 plus the transfer 1 to the adjacent most significant binary digit.

0	1	0	1	
+0	+0	+1	+1	
0	1	1	1	Перенос 1

Fig. 4.37. Binary addition table

Another example of adding binary numbers is shown in fig. 7.5.2, a.



1.1 *Fig. 4.38. Binary addition a) – an example of binary addition; b) - the abbreviated form of the binary addition table*

The solution looks simple until we get to the digit of two, where we need to find the binary sum. This sum is equal to 3 in the decimal system, which corresponds to the number 11. This case is not shown in fig. 4.37. The sum 1+1+1 can occur in any digit except the ones digit. One more possible combination 1+1+1 is included in the new (abbreviated) table in fig 4.38.

This table is valid for all bits of binary numbers, with the exception of the ones bit.

The corresponding truth table (table 7.5.1) provides all possible combinations of binary one-digit terms A, B and the carry signal C_{in}.

Table 4.12. Full adder truth table

Input			***Output***	
With $_{in}$	IN	AND	S	From $_{0}$

0	0	0	0	0
0	0	1	1	0
0	1	0	1	0
0	1	1	0	1
1	0	0	1	0
1	0	1	0	1
1	1	0	0	1
1	1	1	1	1
Carry+ B + A			Sum	Carry

Full adders are used to add all binary digits except ones. They must have an additional carry input.

A full adder is a 3-input circuit. The signals at its outputs S and C_0 are obtained as a result of the addition of three input signals (at the inputs A, B and C_{in}).

The expanded logical circuit of a full adder is given in fig. 4.39. It is based on a structural scheme with two half-adders.

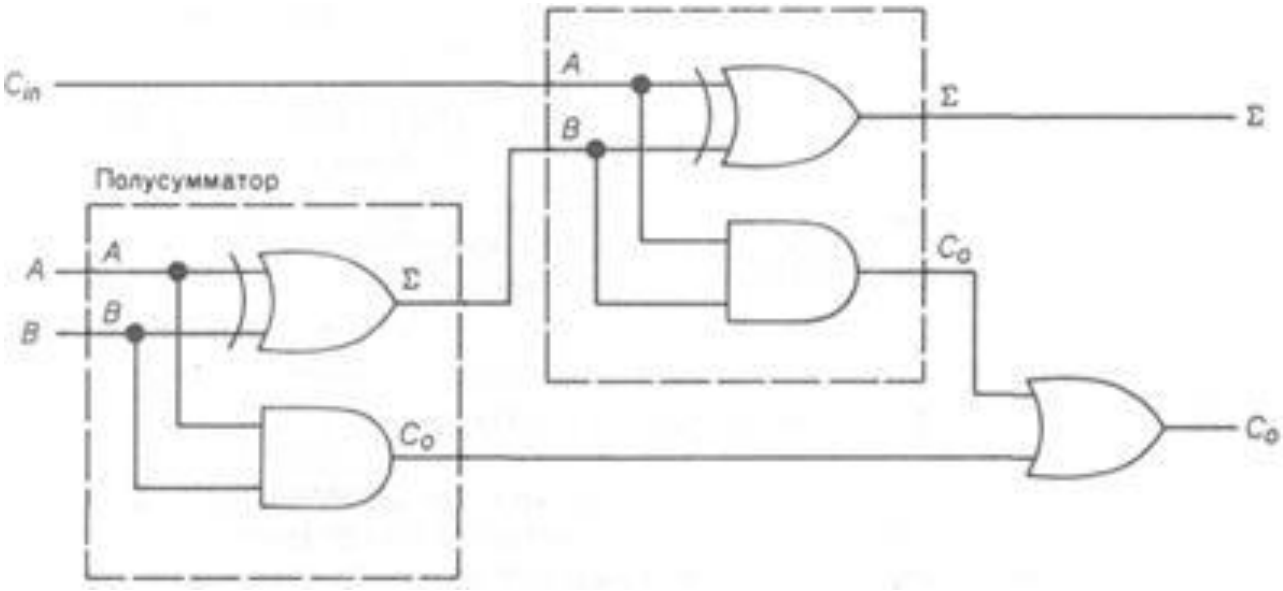


Fig. 4.39. The logic circuit of a full adder

19.4.1.1.1. Another circuit of a full adder using two logical exclusive elements or three logical elements AND -NOT is given in fig. 4.40. Note that the circuits are shown in fig. 4.39 and 4.40. differ only in the replacement of logical elements AND and OR with logical elements AND-NOT.

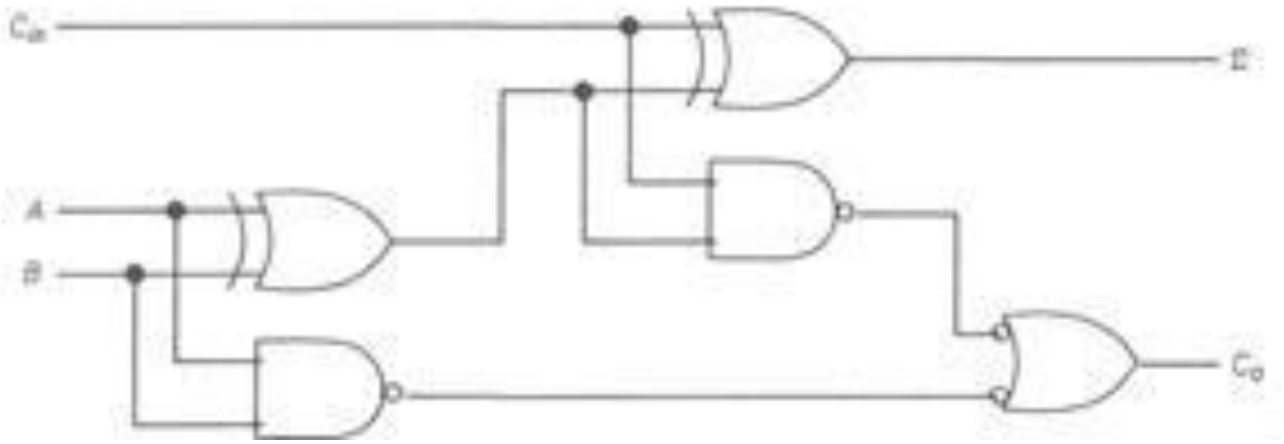


Fig. 4.40. The logic circuit of a full adder using exclusive OR and AND-NOT

Half-adders and adders are usually used together. A large number of circuits similar to half-adders and full-adders are part of microprocessor arithmetic- logical devices (ALDs). Microprocessor ALDs can also perform subtraction using the same half-adders and adders.

By combining half-adders and full-adders in a certain way, the devices are obtained that simultaneously perform the addition of several binary digits. The device, the circuit of which is shown in fig. 7.5.5. performs the operation of adding two 3-digit numbers. The numbers are summands $_{A2A1A0}$ and $_{B2B1B0.}$

Signals corresponding to the values of the unities digit in terms enter the adder input to the unity digit (half adder). The input signals for the binary full adder are the carry signal from the output of the half adder (supplied to the C_{in} input) and the values of A $_1$ and 1 of the binary digits and terms. Next, the adder of fours adds A $_2$ and B $_2$ and the carry signal from the adder of twos. A binary sum is set at the binary output of the device (shown in the lower right corner of fig. 4.41).

As a result of adding two 3-bit binary numbers, you can get a 4-digit number, so we have an extra bit of eights on the sum indicator. Note that this bit is connected to the output (C_0) of the adder of four.

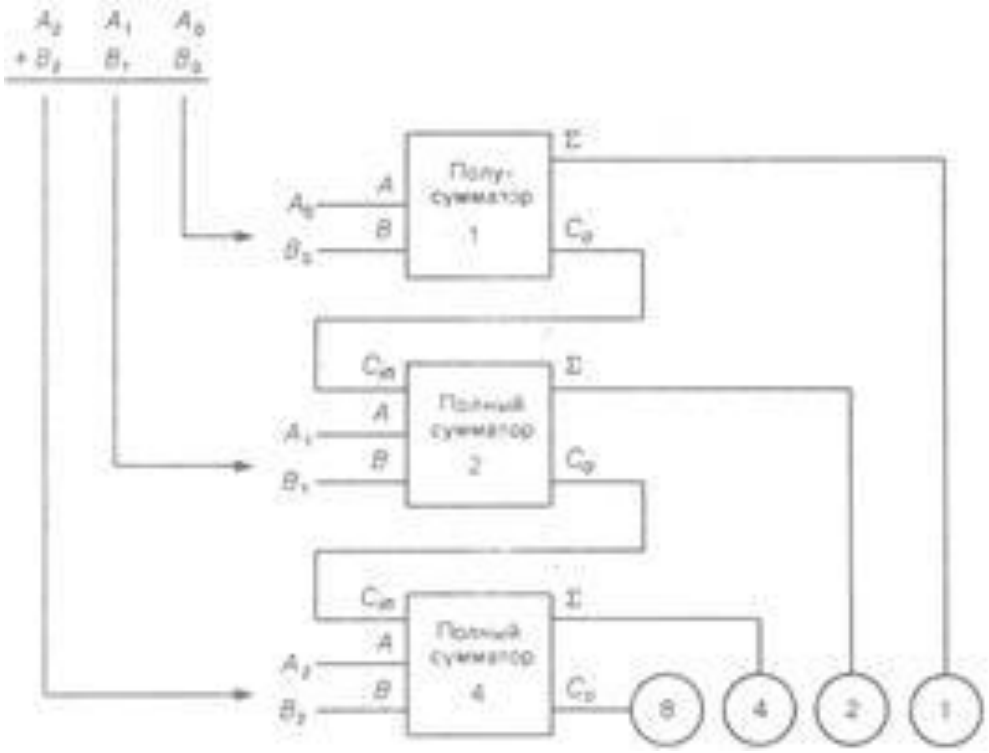


Fig. 4.41. 3-bit parallel adder

The logic of operation of the 3-bit adder is almost no different from the sequence of operations performed during manual addition (adding single-digit numbers plus carry to the next digit). However, the electronic adder performs these operations much faster.

Let us note once again that in multi-bit adders, half-adders are used only to add in the unity digit; all other digits use full adders. The above mentioned a 3-bit adder is called a parallel adder.

The information bits of all digits enter the inputs simultaneously in a parallel adder. The result (sum) appears at the output almost instantly. The parallel adder in fig. 4.41. belongs to the class of combinational logic circuits. Various additional registers are usually used to fix data at the inputs and outputs of adders.

It will be shown later that adders and subtractors are similar to each other, and furthermore half-subtractors and full-subtractors are used similarly to half-adders

and full-adders. The binary subtraction table is given below and it shows the rules for subtracting the binary numbers presented in fig. 4.41. in the form of a truth table. We see that B is subtracted from A (A and B are input signals), the result (difference) appears at the output D_1. If it is more than A (as in a line 2 of the table), you need to borrow 1 to the adjacent most significant digit. The borrow signal is indicated in column $_0$.

Table 4.13. Table of binary subtraction

Input		Output	
AND	IN	D_i	B_0
0	0	0	0
0	1	1	1
1	0	1	0
1	1	0	0
AB		Difference	Borrowing

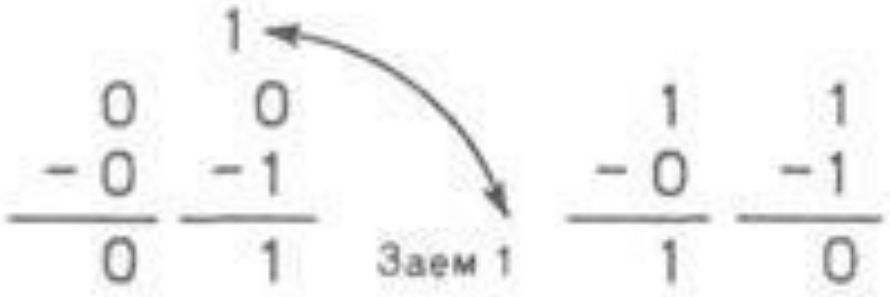


Fig. 4.42. Table of binary subtraction

When subtracting multi-digit binary numbers, the credit of "ones" in the most significant digits should be taken into account.

A truth table containing all possible combinations resulting from the subtraction of binary numbers is given below.

Table 4.14. Full subtractor validity table

Input			***Output***	
A	IN	B_{in}	D_i	B_0
0	0	0	0	0
0	0	1	1	1
0	1	0	1	1
0	1	1	0	1
1	0	0	1	0
1	0	1	0	0
1	1	0	0	0
1	1	1	1	1
$A - B - B_{in}$			Difference	Borrowing

Conventional designation of a full subtractor is shown in fig. 4.43, a. On the left are inputs A, B, B_{in}, on the right - outputs $D_{I,}$, B_0. The figure 4.43b shows how to combine half-subtractors and an OR logic element to obtain a full subtractor. The detailed logic diagram of a full subtractor is shown in fig. 4.43. This circuit operates in accordance with the truth table. If necessary, logical elements AND and OR can be replaced by three logical elements AND-NOT. In this case, we will get a full subtractor circuit similar to a full adder circuit.

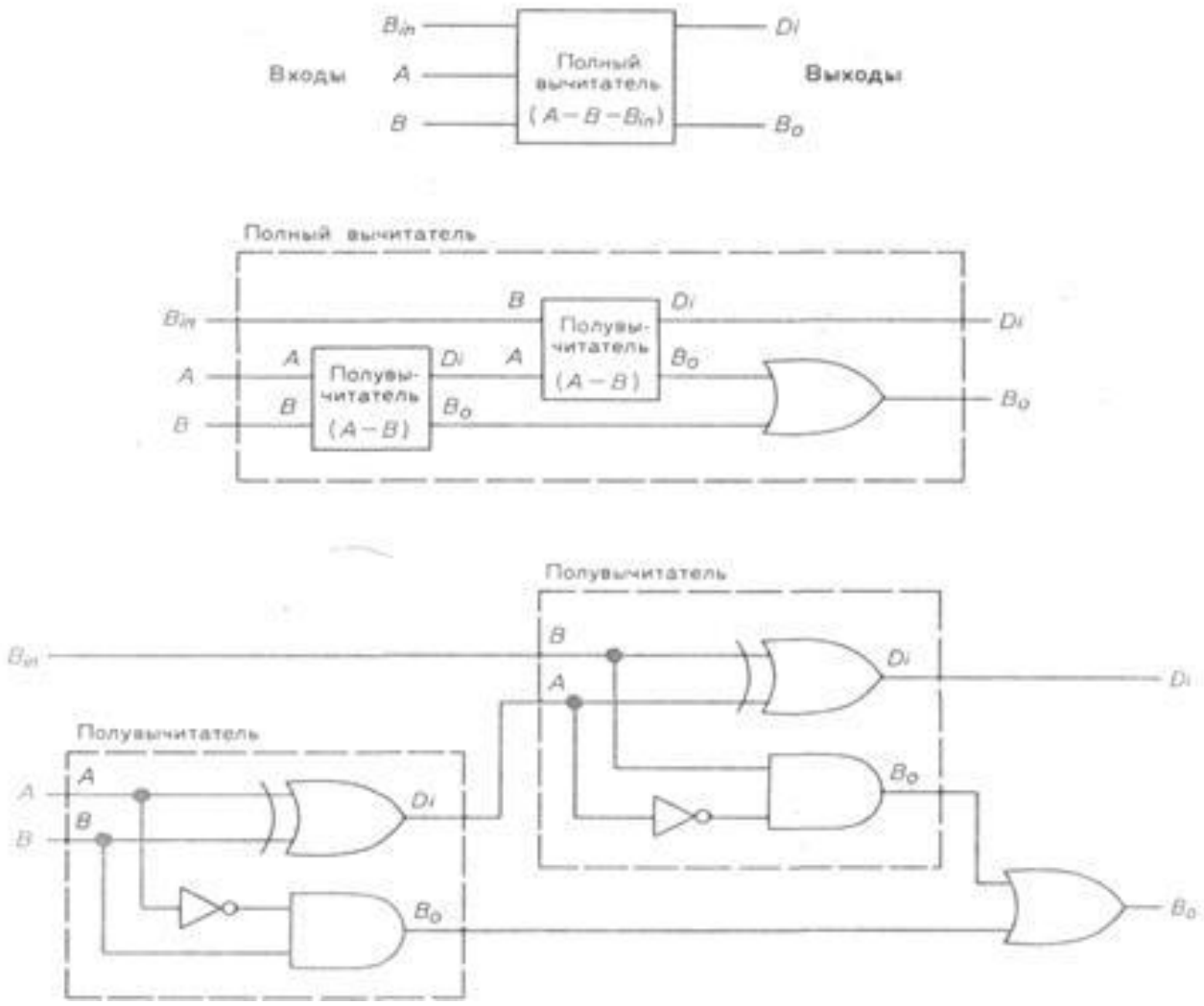


Fig. 4.43. Full subtractor a) – conditional graphic notation; b) – structural diagram for the case of using two half-subtractors and an OR logic element; c) circuit diagram

The devices called parallel subtractors are obtained by combining half-subtractors and full -subtractors. A parallel subtractor is assembled similarly to the three-bit adder considered above. The adder in fig. 4.42 is called a parallel one, because the information bits of all digits in the addends arrive at this adder simultaneously.

Figure. 4.44 shows the structural diagram obtained by combining one half-subtractor and three full subtractors. This is a circuit of a 4-bit parallel subtractor that performs the operation of subtracting one binary number $B_3 B_2 B_1 B_0$ from the binary number $A_3 A_2 A_1 A_0$. Please note that the upper (in the circuit) subtractor

performs subtraction in the unities digit (CMP). The output B_0 of this subtractor is connected to the subtractor of the digit of twos.

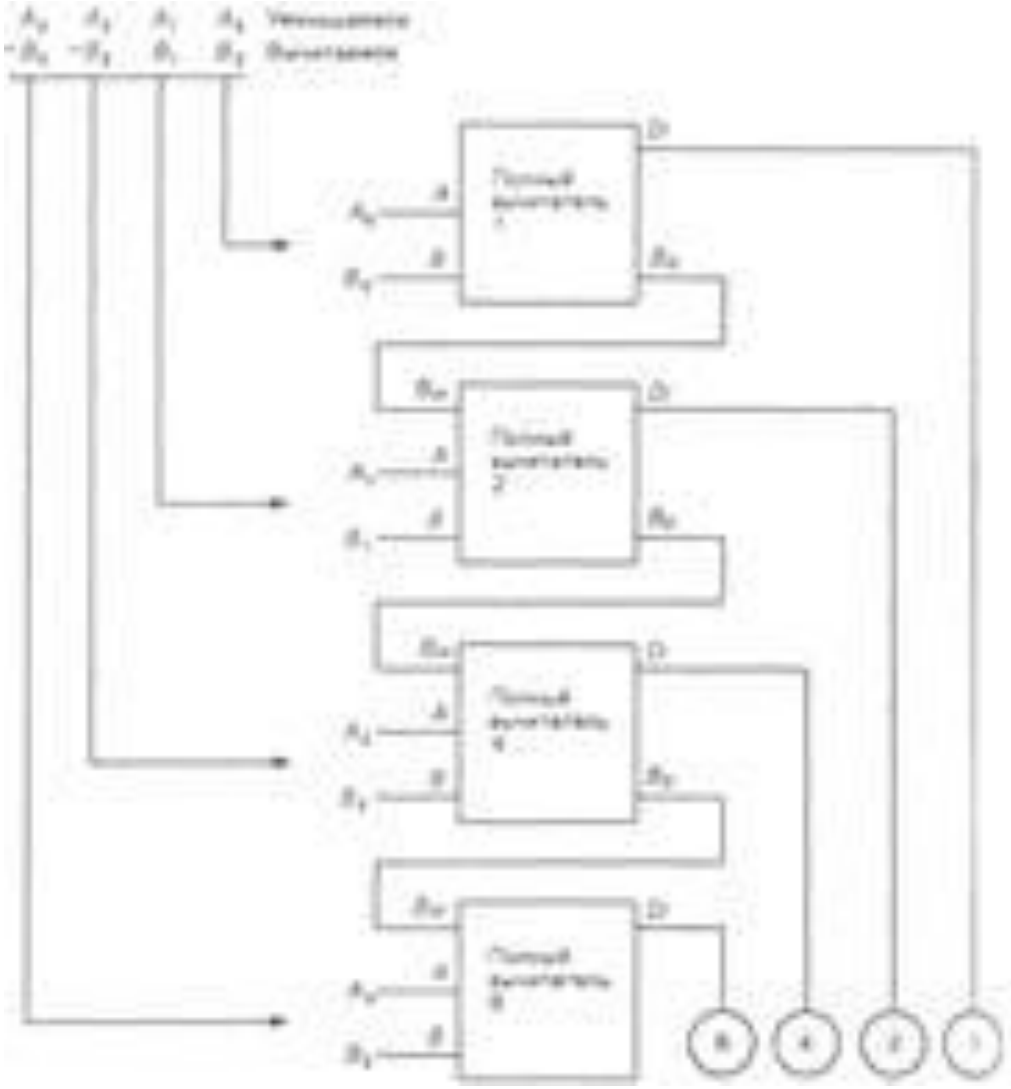


Fig. 4.44. 4-bit parallel subtractor

In general, the output of the borrowing B_0 of each subtractor is connected to the input of the borrowing $B_{in\ of}$ the subtractor of the adjacent most significant digit. These communication lines " monitor" the borrowings in the process of subtracting binary numbers.

The program for implementing a 4-bit adder using the AHDL language in the MAX+PLUS II integrated environment looks like this:

```
SUBDESIGN add_gate
        (
                A [4..1], B[4..1], cin: input;
                C [4..1], cout: output;
        )
VARIABLE
        carry_out [5..1]: node;
BEGIN
        carry_out [1] = cin;
                FOR i IN 1 TO 4 GENERATE
        C[i] = A[i] $B[i] $ carry_out[i];
carry_out [i + 1] = CARRY (A [i] & B[i] # carry_out[i] & (A[i] # B[i]));
        END GENERATE;
                cout = carry_out [5];
END;
```

The program for implementing a 4-bit subtractor using the AHDL language in the MAX+PLUS II integrated environment looks like this:

```
SUBDESIGN add_sub
        (
                A [4..1], B [4..1]: input = GND;
                Res [4..1], Cout: output;
        )
VARIABLE
        S [4..1]: node;
        Cout_int: node;
BEGIN
                (Cout_int, S []) = (GND, A []) - (GND, B[]);
                (Cout, Res []) = (Cout_int, S[]);
```

END;

The signal editor window of the add_gate project is shown in fig. 4.45.

Fig. 4.45. Test results of a 4-bit adder

The signal editor window of the add_sub project is shown in fig. 4.46.

Fig. 4.46. 4-bit subtractor test results

4.9. COM port implementation project in CAD MAX+PLUS II

A serial interface for one-way data transmission uses a single signal line, along which information bits are transmitted one after the other sequentially. This means of transmission determines the name of the interface and the port that implements it. These names correspond to the English terms Serial Interface and Serial Port. Serial data transfer can occur in both asynchronous and synchronous modes.

In asynchronous transmission each bit is preceded by a start bit, which signals the receiver about the start of the nex trasmission, usually followed by data bits and possibly a parity bit. The transmission ends with a stop bit, which guarantees a defined delay between adjacent trasmissions (fig. 4.47).

Fig. 4.47. Asynchronous transmission format

The start bit of the next sent byte can be sent at any time after the end of the stop bit, that is, pauses of unfixed duration are possible between transmissions. The start bit, which always has a strictly defined value (log. 0), provides a simple mechanism for synchronizing the receiver. The receiver and transmitter operate at the same communication speed, measured in the number of transmitted bits per second. The internal clock generator of the receiver uses a counter-divider of the reference frequency, which is reset to zero at the moment of reception (leading edge) of the start bit. This counter generates internal gates by which the receiver fixes the bits it receives. Ideally, the gates are located in the middle of the bit intervals, which makes it possibile to receive data in case of some inconsistencies of the receiver and transmitter speed.

It is easy to see that when transmitting 8 bits of data, one control bit and one stop bit, the maximum allowable inconsistency between the receiver and the transmitter speeds, at which the data will be recognized correctly, cannot exceed 5%. Taking into account the phase changes (delayed signal edges) and the discrete operation of the internal synchronization counter, less frequency deviations are actually permissible. The smaller the division factor of the internal frequency of the internal generator (transmission frequency), the greater the error of binding the strobes to the middle of the bit interval, and, accordingly, the requirements for frequency matching are the most stringent. Also, the higher the transmission frequency, the greater the impact of factors that lead to errors.

The format of the asynchronous sending allows you to detect possible transmission errors:

1) if a drop is received, signaling the start of the package, and the level of a logical unity is fixed behind the start bit strobe, then the start bit is considered false and the receiver will go to the standby state again. The receiver may not report this format error;

2) if a logical zero level is detected under the hour assigned to the stop bit(s), a stop bit error (also a format error) is recorded;

3) if parity control is used, then after transfering the data bits (before the stop bit) a control bit is sent. This bit complements the number of single data bits to be even or odd depending on the accepted agreement. Receiving a bit with a false value of the control bit when the parity control is enabled leads to fixing the error of the received data.

Format control allows you to find a line break: in this case, a logical zero is usually accepted, which is initially interpreted as a start bit and zero data bits, but then the stop bit control is triggered.

The number of data bits can be 5, 6, 7, or 8 (5- and 6-bit formats are less common). The number of stop bits can be 1, 1.5 and 2 ("one and a half bit" means only the length of the stop interval).

Asynchronous exchange in a personal computer is implemented using the RC-232 protocol.

The RC-232 interface is designed to connect equipment that receives or transmits data (ODTE-one data transfer endpoint or DTE- data transfer equipment) to data link terminal equipment. A computer, printer, plotter or other peripheral devices can act as an DTE. This equipment corresponds to the abbreviation DTE - Data Transfer Equipment. A modem usually plays the role of ACD, this equipment corresponds to the abbreviation DCE - Data Communication Equipment. The ultimate purpose of the connection is to connect two DTE devices, the complete connection diagram is shown in fig. 4.48. The interface allows you to eliminate the communication channel together with a pair of DTE devices (modems) by connecting the devices directly using a null-modem cable (fig. 4.49).

Fig. 4.48. Complete connection diagram according to RC-232

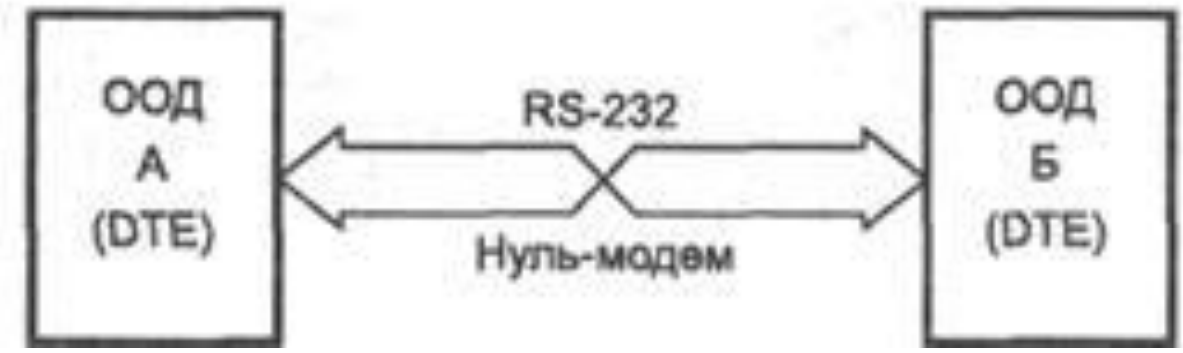


Fig. 4.49. Connection by RC-232 null-modem cable

The standard describes interface control signals, data transfer, electrical interface and connection types. The standard describes asynchronous and synchronous exchange modes, but COM ports support only asynchronous mode.

The basic structural diagram of the COM port when implemented in the MAX+PLUS II automated design system is shown in fig. 4.48. It is an expanded version of figure 4.49, where the remote control with 20 output signal options will act as one end object of data transmission.

Let's take the stop bit as 1 and the number of data bits as 8. The frequency of the sync signal can be 115200 bits per second or 9600 bits per second. To provide the circuit with the ability to switch the frequency, we will introduce a switch signal to it, which will switch the frequency of the synchronization signals that will come to the clocking input.

All circuit elements can be described separately using the AHDL language in a text editor, and then, using a symbol editor, convert the elements into symbols and assemble the entire circuit in a graphic editor.

Let's consider the functional purpose of each of the devices listed in the diagram.

A 20-by-10 encoder has a 20-bit input (Encipherer_1_input [20..1]) and a 10-bit output (Encipherer_1_output[10..1]). A description encoder using a truth table performs two functions.

First, depending on the key pressed on the remote control, it outputs a binary code that is in the range from 0000000011 to 0000101001. This binary sequence of output code always starts with 0 and ends with 1, these numbers are nothing but start and stop -bit. That is, the encoder outputs a ready-made code that can be transmitted over a communication line.

Secondly, it is possible that two keys will be pressed at the same time, which can lead to a system failure. That's why the line

WHEN OTHERS => Encipherer_1_output [] = b"1111111111";

instructs the cipher to output the binary code 1111111111 in any other cases not provided for by the cipher's truth table - this is a protective function.

A 5-bit counter on D-triggers has an input (Counter_1_input), which receives a synchronization signal from an external pulse generator, a clearing input (Reset) and performs the function of a frequency divider. This counter has two outputs

(Counter_1_output [1] and Counter_1_output [1]) corresponding to the input frequency divided by 2 and the input frequency divided by 12.

The multiplexer has 4 information inputs (Multiplexer_1_input [4..1]), 2 address inputs (Switch_signal[2..1]) and an enable input (Enable). The device is implemented using an emulated truth table and its task is to switch the output (Multiplexer_1__output) to one of the information inputs depending on the state of the address inputs. Since the operating frequency of the system can be equal to 115200 bits per second or 9600 bits per second, one address input is sufficient for switching control, and the other address input (Switch_signal [2]) corresponding to the name SWITCH [2] in the graphic editor window is grounded.

For the same reason, the Multiplexer_1_input [4] and Multiplexer_1_inpu t [3] signals corresponding to the names MUX [4] and MUX [3] in the graphic editor window are grounded. Another address input may be needed if in the future, when modifying the scheme, the number of possible frequency variations will increase to four.

A 4-bit counter on D triggers has an input (CLK) to which the synchronization signal selected by the multiplexer and a reset input (Reset) are sent. This counter controls the loading of the serial register by setting its own output (LOAD) to 0 or 1: 0 - loading is prohibited, 1 - loading is allowed.

A 10-bit parallel D-triggers register stores and shifts with the arrival of a new synchronization pulse the source code generated and transferred to it by the multiplexer. The register has a 10-bit input for loading "information" code (Register_1_input [9..0]), an enable input (Enable), a preset input (Set), a synchronization input (Clk), a load control input (Load) and a single-bit output (Register_1_output).

The program for implementing the 20 by 10 encoder (description of the encoder validity table) using the AHDL language in the MAX+PLUS II integrated environment looks like this:

Subdesign encipherer_1

```
(
        Encipherer_1_input [20..1]: input;
        Encipherer_1_output [10..1]: output;
)
Begin
        CASE Encipherer_1_input [] IS
WHEN b"00000000000000000001" => Encipherer_1_output [] = b"0000000011";
WHEN b"00000000000000000010" => Encipherer_1_output [] = b"0000000101";
WHEN b"00000000000000000100" => Encipherer_1_output [] = b"0000000111";
WHEN b"00000000000000001000" => Encipherer_1_output [] = b"0000001001";
WHEN b"00000000000000010000" => Encipherer_1_output [] = b"0000001011";
WHEN b"00000000000000100000" => Encipherer_1_output [] = b"0000001101";
WHEN b"00000000000001000000" => Encipherer_1_output [] = b"0000001111";
WHEN b"00000000000010000000" => Encipherer_1_output [] = b"0000010001";
WHEN b"00000000000100000000" => Encipherer_1_output [] = b"0000010011";
WHEN b"00000000001000000000" => Encipherer_1_output [] = b"0000010101";
WHEN b"00000000010000000000" => Encipherer_1_output [] = b"0000010111";
WHEN b"00000000100000000000" => Encipherer_1_output [] = b"0000011001";
WHEN b"00000001000000000000" => Encipherer_1_output [] = b"0000011011";
WHEN b"00000010000000000000" => Encipherer_1_output [] = b"0000011101";
WHEN b"00000100000000000000" => Encipherer_1_output [] = b"0000011111";
WHEN b"00001000000000000000" => Encipherer_1_output [] = b"0000100001";
WHEN b"00010000000000000000" => Encipherer_1_output [] = b"0000100011";
WHEN b"00100000000000000000" => Encipherer_1_output [] = b"0000100101";
WHEN b"01000000000000000000" => Encipherer_1_output [] = b"0000100111";
WHEN b"10000000000000000000" => Encipherer_1_output [] = b"0000101001";
WHEN OTHERS => Encipherer_1_outpu [] = b"1111111111";
        End CASE;
        End;
```

Program for implementing a multiplexer with 4 informative, 2 address and the enable inputs (described by emulated multiplexer table truth) using AHDL language in integrated MAX+PLUS II environment looks like this:

```
Subdesign multiplexer_1
    (
        Multiplexer_1_input [4..1]: input;
        Switch_signal [2..1]: input;
        Enable: input;
        Multiplexer_1__output: output;
    )
    Begin
        if Enabled == 0 then
            case Switch_signal [2..1] is
    when 0 => Multiplexer_1__output = Multiplexer_1_input [1];
    when 1 => Multiplexer_1__output = Multiplexer_1_input [2];
    when 2 => Multiplexer_1__output = Multiplexer_1_input [3];
    when 3 => Multiplexer_1__output = Multiplexer_1_input [4];
        end case;
    end if;
End;
```

The program for implementing a 10-bit serial shift register using the AHDL language in the MAX+PLUS II integrated environment looks like this:

```
    Subdesign register_1
        (
        Register_1_input [9..0]: input;
        Enable, Set, Clk, Load: input;
        Register_1_output: output;
```

```
        )
    Variable
        Triggers [9..0]: DFFE;
    Begin
        Triggers [9..0].clk = Clk;
        Triggers [9..0].prn = Set;
        Triggers [9..0].ena = Enable;
    IF Load == 0
    THEN
        Triggers [] .d = (Triggers[8..0].q, VCC);
    ELSE
        Triggers [] .d = Register_1_input[];
    END IF;
    Register_1_output = Triggers [9] .q;
End;
```

The program for implementing a 4-bit counter using the AHDL language in the MAX+PLUS II integrated environment looks like this:

```
Subdesign counter_2
    (
        CLK: input;
        Reset: input;
        LOAD: output;
    )
    Variable
        TRIG [3..0]: DFF;
    Begin
        TRIG []. clrn = Reset;
        TRIG []. clk = CLK;
```

```
        IF (TRIG [] .q == B"1011")
        THEN TRIG [] .d = B"0000";
        LOAD = B"1";
        ELSE TRIG [] .d = TRIG[].q + 1;
        LOAD = B"0";
    END IF;
End;
```

The program for implementing a 5-bit counter by modulo 12 using the AHDL language in the MAX+PLUS II integrated environment looks like this:

```
Subdesign counter_1
    (
        Counter_1_input: input;
        Reset: input;
        Counter_1_output [2..1]: output;
    )
    Variable
        Triggers: JKFFE;
        TRIG [4..0]: DFF;
    Begin
        Triggers.j = vcc;
        Triggers.k = vcc;
        Triggers.clrn = Reset;
        Triggers.clk = Counter_1_input;
        TRIG []. clrn = Reset;
        TRIG []. clk = Counter_1_input;
        IF (RIG []  .q == B"11000")
        THEN TRIG [] .d = B"00000";
        ELSE TRIG [].d = TRIG[].q + 1;
```

```
    END IF;
    Counter_1_output [2..1] = (TRIG[4].q, Triggers.q);
End;
```

The diagram of the project of the COM port assembled from individual subroutine symbols in the graphic editor window is shown in fig. 4.50. Subroutine symbols depicted in the form of blocks with named inputs and outputs are connected by communication lines.

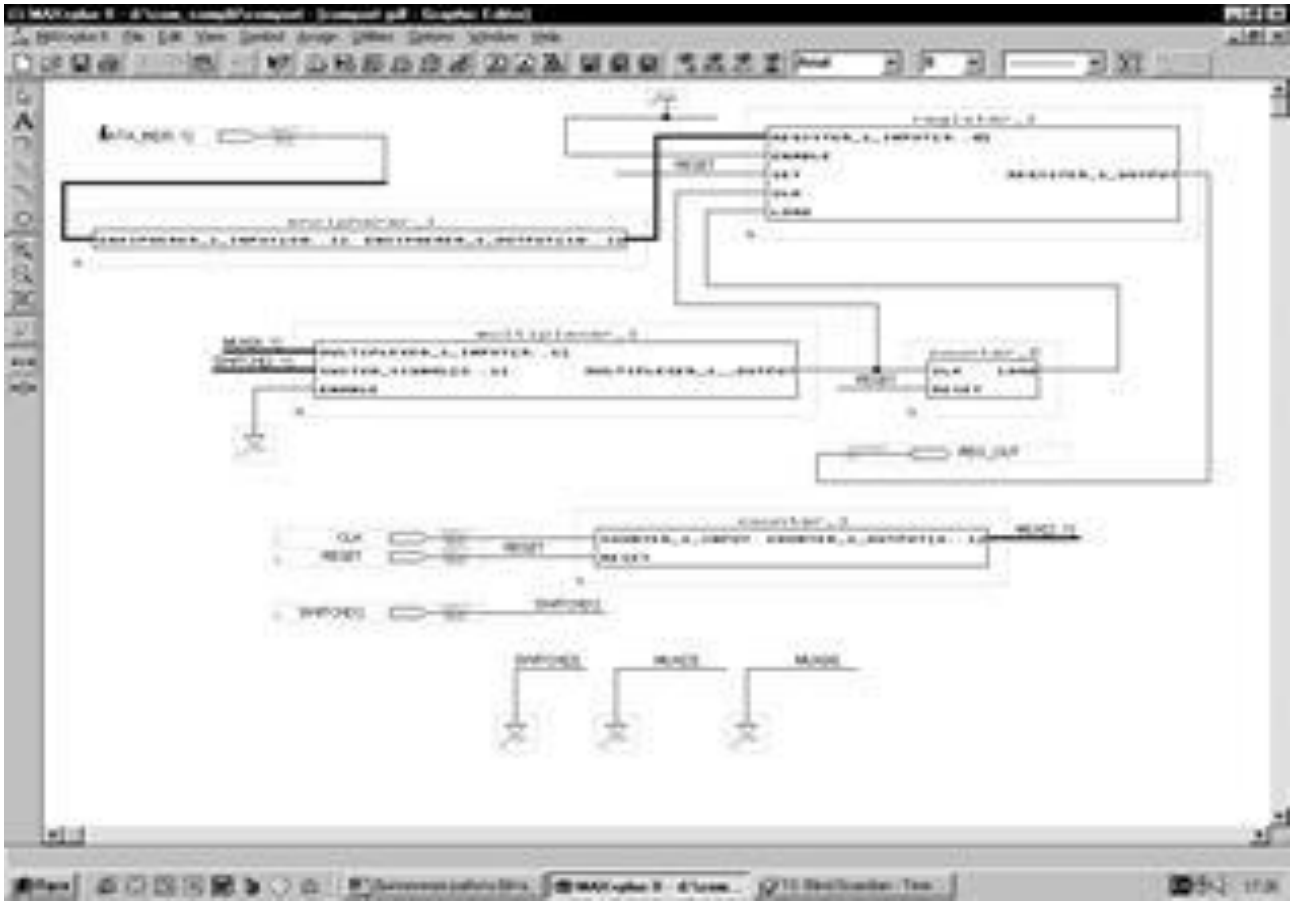


Fig. 4.50. Implementation of the COM port project in the graphic editor

The results of testing the software implementation of the COM port in CAD MAX+PLUS II are shown in fig. 4.51. The step-by-step process of compiling the entire project as well as its components is described in section 4 of this book.

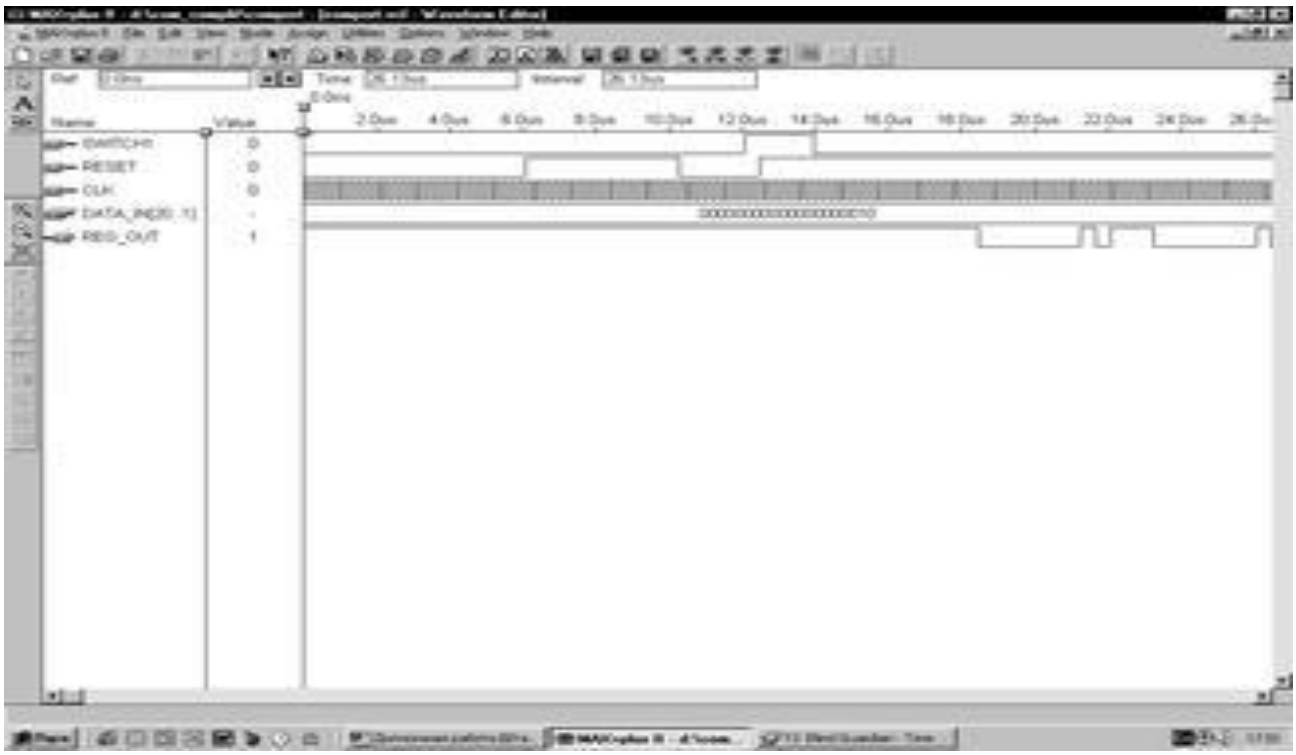


Fig. 4.51 Results of testing of the COM port project

Thus, the information component of the MAX+PLUS II integrated environment base is a description of the structure of the considered environment, the purpose of its main applications, as well as their relationship, a detailed procedure for developing a new project in the MAX+PLUS II integrated environment. The process of compiling the created project will allow to reduce the hours for the training of FPGA programming specialists and their acquisition of practical skills.

In order to make the programming process more understandable, a classification was made and the architecture of the most common programming logic integrated circuits was revealed.

The software component base for working with the MAX+PLUS II integrated environment is a theoretical description of basic microelectronics devices together with programs for their implementation:

1) JK trigger;

2) D-trigger;

3) RS trigger;

4) synchronous RS trigger;

5) 4-bit serial shift register;

6) 4-bit parallel ring shift register;

7) 4-bit asynchronous counter with end-to-end transfer modulo 16;

8) asynchronous counter module 10;

9) asynchronous three-digit subtraction counter;

10) 3-digit universal counter;

11) 10 by 4 encoder;

12) 3-bit decoder with inverse inputs;

13) multiplexer with two address inputs, four information inputs and an enable input;

14) demultiplexer with three address inputs, one information and an enable input;

15) 4- bit adder;

16) 4- bit subtractor.

Since recently the architectures of programming logic integrated circuits have been rapidly developing and improving but the design methods based on them remain unchanged, the data presented in this work can be used both for didactic and research purposes.

CHAPTER V. OrCAD COMPUTER DESIGN

SYSTEM

5.1. General information about the OrCAD system

OrCAD is a computer program package designed to automate electronics design. It is mainly used to create electronic versions of printed circuit boards for the production of printed circuit boards, as well as for the production of electronic circuits and their simulation.

OrCAD Capture - graphic circuit editor;

OrCAD Capture CIS (Component Information System) is a graphic circuit editor supplemented with a tool for using component databases; in this case, registered users receive access via the Internet (with the help of the ICA, Internet Component Assistant) to the catalog of components, which contains more than 200,000 items;

PSpice Schematics - a graphic scheme editor borrowed from the DesignLab package;

OrCAD PSpice A / D - a program for modeling analog and mixed analog-digital devices, data is transferred in it both from PSpice Schematics and from OrCAD Capture;

OrCAD PSpice Optimizer - a parametric optimization program;

OrCAD Layout - graphic editor of printed circuit boards;

OrCAD Layout Plus - the OrCAD Layout program, supplemented by the wireless SmartRoute Autorouter, which uses neural network optimization methods (also used in Protel 99 SE and P-CAD 2000 systems);

OrCAD Layout Engineer's Edition - a program for viewing printed circuit boards created using Layout or Layout Plus, a tool for the general arrangement of components on the board and the laying out the most critical circuits performed by a circuit engineer before issuing a printed circuit board design task to a designer;

OrCAD GerbTool - a program for creating and modifying control files for photoplotters (developed by the WISE Software Solutions company specifically for OrCAD, an analogue of the SAM350 program);

General characteristics of the OrCAD Capture program

The OrCAD Capture program is designed to create a project, part of which can be specified in the form of a circuit diagram, and the other part can be described in the high-level VHDL language. In addition, simulation programs for PSpiee analog, digital and mixed analog-digital devices and parametric optimization PSpiee Optimizer are launched using OrCAD Capture shell. The projects in OrCAD Capture program are divided into several types.

When creating a project according to its type, the necessary libraries of components are automatically loaded (later their list can be changed manually), at the same time for all specialized projects it is possible to transfer information to the OrCAD Layout program for creating printed circuit boards. The relationship between OrCAD Capture and other programs of the OrCAD system is shown in fig. 5.1. When creating schematic diagrams of a project, the necessary information is found in the built-in database, supplied with the system and is replenished by users. Moreover, if the Component Information Systems (CIS) option is available, official users get access via the Internet to an extended database containing information about approximately 200,000 components of various companies (their symbols and cases are given).

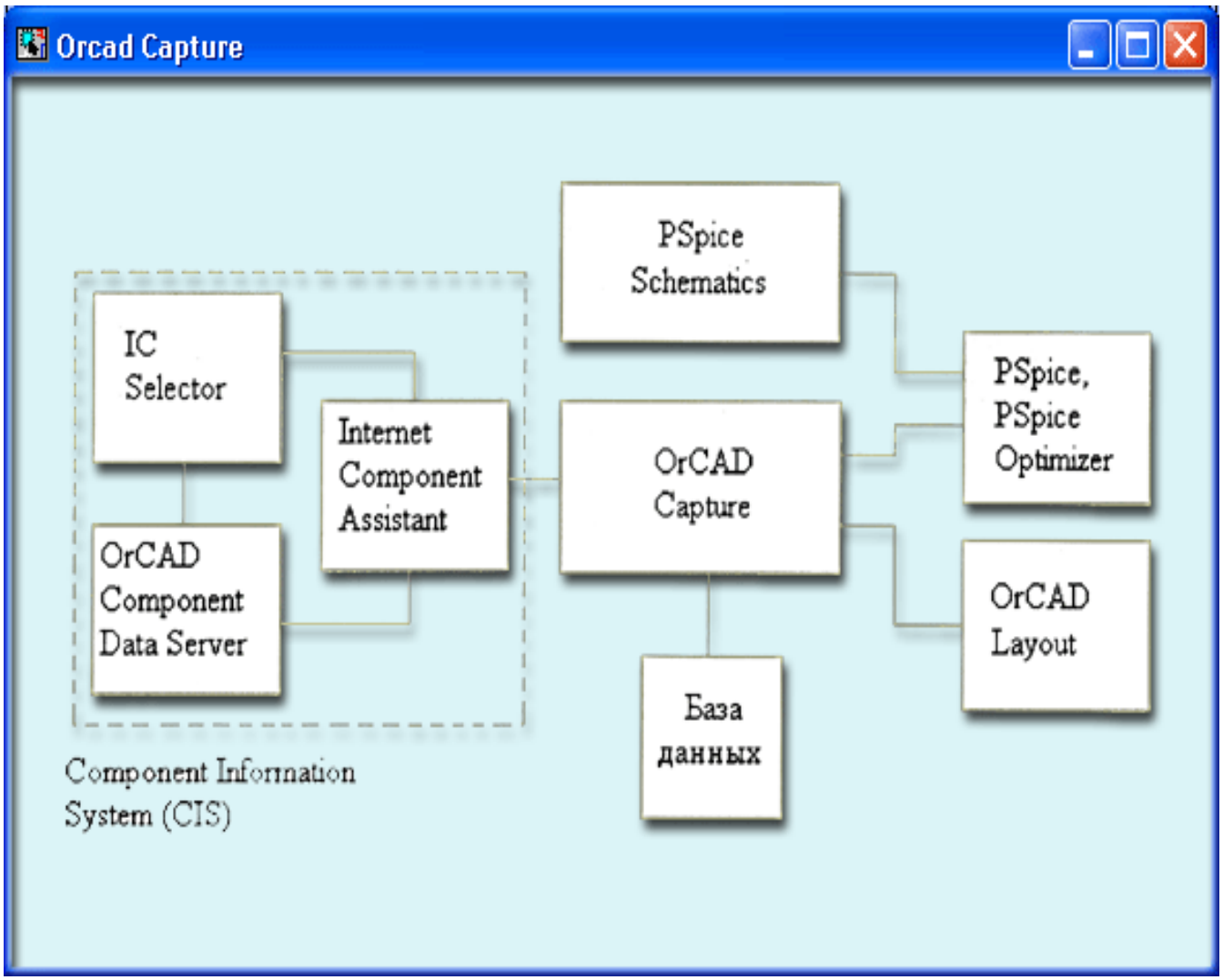


Fig. 5.1. Interaction of OrCAD Capture with other programs

The screen of the OrCAD Capture 9.2 program is shown in fig. 5.2. In its upper part there is a menu of commands and below - a toolbar.

Command menu and toolbar. The composition of the toolbar icons depends on the selected operating mode and the type of the current project, their composition is shown in fig. 5.3 and given in table. 5.1.

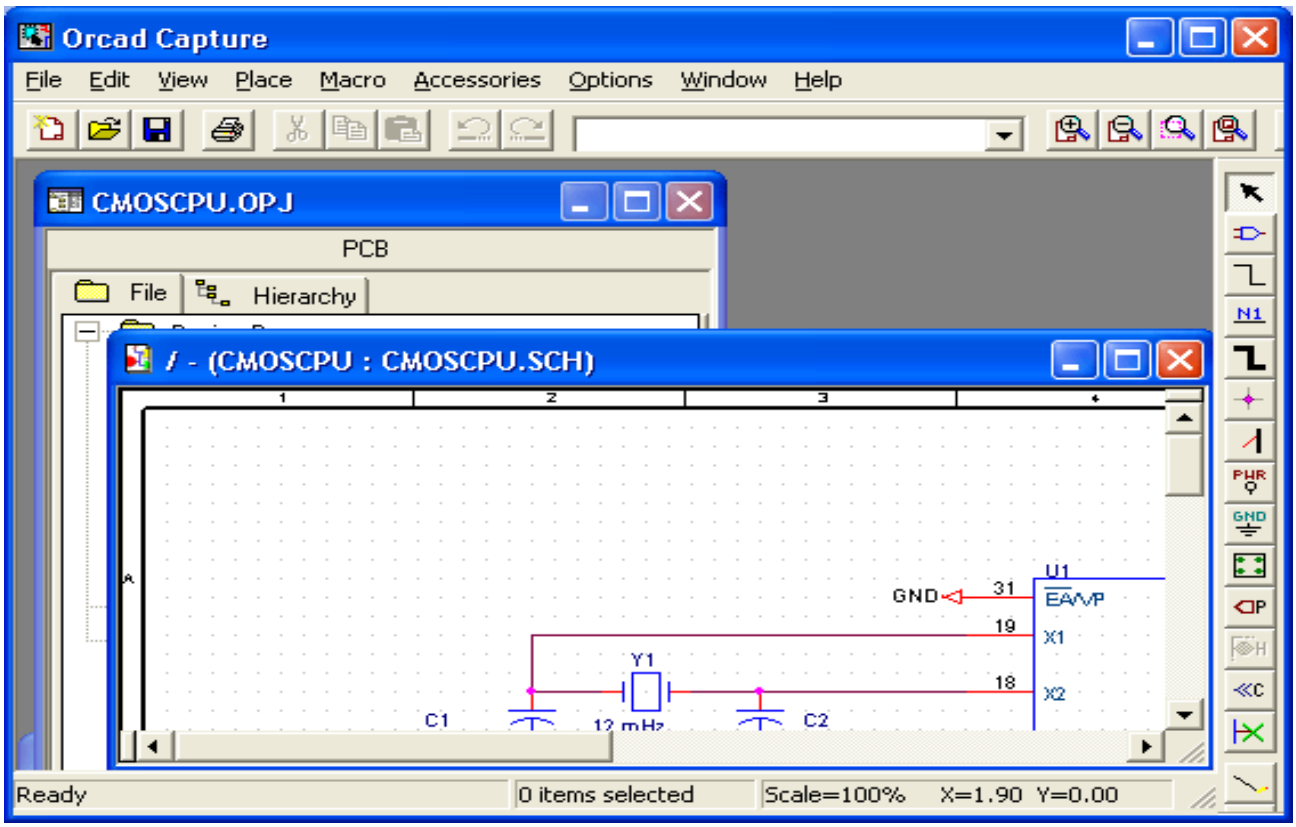


Fig. 5.2. OrCAD Capture program screen

Table 5.1. Toolbar icons

Icon	Equivalent command	Command description
	New	Creating a new document
	Open	Opening an existing document
	Save	Saving changes made in the current project
	Print	Print the current diagram or component symbol image
	Cut	Delete the selected object by copying it to the clipboard
	Copy	Copy the selected object to the clipboard
	Paste	Paste an object from the clipboard
	Undo	Undoing the result of one last command
	Redo	Undoing the result of one last Undo command

	Zoom in	Zoom in
	Zoom area	Displaying the selected part of the image on the entire screen
	Zoom all	Displaying the full image of the schematic page on the screen
	Annotate	Assignment of positional markings of the components of the selected page of the scheme
	Back Annotate	Performing permutations of logical equivalent sections of components and pins in the process of inverse adjustment
	Design Rules Check	Checking compliance with DRC design rules and ERC electrical circuit rules
	Create Netlist	Compiling a netlist file of of the selected circuit page in EDIF 200, SPICE, VHDL, Verilog, Layout, etc. formats.
	Cross Reference	Compiling a cross-reference file
	Bill of Materials	Compiling a report about a project or a selected page
	Snap to Grid	Snapping the cursor to grid nodes on the window for editing diagrams and component symbols (similar to the Options> Preferences> Grid Display command)
	Project manager	Downloading the project manager
	Help Topics Help Topics	Displaying the topic, subject index and search tools for the terms of the built-in instructions

The composition of the command menu depends on the selected operating mode and the type of the current project. The content of the command menu when the project manager is activating is shown in fig. 5.4.

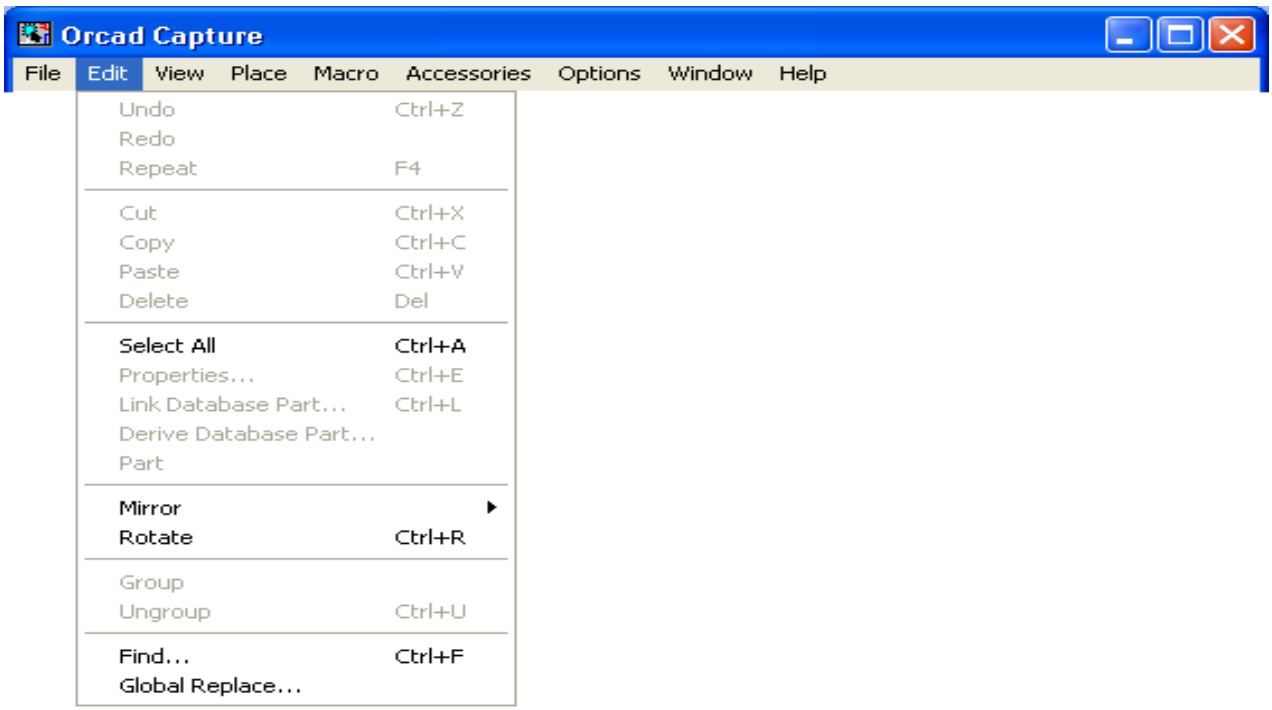


Fig. 5.4. The composition of the project manager command menu

The project manager is located on the left side of the Capture program screen., The flat file structure of the project is expanded in File mode and its hierarchical structure in Hierarchy mode. File structure of the project contains a number of sections:

Design Resource - project description (project file * .dsn, individual scheme pages, list of Design Cache components, VHDL files, list of used component libraries * .olb);

Outputs - design results;

PSpice Resource - information for modeling with PSpice (Include Files, Model Library, Simulation Profiles, Stimulus Files) etc.

Double-clicking the left mouse button on the name of a specific file or on its icon loads it into the corresponding editor (when selecting a scheme file, the scheme editor is loaded, when selecting a text file - the built-in text editor). Right-

clicking on the icon of a separate file or directory opens a menu, the composition of which depends on the type of selected object:

Add File - adding a file;

Part manager to download the component manager;

Edit - file editing;

Properties - viewing and editing object properties;

New Schematic - creation of a new schematic;

Design Properties - editing project parameters;

Save - save the changes made;

Save As ... - save the changes made in the project with a new name;

Simulate Selected Profile (s) - performing simulation using PSpice according to the selected profile (simulation task file);

View Simulation Results - viewing graphical results of simulation;

View Output File - viewing a text file and simulation results;

Edit Simulation Settings - editing the simulation task;

MakeActive - activation of the selected profile;

New Page - adding a new page of the scheme;

Edit Page - editing the scheme page;

Schematic Page Properties - editing the setting parameters of the schematic editor;

Edit Selected object properties - editing the attributes of the object selected in the scheme;

Make Root - move the selected scheme to the top level of the hierarchy;

Rename - file renaming.

Scheme editor. Figure 5.5 shows the schematic diagram page editor window where additional toolbars are located (fig. 5.6), the commands of which are listed in the table. 2.2 and 2.3.

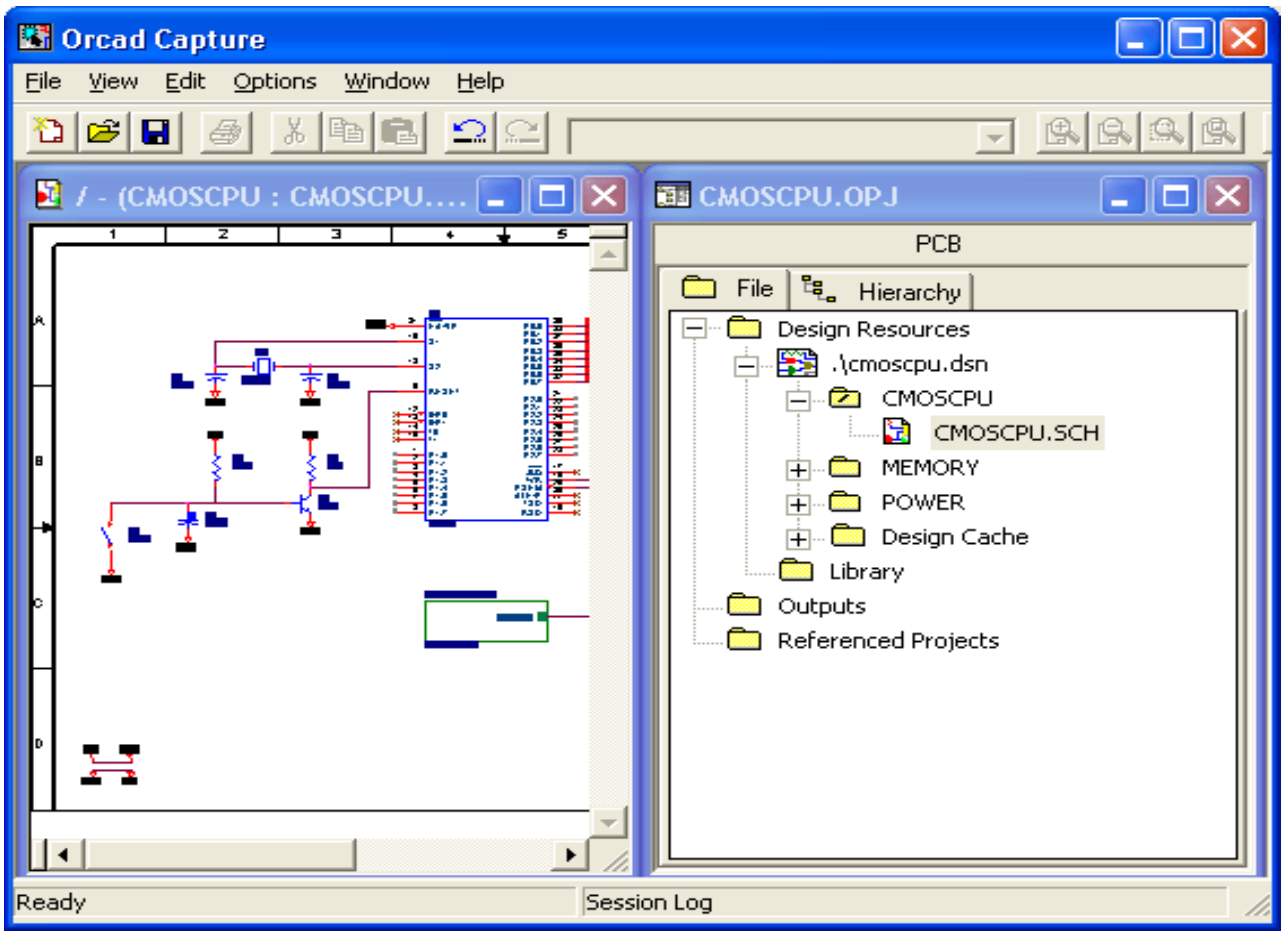


Fig. 5.5. Schematic page editor window

Fig. 5.6. Schematic Editor Toolbar

Table 5.2. Scheme editing mode toolbar icons

Icon	Equivalent command	Command description
	Select	Object selection mode
	Part	Selecting a component in the library to place its symbol on the diagram
	Wire	Drawing electrical circuits. By pressing the Shift key it is possible to enter non-orthogonal chains
	Net Alias	Placement of aliases (additional names) of chains and buses
	Bus	Bus image (group communication line)
	Junction	Plotting the electrical connection point of two circuits
	Bus Entry	Applying the taps of bus axes, located at an angle of 45 °
	Power	Placement of the symbols of power sources and "ground"pins
	Ground	Placement of the symbols of power sources and " ground "pins
	Hierarchical Block	Placement of hierarchical blocks
	Hierarchical Port	Placement of hierarchical block ports
	Hierarchical Pin	Placement of hierarchical block pins
	Off- Page Connector	Placement of page connector symbols
	No Connect	Connection to symbol component pin no connections
	Line	Line drawing
	Polyline	Drawing a broken line
	Rectangle	Drawing a rectangle
	Ellipse	Drawing an ellipse / circle
	Arc	Drawing an arc
	Text	Placement of one or more lines of text with the definition of its size, color, orientation and font

	New Simulation Profile	Creating a new simulation task file
	Edit Simulation Settings	Editing a simulation task
	Run PSpice	Launch the Pspice program
	View Simulation Results	View graphical simulation results
	Voltage/Level Marker	Placement of the voltage / logic level marker
	Voltage Differential Markers	Placing two voltage difference markers
	Current Marker	Placing two voltage difference markers
	Power Dissipation Marker	Setting a scattering marker
	Enable Bias Voltage Display	Displaying nodal voltages at the operating point on the diagram
	Toggle Voltage On Selected Net	Show / delete the value of the DC potential of the selected circuit
	Enable Bias Current Display	Display the branch currents at the operating point on the diagram
	Toggle Current On Selected Part/Pin	Show / delete the DC current value of the selected component pin
	Enable Bias Power Display	Display of the dissipated power of the branch at the operating point on the diagram
	Toggle Power On Selected Part	Show/delete the DC power dissipation value of the selected component

Text Editor. The text editor allows you to create and view VHDL files and any other text files. A fragment of the VHDL file, keywords in which and comments are highlighted for clarity in different colors specified in the Preferences section of

the Options menu is shown in fig. 5.7. Downloading a VHDL file to the editor is performed after double-clicking the left mouse button while placing the cursor on the file name in the project manager, text files of other types are opened in the usual way using the File> Open> Text File command.

Status line. At the bottom of the Capture screen, there is a status line (fig. 5.8), which displays the name of the selected tool or menu, the name of the current state of the program (in the left field), the number of selected objects (in the middle field), the image scale and the current coordinates of the cursor (in the right field).

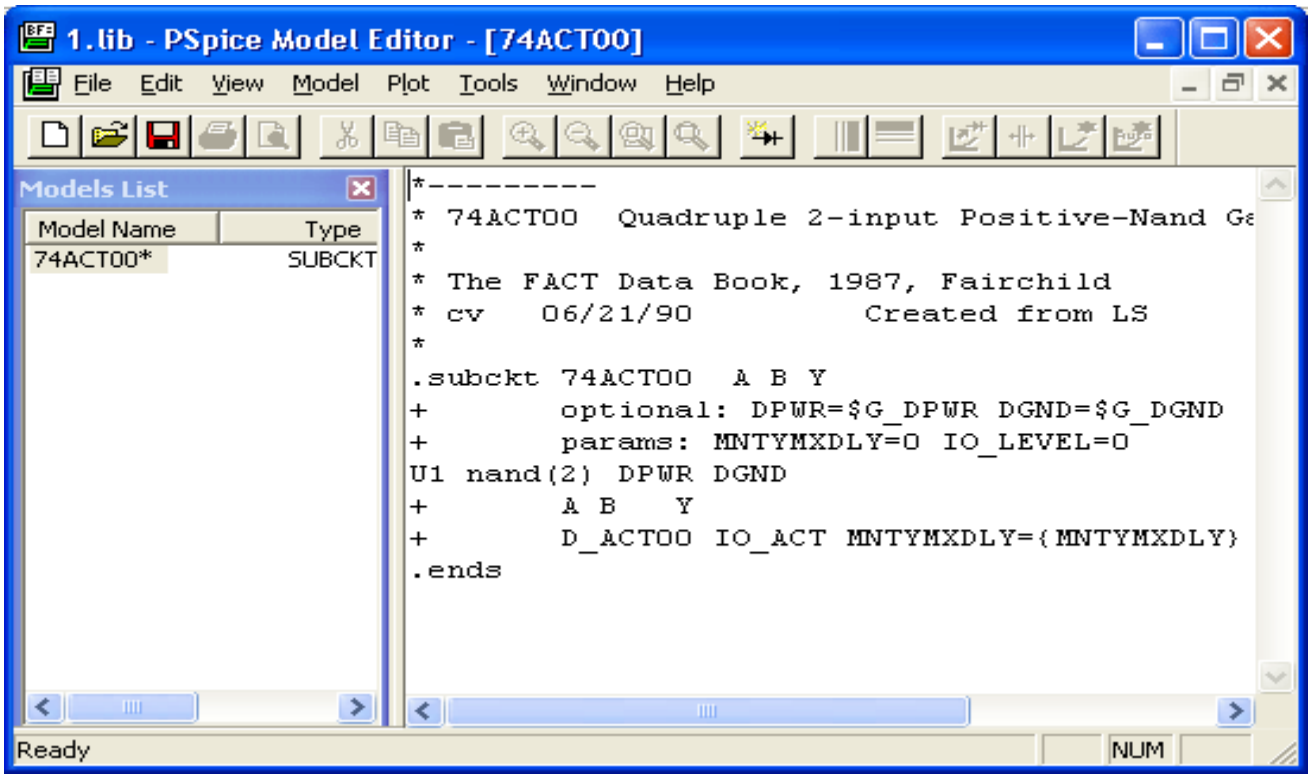


Fig. 5.7. Symbol Editor toolbar

Selection of objects. After selecting an object or a group of objects, you can perform various operations, including moving, copying, deleting, mirroring, rotating, rescaling and editing. When editing text files, including VHDL-files, standard methods of selecting objects, adopted in MS Word and similar programs are used.

When editing graphic files (schematic diagrams and symbols of individual components), a separate object is selected by clicking the left mouse button when placing the cursor on the selected object (switching to the selection mode is automatically indicated on the toolbar by highlighting the icon) Canceling the selection of the object is carried out by clicking the mouse while placing the cursor on an empty place on the screen. Adding an object to the selected group of objects is performed by clicking the left mouse button while holding down the CTRL key. Deleting an object from the selected group is also done by holding down the CTRL key.

In addition, it is possible to select objects located in the window (at the same time, before "dragging" the window by moving the cursor with the left mouse button pressed, you must enable the selection mode by clicking on the icon). Selection of all objects of the schematic sheet is carried out using the Edit> Select All command. Overlapping objects are selected by holding down the Tab key.

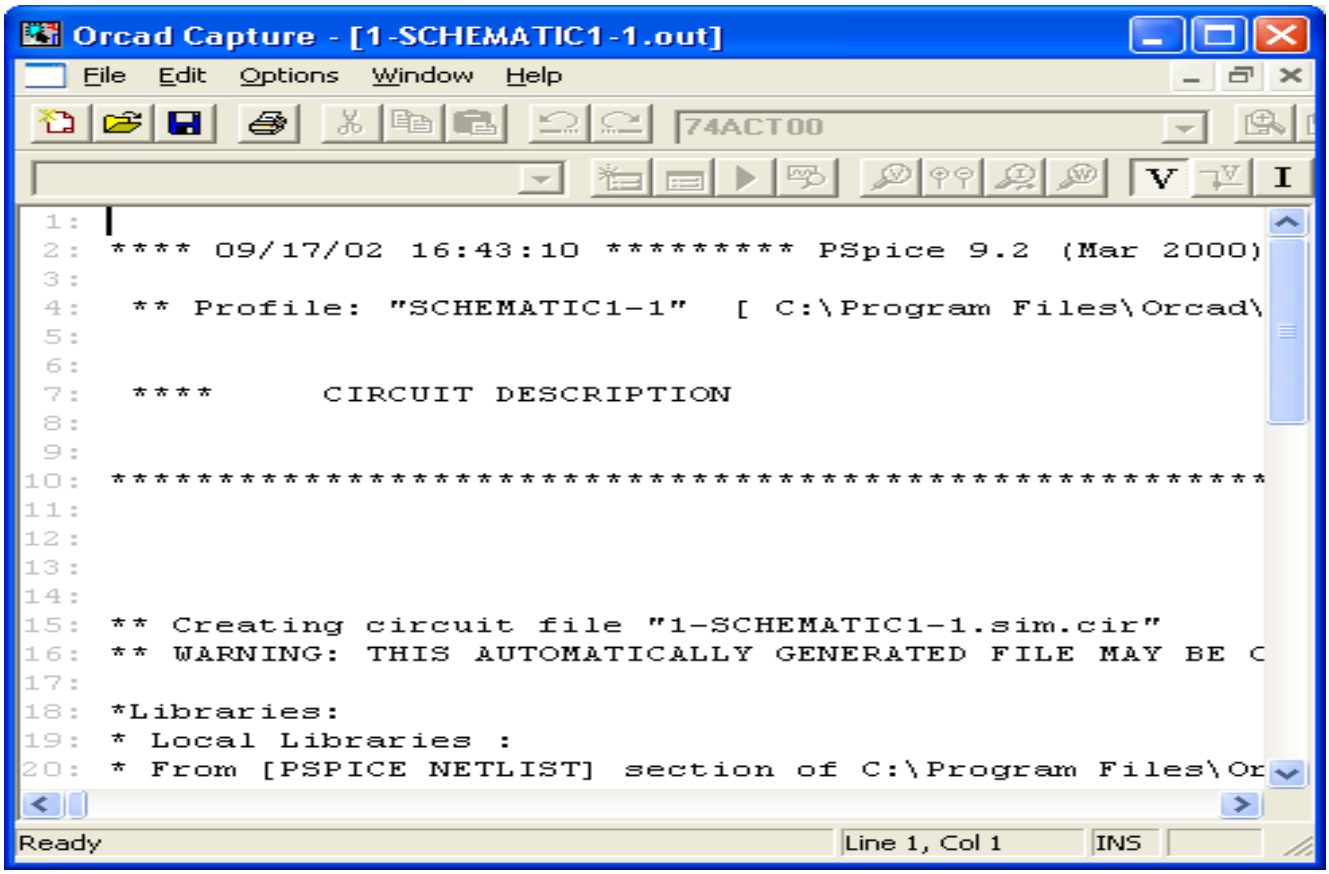


Fig. 5.8. Viewing and editing VHDL files

Editing object properties. Each schematic diagram object has a set of properties, which completely determine its characteristics. These objects include:

Hierarchical ports - outputs of the hierarchical component;

Off – page connectors - page connectors;

DRC marks - error symbols;

Bookmarks;

Parts - symbols of components (including hierarchical blocks)

Nets - chains;

Pins – component outputs;

Title block - the main inscription of the schematic diagram letter (corner stamp).

Each component characteristic (or attribute in DesignLab terminology) has a name and a corresponding value. For example, a bipolar transistor with the positional designation Q1 has the RSV Footprint attribute (case type) that takes the value TO206AA, the Implementation Type = PSpice Model attribute (the type of the PSpice mathematical model), the Implementation attribute (the name of the mathematical model) that takes the value KT315, etc. (See fig. 5.9).

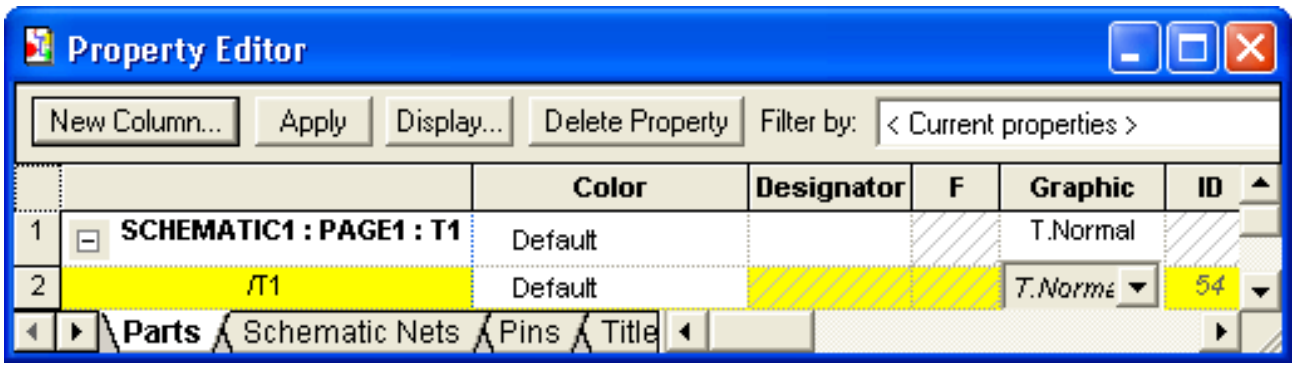


Fig. 5.9. List of characteristics of transistor Q1

The properties (attributes) of one or more circuit components are viewed and edited using the Property Editor, which is called by the command Edit> Properties (it is also activated by double-clicking the cursor on the image of the component symbol or from the context menu by clicking the right mouse button). Viewing electronic tables of properties of project objects of various types is performed using the Edit> Browse> Parts, Nets commands of the project manager (fig. 5.10). Before viewing the spreadsheet, a user must select the type of objects:

occurrence - objects that can be used multiple times in a project

instance - individual objects placed in the current project (mostly).

Only the properties of occurrence type objects can be edited in spreadsheets, the properties of instance type objects can be edited only using the Property Editor.

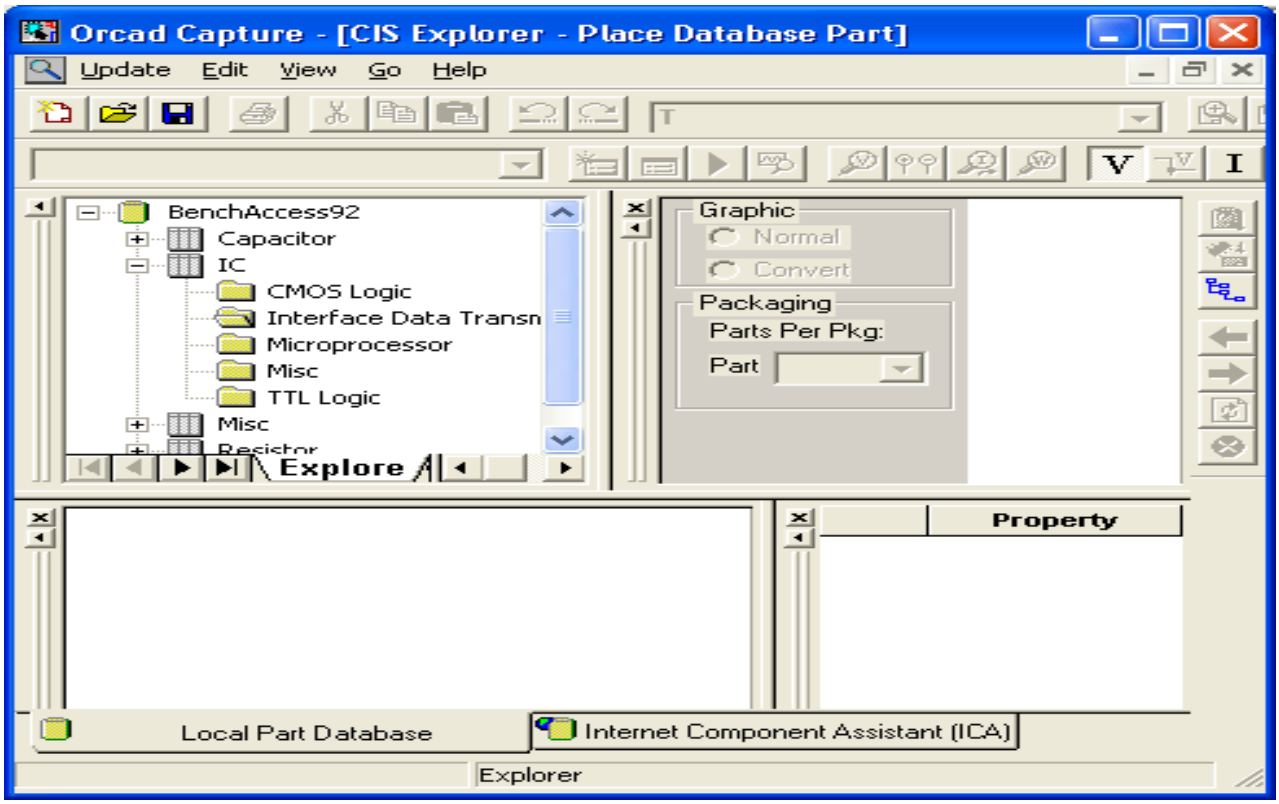


Fig. 5.10. Viewing project characteristics in a spreadsheet

Moving and resizing graphic objects. Some graphic objects, such as conductors, buses (group communication lines), lines, ellipses (especially circles), rectangles and polygons can be resized and shaped. All other objects can only be moved, rotated, mirrored and deleted. The edited objects must be selected in advance - as a result, special icons are displayed on the screen for each selected graphic object (fig. 5.11). To change the shape or size of graphic objects, you need to click the left mouse button while placing the cursor on one of these icons and then, without releasing the button, move the cursor accordingly; editing is completed by releasing the left mouse button.

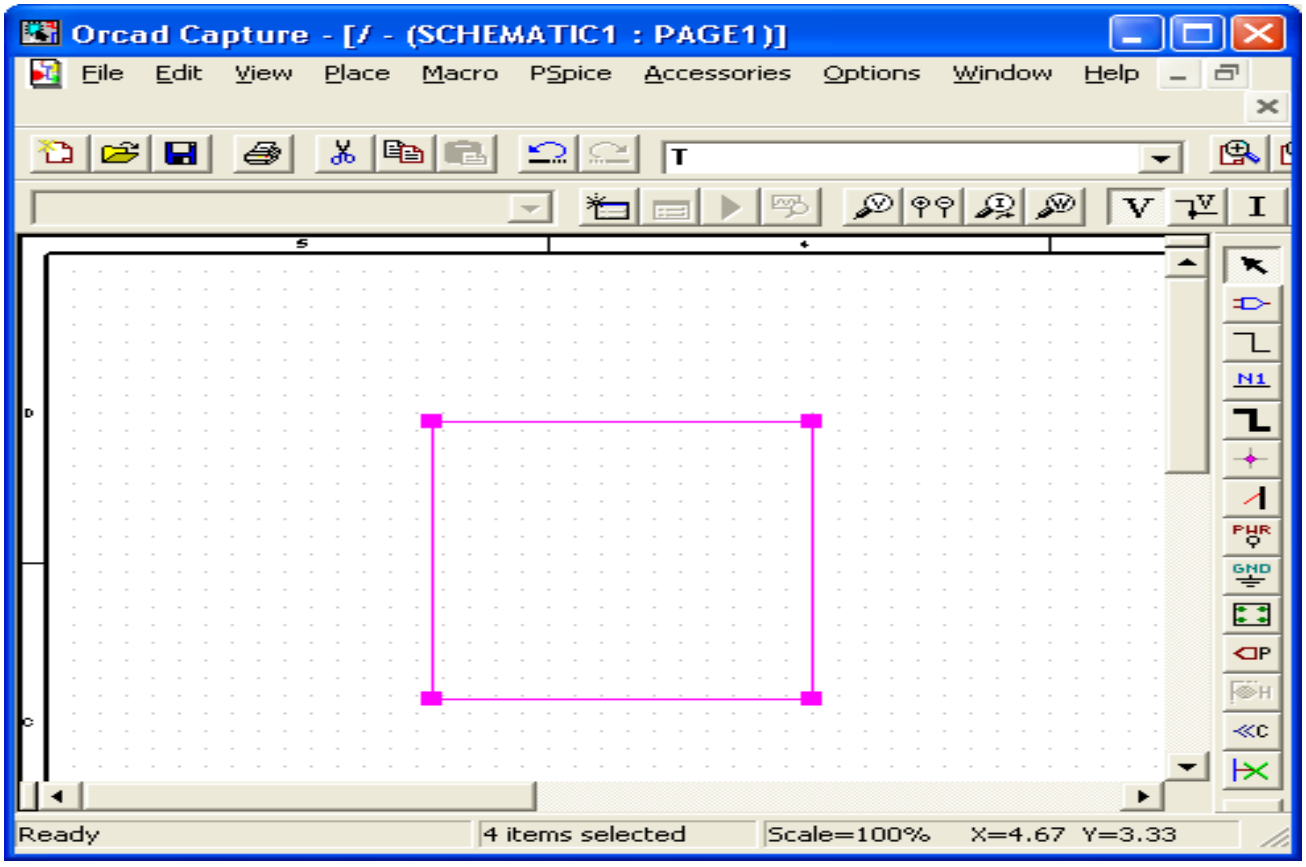


Fig. 5.11. Objects resizing

To move any selected object, you need to click the left mouse button while placing the cursor at any point on the object's outline, except for these icons, and then, without releasing the button, perform the movement. When moving a group

of objects, the cursor changes its shape (it takes the form of a star), and it can be placed at any point inside the contour bordering the selected group.

Roll "forward and backward is the repetition of the last operation. The Edit> Undo command cancels the execution of the last command ("rollback"), while the name of the last executed command Place, Delete, Copy, Past, Move, Resize, Rotate, Mirror is automatically added to the name of the Undo command in the Edit submenu, e.g. Undo Place Edit> Redo command undoes the Edit> Undo command ("rollback" forward).

Repeating the execution of the last Place, Copy, Past, Move, Resize, Rotate, Mirror command is performed using the Edit> Repeat command, while the name of the last executed command is automatically added to the name of the Repeat command in the Edit submenu. To shift the copied object to a specified distance, before executing the Edit> Repeat Copy command, select the copied object, press the CTRL key and without releasing it, move the copied object to the required distance. After that, successive execution of the command Edit> Repeat Copy (F4) creates an array of copied objects, offset from each other by a given distance (it is convenient, for example, when creating buses).

Projects created using the OrCAD Capture program are saved in files with the extension .opj (according to the terminology adopted in the program, the project is called Project), which contain links to the names of all the files used: files of individual circuits (* .dsn, according to the terminology accepted files schemes are called Design, also translated as "project"), libraries, text VHDL files, project report files, etc. A project file may contain links to one or more folders (these folders are displayed in the project manager window, see fig. 5.2), which are associated with the schematic diagram files. A schematic folder contains one or more schematic pages. The scheme file also contains a Design cache - a cache of the project, which contains copies of the symbols of the components used in the scheme. A project can contain links to multiple libraries. However, it can have only one schema (a file with the name extension .dsn) consisting of one or more pages. You can create a new

project and then create new circuits, libraries and VHDL files. To create a new project, the File> New Project command is executed, after which in the dialog box (fig. 5.12) the name of the project is indicated on the Name line (Cyrillic characters are not allowed if modeling is assumed), and the name of the subdirectory of the location is indicated on the Location line project (in this case, it is convenient to use the Browse button to view the file structure). Next, the project type is selected in the middle part of this window.

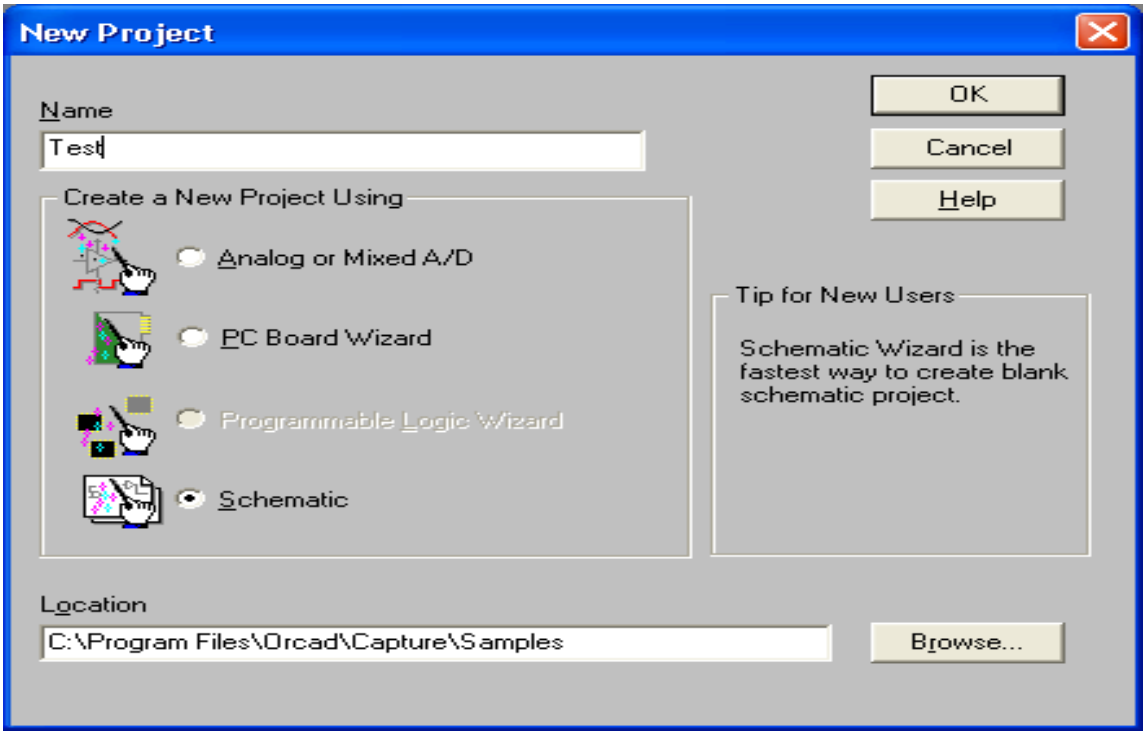


Fig. 5.12. Display Invisible Power Pins

Analog or Mixed-Signal Circuit - analog, digital or mixed analog-digital devices modeled using the PSpice A / D program (further development of the printed circuit board using OrCAD Layout is also possible). At the beginning of the creation of the project you can download the prototype by specifying its name in the image shown in fig. 5.13, and in the dialog box (Create based upon an existing project option) it is possible to download one of the 4 standard prototypes or any previously created project.

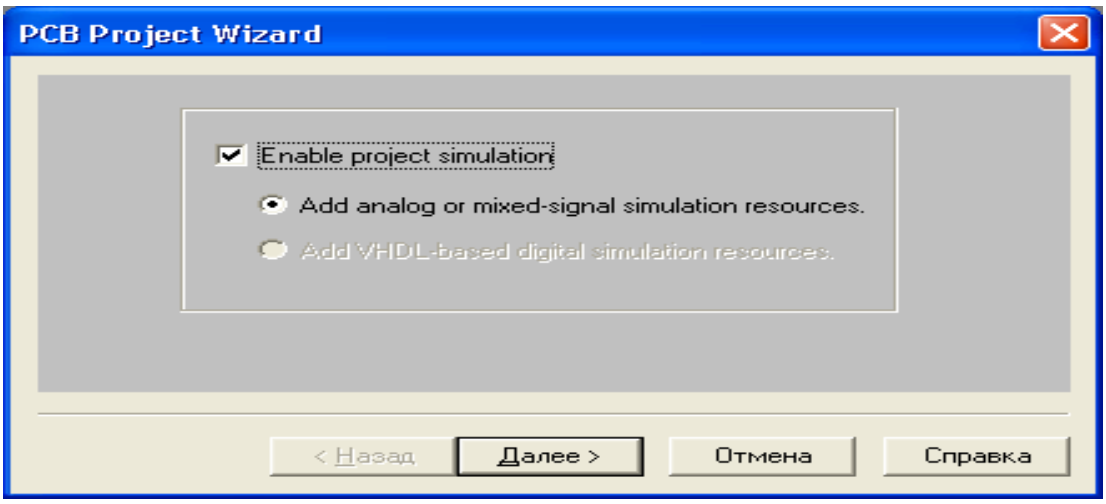


Fig. 5.13, a) Selection of a project prototype (a) or RSV-project modeling capabilities (b) with a selection of Pspice symbol libraries

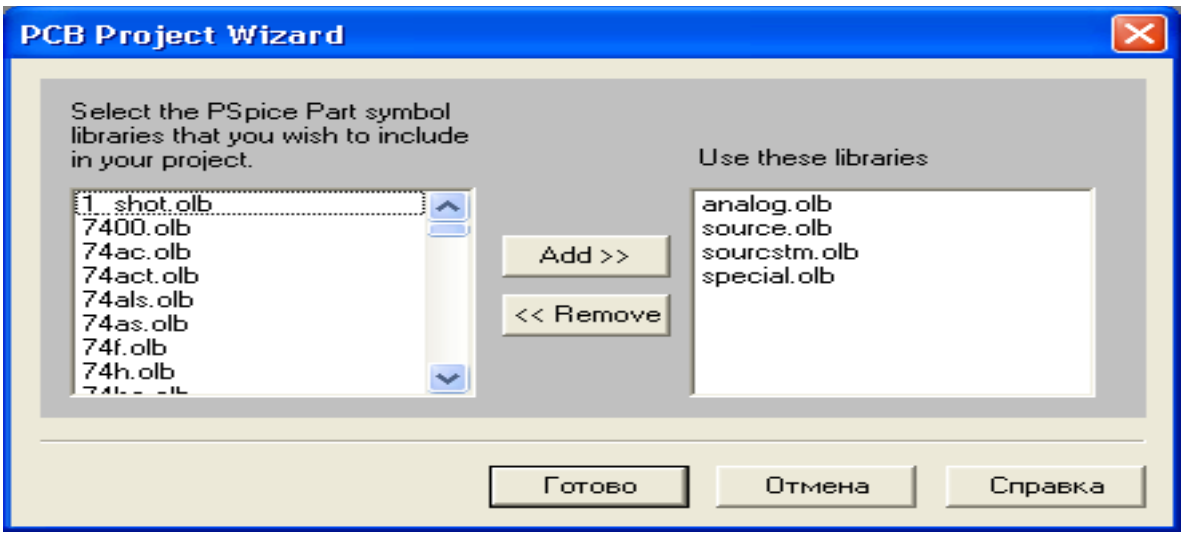


Fig. 5.13, b)

PC Board - printed circuit boards (simulation of mixed analog-digital devices is performed using PSpice). The need for simulation is indicated in the dialog box shown in fig. 5.13, b - for this, you need to check the Enable project simulation line in order to add the list of PSpice symbol libraries (Add analog or mixed-signal simulation resources) to the project.

Schematic are not specialized projects (only the creation and documentation of schematic diagrams is possible, modeling and development of printed circuit boards is not provided for).

Remark.

The choice of the project type determines the set of OrCAD Capture menu commands, which is not very fundamental, because there is a possibility of exchanging data between any projects.

Standard Windows dialog boxes: Print Setup, Print Preview and Print are used to output data to a printer, plotter, or PostScript file (*.prn). Output commands can be selected from the File menu of the project manager or from the OrCAD Capture, OrCAD PSpice, etc. programs. You can print out a schematic page, component symbol, packaging information and text files, etc. in the following order:

1. The Print Setup command of the File menu configures the printer/plotter;
2. Using the Print Preview command of the File menu, the parameters are set (fig. 5.14):
   Scale to paper size: automatic scaling of the image so that all selected pages of the project diagram fit completely on a sheet of paper of the size specified below Scale to page size: automatic scaling of the image so that the selected page of the scheme fits completely on a sheet of paper of the size specified below;
   Page size: - A4, A3, A2, A1, A0, Custom (set by the user);
   Saling: image scale;
   Print offsets X, Y: image shift horizontally and vertically;
   Print Quality: print quality (resolution);
   Copies: number of copies;
   Print to File: output images to a file;
   Collate copies: the order of printing copies - first all copies of the first page are output, then all copies of the second, etc.;
   Force Black & White: black and white image output.
3. Selection of printed pages of the scheme is carried out in different ways:
   one or more scheme pages are selected in the project manager window;
   the required circuit page is loaded into the circuit editor;

to print all pages of the project scheme just select the name of the project scheme (Design) in the project manager.

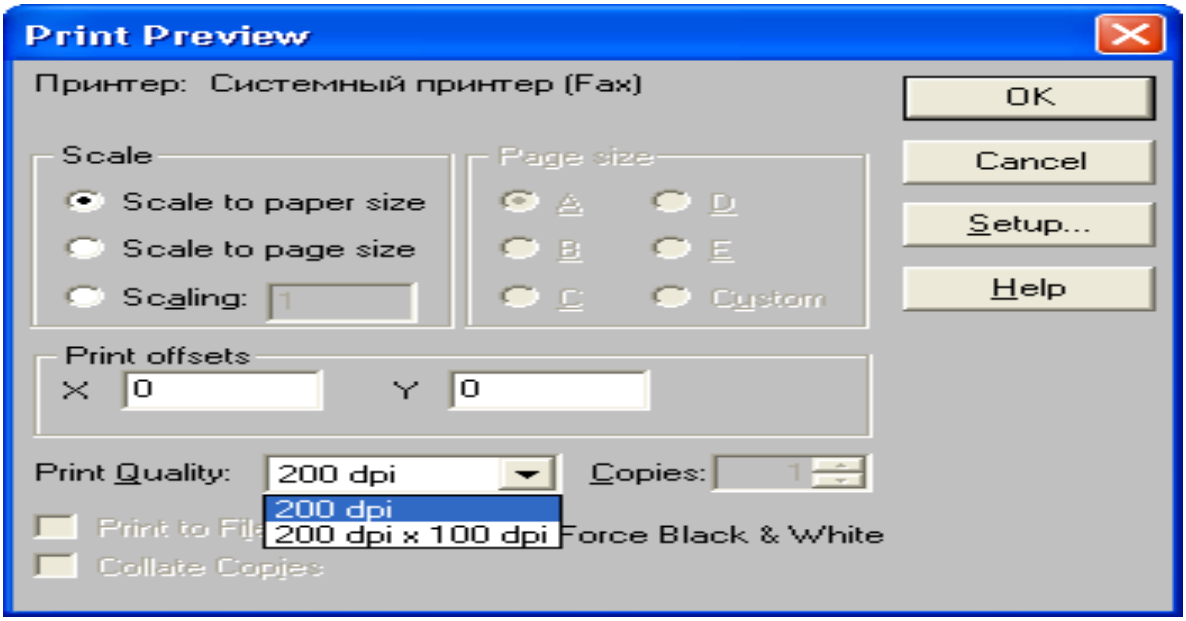


Fig. 5.14. Setting print options

Electrical diagrams of most projects are placed on several pages of a not very large format. There are two ways to organize large-volume diagrams: flat, conventional multi-page structures and hierarchical structures.

Electric circuits located on different pages of a multi-page circuit are connected to each other using the so-called off-page connectors, which have the same names. All pages of such schemes are contained in one folder at the same level. Their structure is shown in the project manager when you press the File key.

Special symbols called hierarchical blocks are placed on the diagrams of hierarchical projects. The circuit diagram of each such block is placed in the form of a separatediagram, which is placed in a folder at the same hierarchy level as the main diagram.

Before creating a new project using the OrCAD Capture program, it is necessary to set its configuration parameters using three commands in the Options menu of the project manager:

- the Preferences command specifies the scheme parameters that are saved in the Capture.ini configuration file and which are not initialized each time the OrCAD Capture program is started; changes to these parameters are made in already existing schemes; if a project is created in another OrCAD system, the parameters contained in the current Capture.ini file will be taken into account;

- the Design Template command sets the parameters of the scheme, which are set by default when creating all new projects (they are entered in the [Design Template] section of the Capture.ini file); changes to these parameters are not made in already existing schemes, therefore, before creating new schemes, it makes sense to review and, if necessary, change their values;

- the Design Properties or Schematic Page Properties command sets the parameters of the individual current scheme.

Let's consider these project configuration methods in detail.

On the Color / Print tab, the colors of all scheme objects are viewed and set (using a palette, which can be opened by clicking the left mouse button on a painted rectangle) and the objects that should be printed are marked (to do this check the box next to the the object name the title block color (corner stamp, Title Block) is also assigned to the drawing frame (Border) and the the grid lines ticks on the drawing frame (Grid Reference); the colors of graphic objects (lines, polygons and arcs) are set on the Miscellaneous tab, if default colors are specified on this tab, then they are set in accordance with the graphics color (Graphics) on the Color / Print tab.

Select the style of grid images in the form of dots (Dots) or lines (Lines) separately for the Schematic and Symbol editor on the Grid Display tab; the Displayed panel indicates the need to display the grid on the display screen, and

the Pointer snap to grid panel emphasizes the need to "bind" the cursor to grid nodes when placing objects on the diagram.

The Pan and Zoom tab specifies the image zoom factor (Zoom Factor) and the panning factor (Auto Scroll Percent) for the diagram and symbol editor (the diagram panning, i.e. its displacement without changing the scale, is carried out by the cursor approaching the border of the working window, if the left mouse button is pressed and held).

On the Select tab, it is set whether objects will be selected if the border of the selection rectangle crosses them (Intersection) or if they are completely inside the selection area (Fully Enclosed); the Maximum number of objects to display at high resolution while dragging panel indicates the maximum number of objects displayed on the screen when they are selected in the window and moved.

On the Miscellaneous tab, you can select the style of filling closed shapes (Fill Style), the style and width of lines (Line Style and Width) and the color of graphic objects (Color), as well as the font used in the project manager and the Session Log protocol file; in addition, the following parameters are set:

-Render True Type Tonts with strokes - images of True Type fonts in the form of stroke vector fonts (for printing);

-Fill text - font filling;

-Enable Auto Recovery - automatic saving of project files, schemes and VHDL-texts in the \ WINDOWS \ TEMP \ AUTOSAVE directory;

-Update every xxx minutes - file auto-save interval in minutes;

-Automatically reference placed parts - automatic assignment of positional designations of components placed on the diagram;

-Enable Intertool Communication (ITC) - enabling the check mode and display of results on the screen when transferring data from other OrCAD system

programs, such as OrCAD Layout and OrCAD PSpice; for example, when the ITC mode is enabled between the OrCAD Capture and OrCAD Layout programs, a "hot" connection (cross probing) is established between the circuit and the printed circuit board.

The text editor used when working with VHDL files is configured on the Text Editor tab. The Syntax Highlighting panel specifies the highlighted colors of Keywords, Comments, lines enclosed in Quoted Strings and Identifiers. On the Current Font Setting panel, after pressing the Set button, the font size of the text is set and the color of the objects is not highlighted. The specified objects are highlighted when selecting the Highlight Keywords, Comments, Identifiers, and Quoted Strings panel. The tab spacing of the text editor is specified in the Tab Spacing panel.

The Design Template command defines a set of parameters for new projects, some of which can be reassigned to individual scheme pages. Tabs of dialog windows of this command are shown in fig. 2.21.

The fonts of the texts of various objects located on the diagram are defined on the Fonts tab (fig. 5.15, a).

On the Title Block tab (Fig. 5.15, b), the text entered in different columns of the title block (corner stamp) is defined. In general, there are two main types of the title blocks: accepted by default and individual. The information entered in the main title block by default is specified in the dialog box of the Design Template command; in this case, the main inscription is located in the lower right corner of the new scheme page (if the Library Name is correctly specified in the Title Block tab and if it is the CAPSYM.OLB library, then its name does not need to be specified).

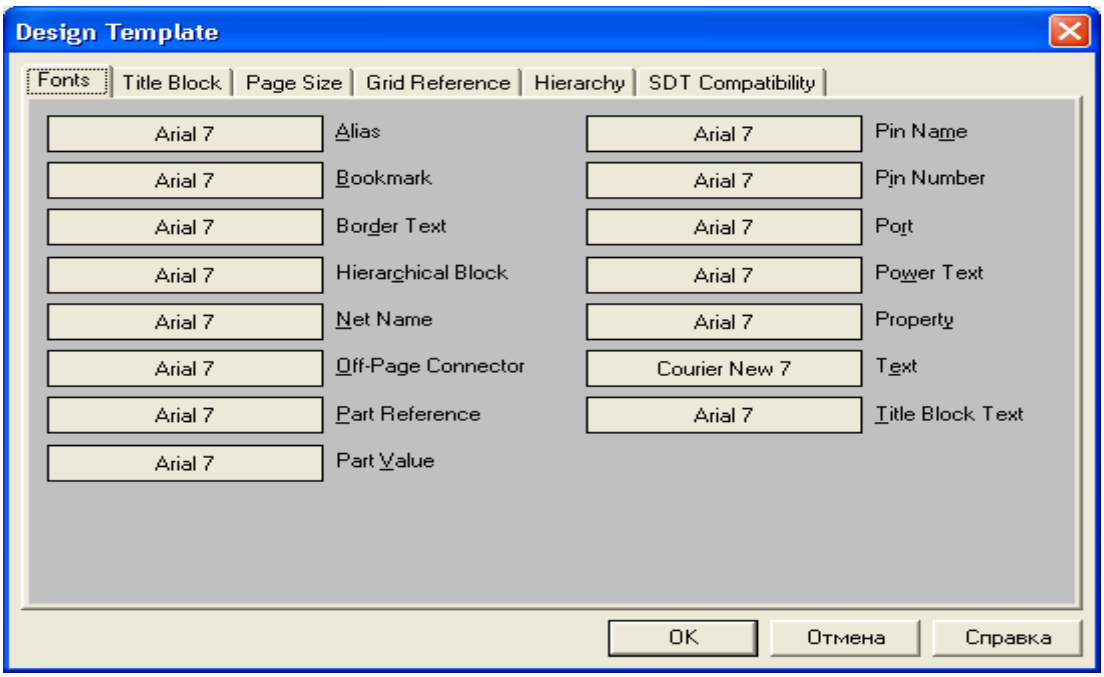


Fig. 5.15, a) Dialog window of the Options> Design Template command

The name of the graphic symbol of the main inscription is specified in the Title Block Name line. Editing of the text entered in the main inscription for each scheme is performed in the scheme editor using the Edit> Properties command. Individually, the main inscriptions are placed on the diagram using the Place> Title Block command.

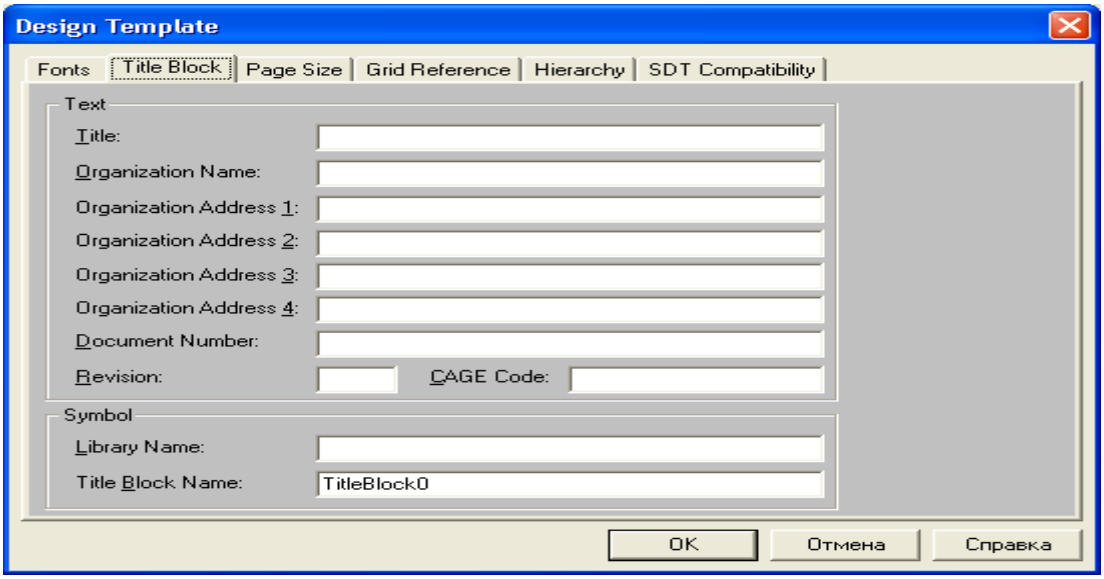


Fig. 5.15, b)

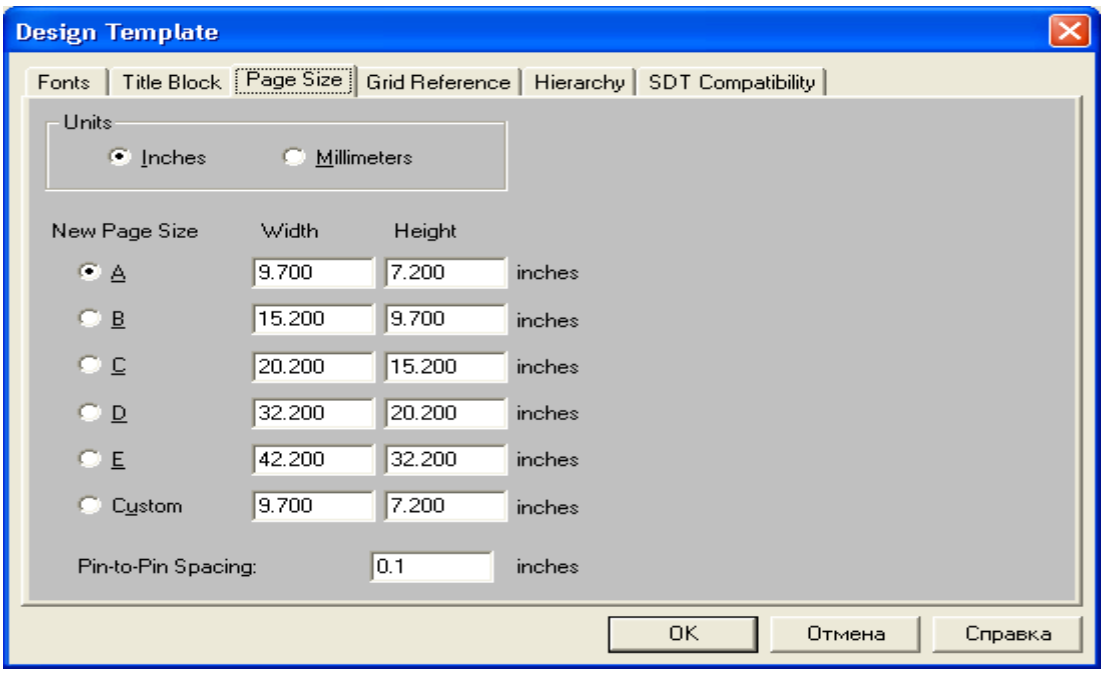


Fig. 5.15, c)

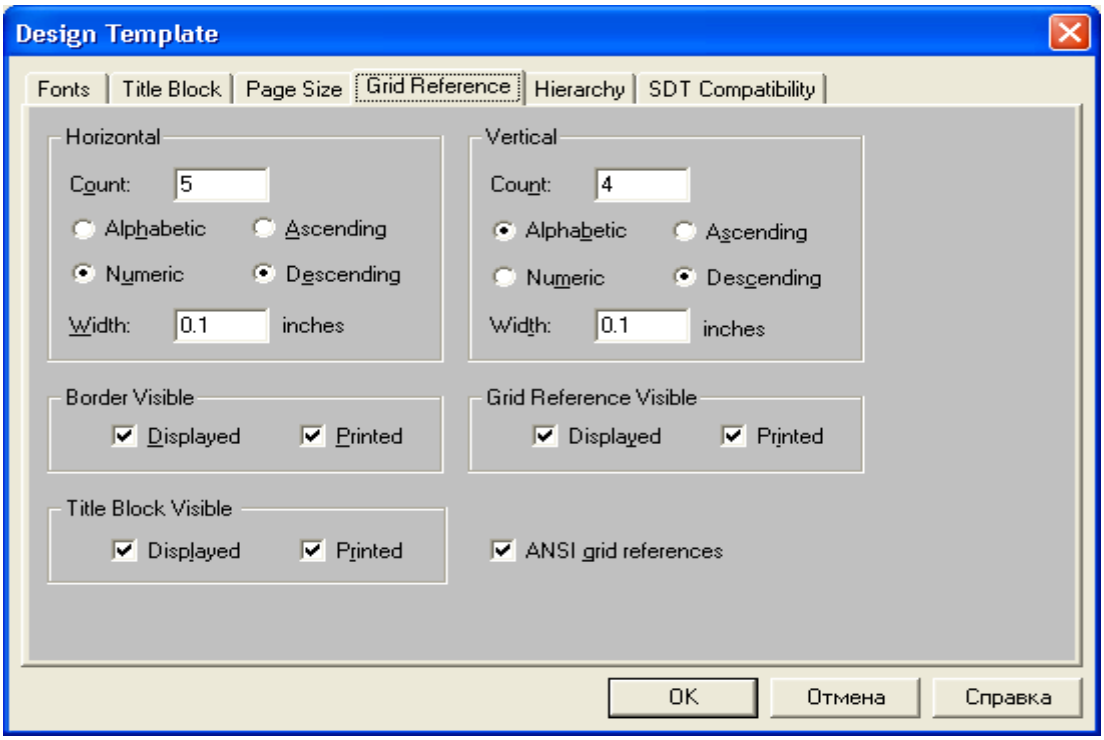


Fig. 5.15, d)

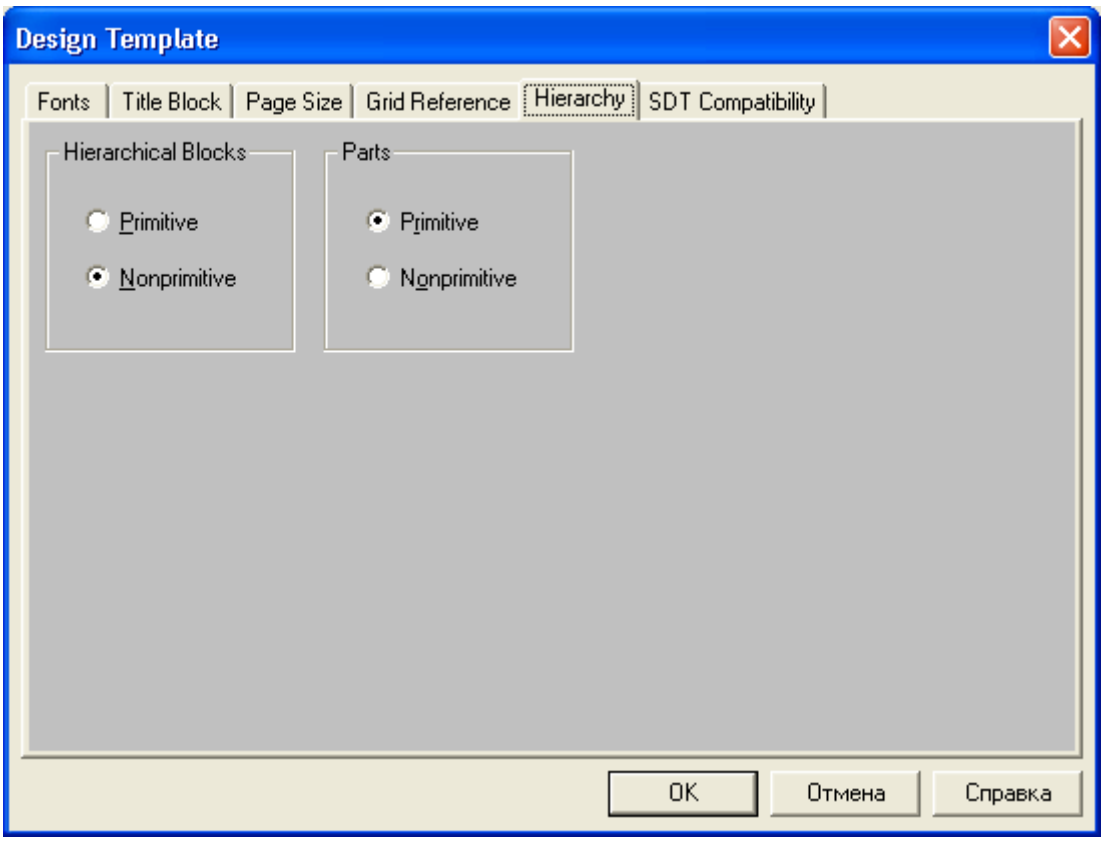


Fig. 5.15, e)

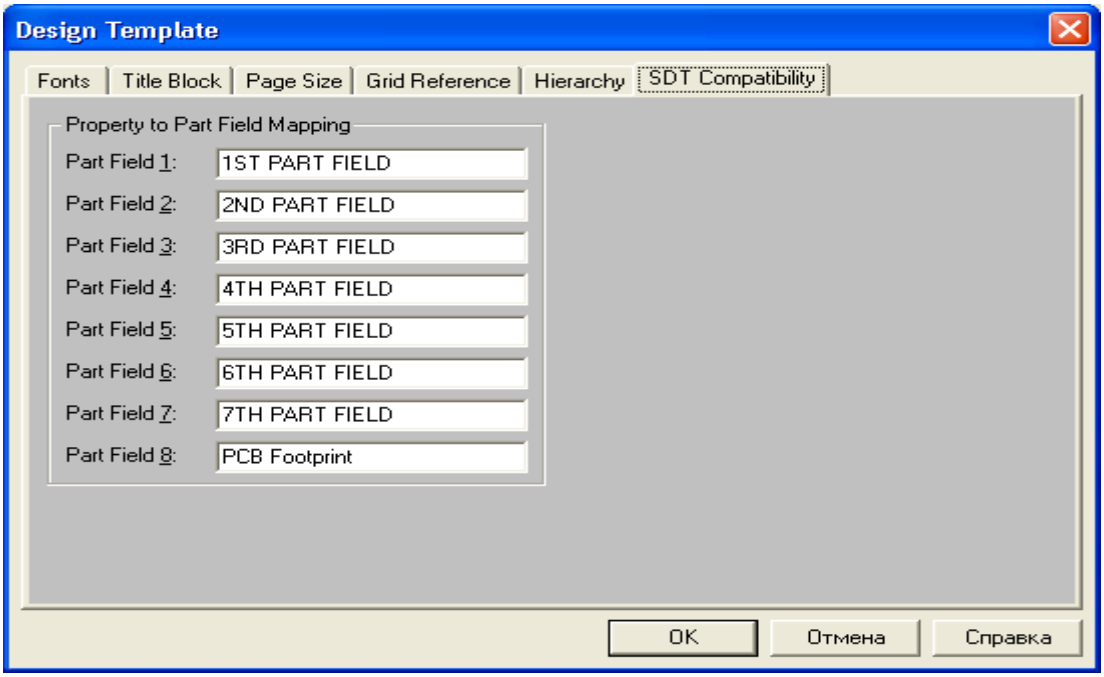


Fig. 5.15,f)

The Page Size tab (fig. 5.15, c) specifies the system of units set by default (Inches or Millimeters) and the sheet size of the scheme A, B, C, D, E (in the English system), A4, A3, A2, A1, A0 (in the metric system) or Custom (sizes are set by the user). The Pin-to-Pin Spacing column indicates the minimum distance between the pins of the components when they are located on the diagram, at the same time this parameter determines the size of the grid step (note that for an existing project or an individual scheme, the grid step cannot be changed, it is set before creating the project, which is rather inconvenient). Please note that when changing the Pin-to-Pin Spacing parameter, the sizes of symbols automatically change when they are placed on a new scheme.

The parameters of the frame located around the scheme sheet are set on the Grid Reference tab (fig. 5.15, d):

-Count - the number of graphs on the frame horizontally and vertically;

-Alphabetic - graph numbering in alphabetical order;

-Numeric - graph numbering in numerical order;

-Ascending - setting the frame graph numbers in ascending order;

- Descending - setting the frame graph numbers in descending order;

-Width - horizontal and vertical width of the frame;

-Border Visible –page borders visibility on the display (Displayed) and when printed (Printed);

-Grid Reference Visible - visibility of the scheme sheet frame on the display (Displayed) and when printed (Printed);

-Title Block Visible - visibility of the main title block on the display (Displayed) and when printed (Printed);

-ANSI grid references - an image of the scheme sheet frame according to the ANSI standard.

The Hierarchy tab (Fig. 5.15, e) specifies the default parameters when creating new hierarchical blocks (Hierarchical Blocks) and components (Parts):

-Primitive - primitive components that do not have a hierarchical structure;

-Nonprimitive - components that have a hierarchical structure.

The SDT Compatibility tab (Fig. 5.15, e) sets the compatibility of the 8 fields of OrCAD Capture component symbol parameters with the component symbol fields in the format of the DOS version of OrCAD Schematic Design Tools (SDT 386+), which is used when saving the project in the SDT format.

Changing the parameters of the current project is performed using the Options> Design Properties command, which has the tabs Fonts, Hierarchy, SDT Compatibility, Miscellaneous (fig. 5.15, a, d, e); changing the parameters of the current schematic page is performed using the Options> Schematic Page Properties command, which has the Page Size, Grid Reference, Miscellaneous tabs.

Note that on the Page Size tab you can change the size of the scheme and the unit system and you cannot change the Pin-to-Pin Spacing parameter, and you can only view information about the project or the scheme sheets on the Miscellaneous tab.

Placement of Component Symbols of the Capture libraries contains component symbols for power and “ground” supplies. They are placed on the diagram using the Place> Part command, which is also activated by clicking on the tool menu icon. In the dialog window of this command (fig. 5.16, a), first select the name of one or more libraries from the Libraries list, the contents of which are displayed on the Part panel (to select several libraries press and hold the CTRL key).

After that, the name of the component is selected on the Part panel, the symbol of which should be placed on the diagram (if several libraries are selected, then the symbol / and then the name of the library is placed after the name of each component). A normal (Normal) or an equivalent image of logic components in DeMorgan style (Convert) is selected in the Graphic section. In the Packaging section, the section number of the component is indicated, after which the image of the selected section of the component is displayed in the window below, indicating the socket numbers of its pins (the line Parts per Pkg indicates the total number of sections).

Clicking the Add Library button opens a dialog box for adding libraries to the Libraries list, clicking the Remove Library button removes the selected library from the list. The Part Search button is designed to search for a specific component in libraries from the Libraries list. After clicking the OK button, the symbol of the selected component is transferred to the diagram. By moving the cursor, the component moves to the desired place of the diagram and is fixed by pressing the left mouse button.

After that, another copy of the same symbol can be placed on the diagram. Clicking the right mouse button opens a pop-up menu (fig. 5.16, b), which duplicates the call to the main menu commands for rotation (Rotate), mirroring (Mirror), changing the image scale (Zoom), editing component parameters (Edit Properties), etc. Completing placement of the symbol of the selected component on the diagram is done after selecting the End Mode command in this menu or pressing the Esc key.

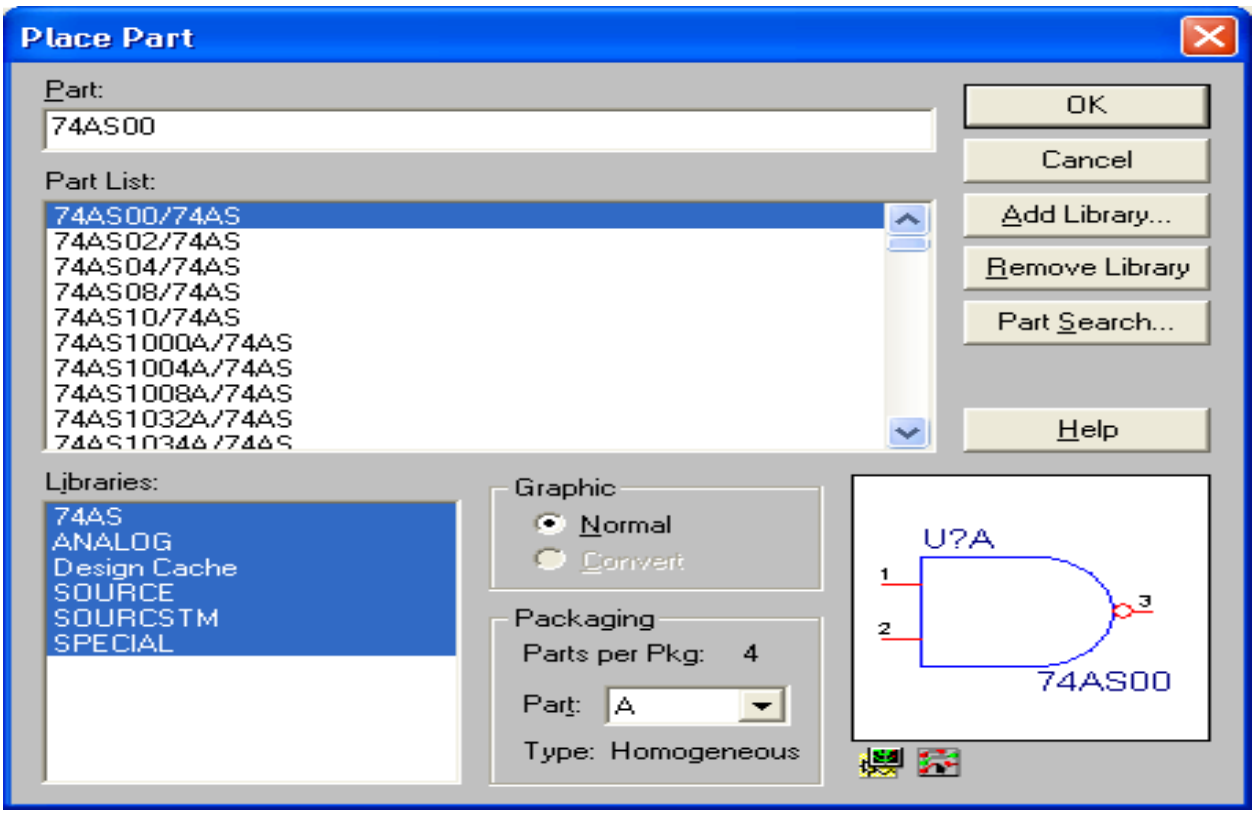


Fig. 5.16, a) Dialog box of the command Place> Part (a) and pop-up menu (b), called after selecting a component by pressing the right mouse button

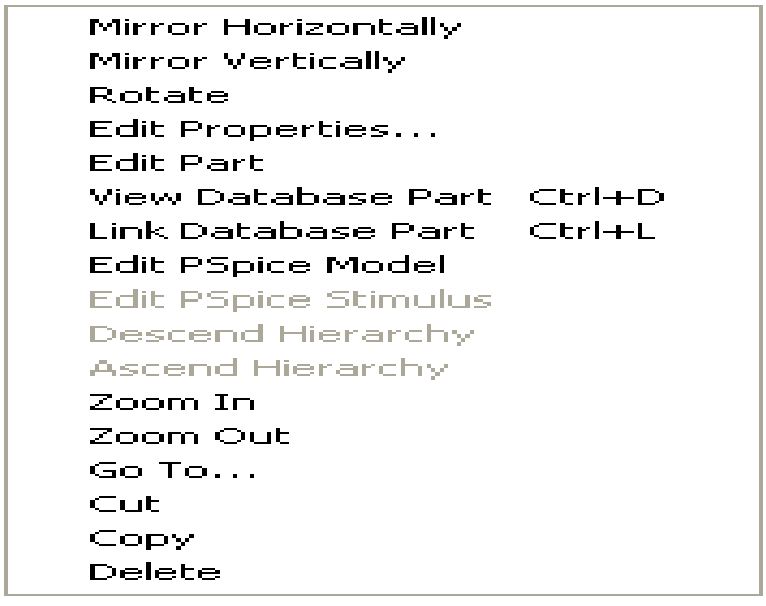


Fig. 5.16, b)

If, without interrupting the placement mode of component symbols on the diagram in the pop-up menu in fig. 5.16, b., select the Edit Properties command, a dialog box for editing the parameters of the current symbol is displayed (fig. 5.17, a). It has the following fields:

Part Value - the nominal value of a parameter of a simple component (resistance, capacity, etc., which are taken into account during modeling) or the name of a complex component (not taken into account by the modeling program);

Part Reference - positional designation of the component. It is inserted here manually, if on the Miscellaneous tab the Options> Preferences command (fig. 2.20, d) is not set to Automatically reference placed parts - automatic assignment of positional markings of components placed on the diagram (see details below). You can select or adjust the name of the component body in the RSV Footprint panel. Selecting the Power Pins Visible panel indicates the need to display the "ground" and power pins on the schematic. The type of a component is selected on the Primitive panel: Yes - elementary (primitive) component; No - a component that has a hierarchical structure, Default - is set by default (in accordance with the configuration setting on the Hierarchy tab of the Options> Design Template command (fig. 2.21, e)). The Packaging panel shows the total number of identical sections of the component and the name (number) of the current section (unfortunately, the section number is placed on the component symbol and cannot be changed on this tab).

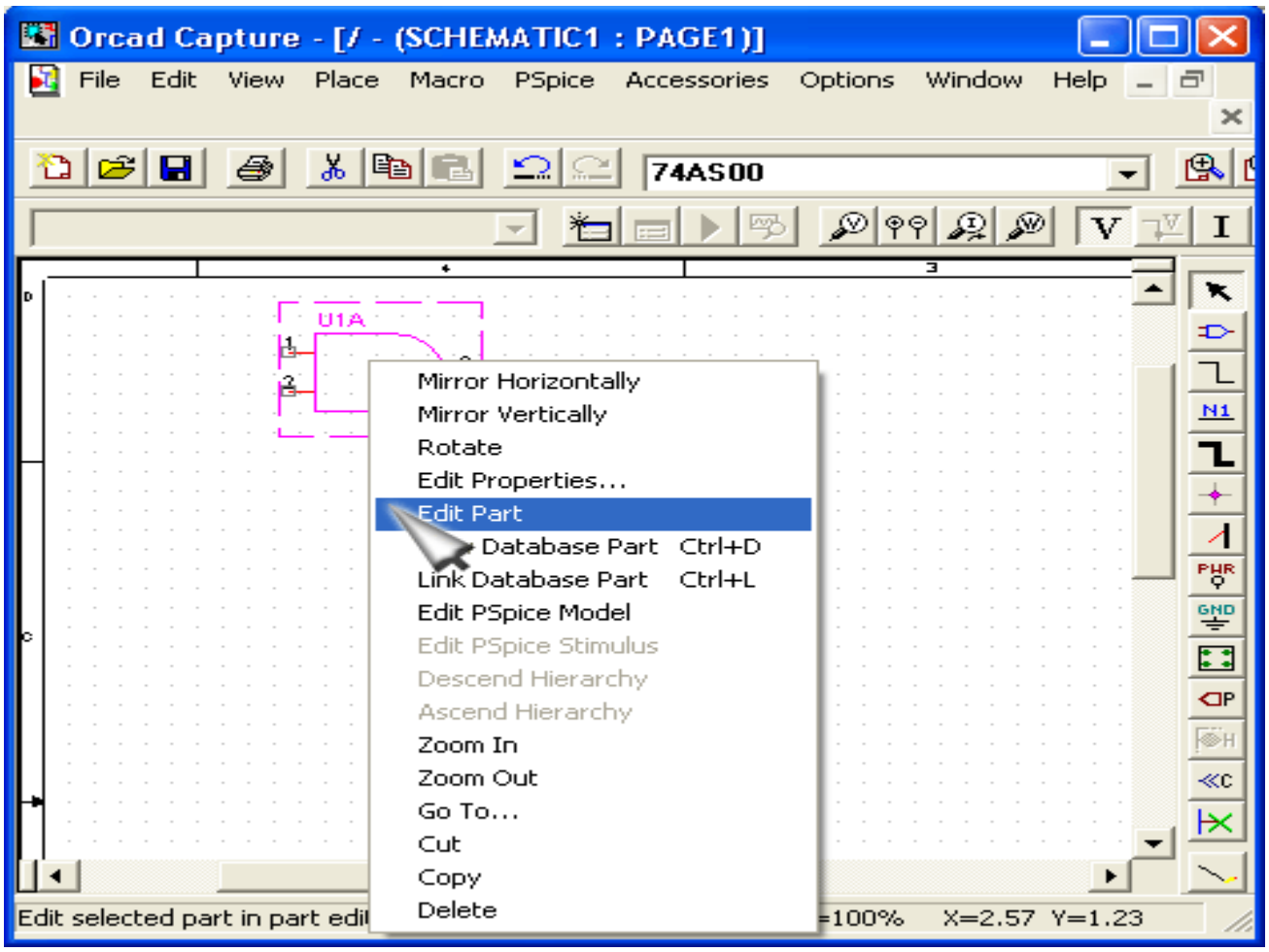


Fig. 5.17, a) Dialog windows for editing the parameters of the component symbol placed on the diagram (a), editing parameters (b), their visibility on the diagram (c) and the type of their models (d)

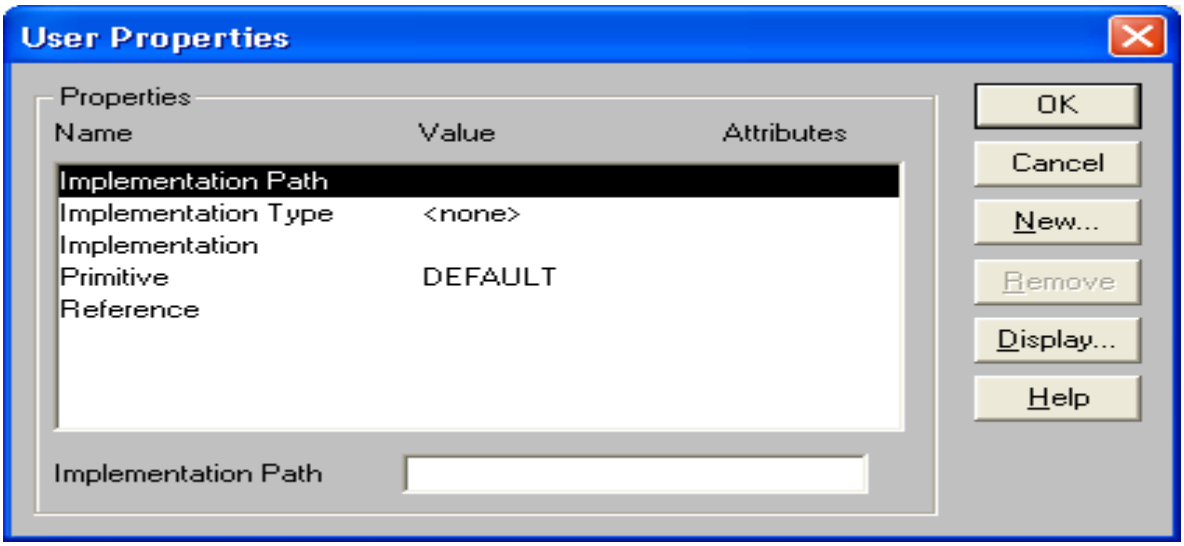


Fig. 5.17, b)

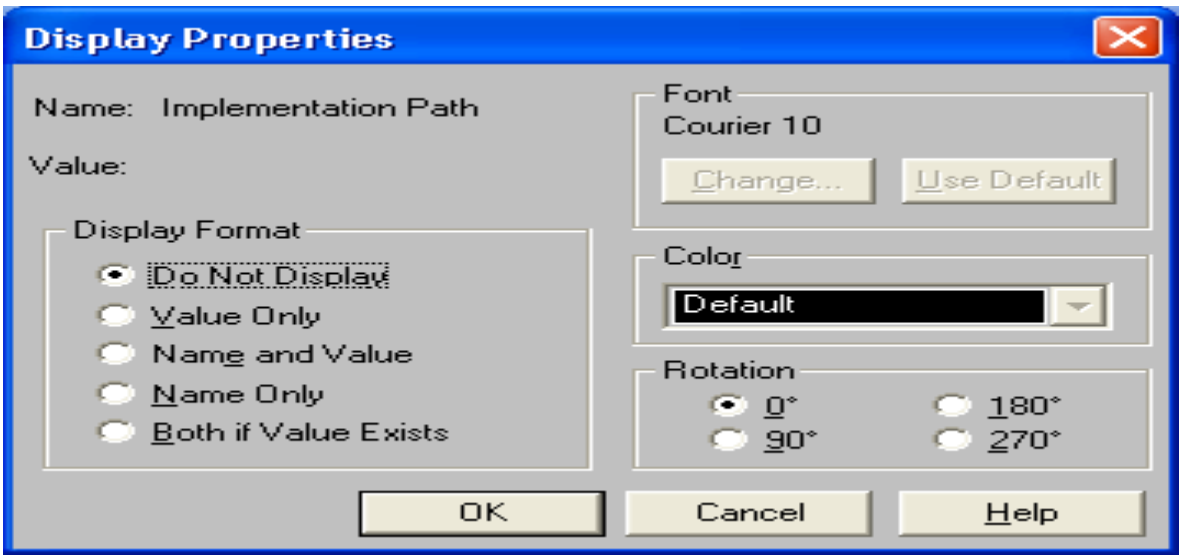


Fig. 5.17, c)

Clicking on the User Properties panel opens a dialog box for viewing and editing component parameters (fig. 5.17, b): the name of a parameter is indicated in the Name column, its value in the Value column, and the characteristics (attributes) of its display on the diagram in the Attributes column (R - read-only, V - visible on the diagram, the last feature is set in the Display panel, see below). After selecting a parameter, its name is displayed in the lower part of the window, and its value is entered in the adjacent panel (after pressing the Enter key, the entered value is displayed in the Value column) - this way, in particular, parameter values necessary for modeling with PSpice are entered (fig. 5.17, b shows the VSIN harmonic voltage source parameter dialog box); they can be entered or edited later using the Edit> Properties command (after creating the scheme using the dialog boxes shown in Fig. 5.18).

Clicking on the Display panel opens a dialog box for setting the visibility of the selected parameter on the scheme:

-Do Not Display - do not display anything on the diagram;

-Value Only - display only the value of a parameter

-Name and Value - display both the name and value of the parameter;

-Name Only - display only the name of the parameter;

-Both if Value Exists - display both the name and value of the parameter, if its value exists.

Clicking on the Attach Implementation panel opens a dialog box for viewing and editing the object type attached to the current component:

- Implementation type - the type of the attached object, which takes the value:
    - PSpice Model -a mathematical model of a component for the PSpice program;
    - PSpice Stimulus -a description of the external signal for the PSpice program;
    - Schematic View - scheme of an object;
    - VHDL - component description in VHDL language;
    - EDIF - list of connections in EDIF format;
    - Project - the scheme of the project (for it, it is necessary to additionally set the outputs of the hierarchical blocks);
- Implementation - the name of the attached object;
- Path and filename, - a full name of the file of the attached object.

The Graphic and Packaging fields of this window (fig. 5.17, a) are the same as in fig. 5.16, a.

After placing the components on the diagram, you can view the parameters of one or more components. To do this, select the components that we are interested in and double-click the mouse cursor or use the Edit> Properties command to open a spreadsheet that lists the parameters of the selected components, examples of which are shown in fig. 5.18. Similar tables contain the parameters of selected Schematic Nets, Pins and Title Blocks. Only parameters that do not have the R (read-only) attribute can be edited in these tables, see fig. 5.17,

b. Shaded cells are assigned to parameters which values are not defined; after determining their values, the hatching is automatically removed.

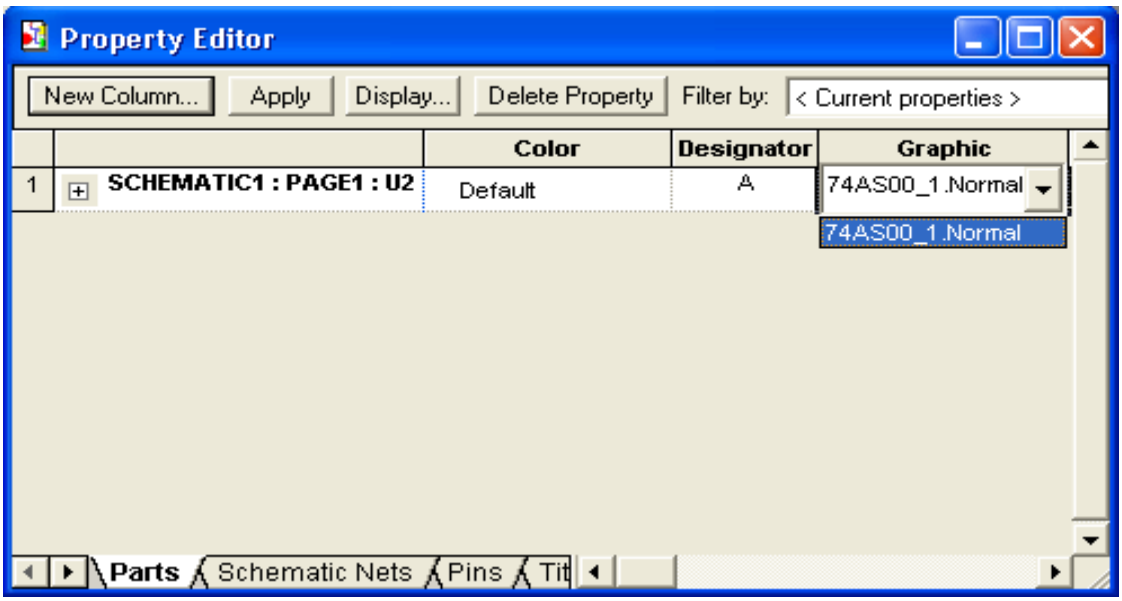


Fig. 5.18, a) Tables of harmonic signal source parameters (a) and digital IC 7412 (b)

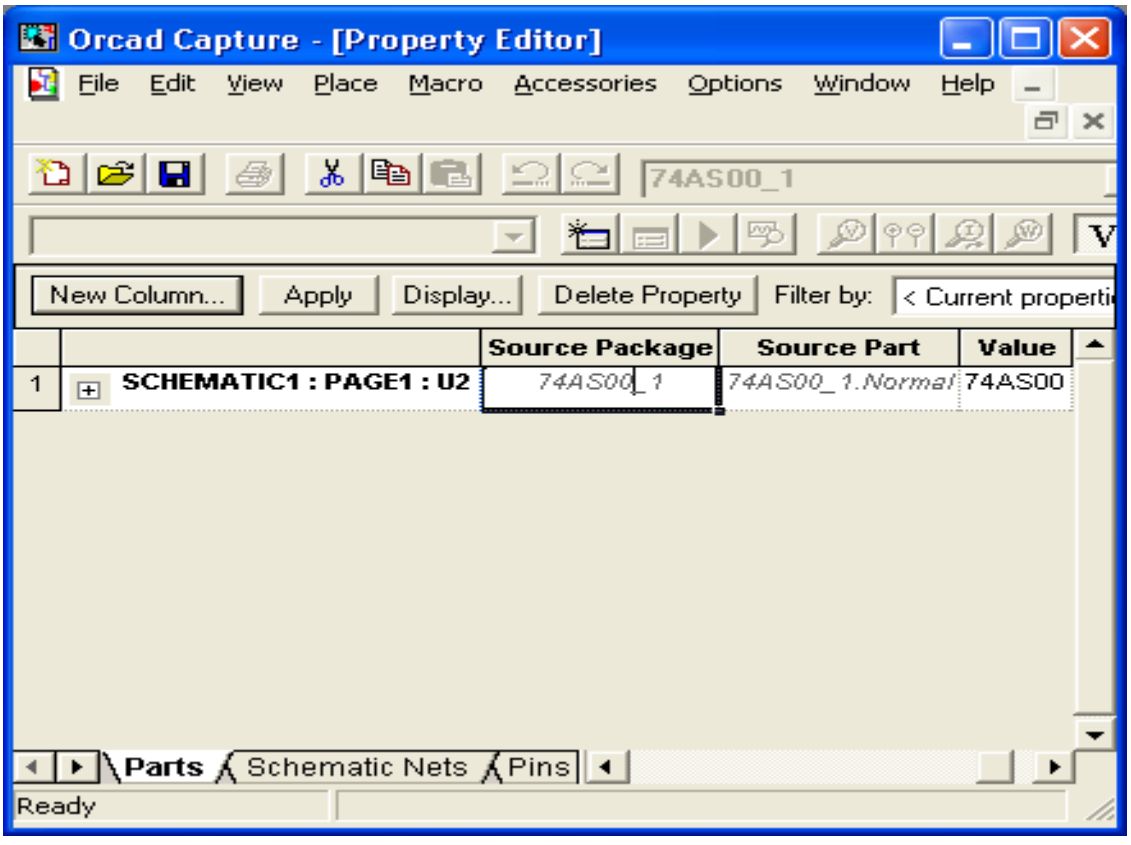


Fig. 5.18, b

5.2. Placement of components and electrical circuits symbols

Placement of component designations. Positional designations of components (Part Reference) and section numbers (Designator) are specified manually or when entering components (see fig. 5.16. a and 5.17, a), or when editing their parameters (see fig. 5.4). In automatic mode, component designations and packaging of component sections are placed on the diagram using the Tools> Annotate command of the project manager or by clicking on the button. The dialog box of this command is shown in fig. 5.19, which has the following fields:

- Scope (area task):
    - Update entire design - update designations and packaging information of the entire project;
    - Update selection - update designations and packaging information of the selected part of the project;
- Action (actions):
- Auto › Back Annotate
    - update - update the designations and packaging information of components that have a question mark "?" instead of a number, the component numbers are increased by one;
    - Unconditional reference update – updating tags and packaging information of all components in the selected area;
    - Reset part reference to "? - replacing component numbers with a question mark "? ";
    - Add - adding links to other pages;
    - Delete Intersheet Reference - delete links to other pages;
- Mode (update mode):

- Update Occurrences - updating the parameters of all individual component samples;
- Update Instances - updating the component parameters and all references to it (this mode is better when working with PSpice projects);

- Physical Packaging (automatic packaging of components according to specified properties, for example, packaging capacitors in a certain case, the capacities of which are within the specified limits):
  - Combined property string – property line;
- Reset reference numbers to begin at 1 in each page – start from 1 the numbering of designations of similar components on each page;
  - Do not change the page number.

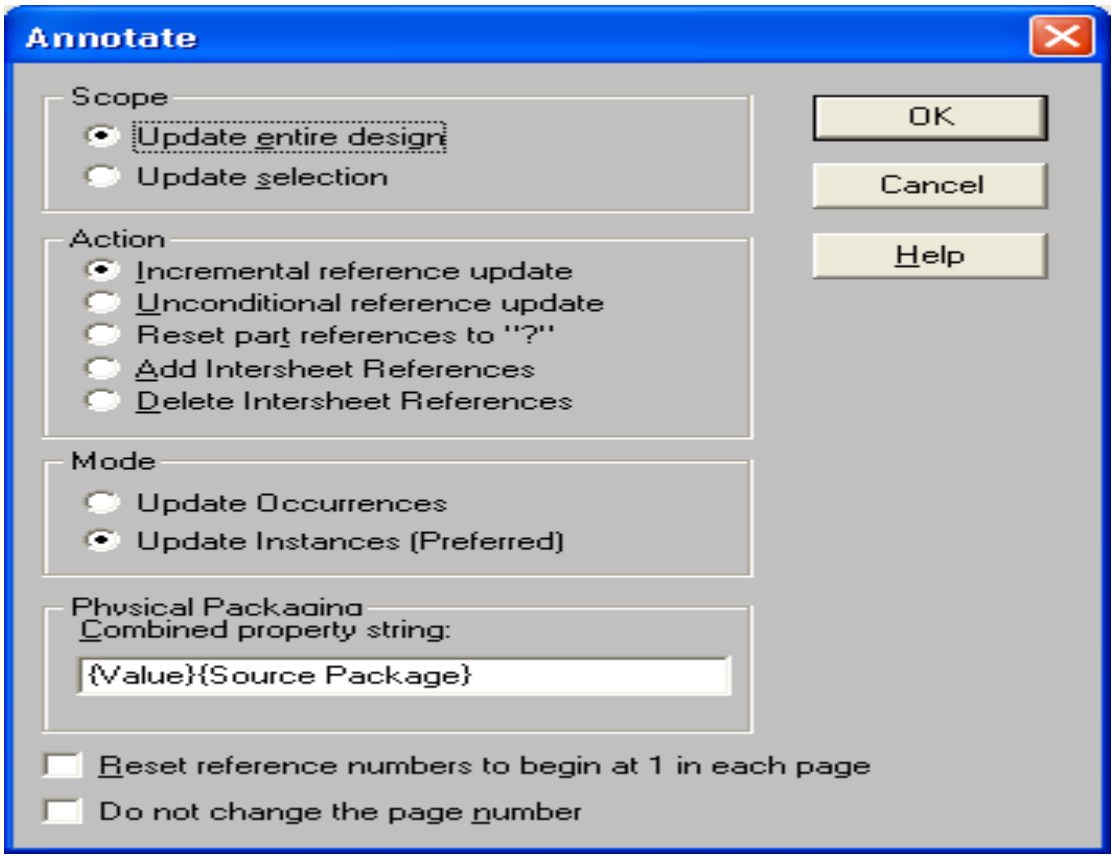


Fig. 5.19. Automatic positioning of components

Using the Annotate command, the adjacent symbols of sections of multi-section components are packed into cases (Fig. 5.20, a) and the component

designations are assigned in the left- to-right and top- to-down direction (fig. 5.20, b). In addition, component symbols can be matched with certain cases that satisfy a number of characteristics specified in the Combined property string.

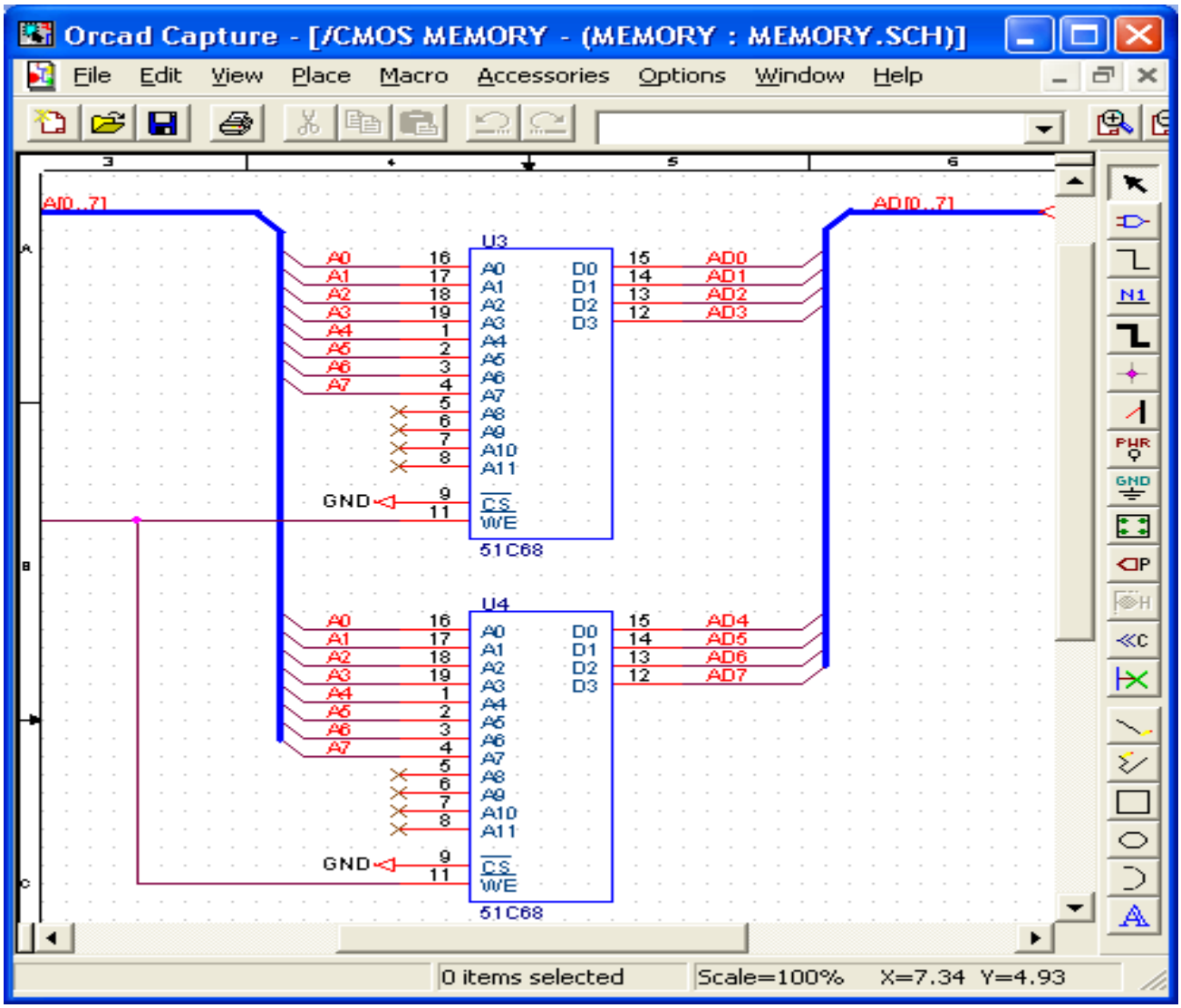


Fig. 5.20, a) Automatic packaging of components (a) and placement of designation (b)

Placement of "ground" symbols and power sources. The commands Place> Ground and Place> Power or clicking on the tool buttons open dialog boxes, an example of which is shown in Fig. 5.21, similar to the component input dialog box

(see fig. 5.16, a). The list of " ground " symbols and power sources placed in the standard CAPSYM.OLB and SOURCE.OLB libraries is given in the table. 2.3. Moreover, these symbols can be placed on the diagram only by Placeground and Place> Power commands. Both of these commands are equivalent. At the same time, power symbols have visible attributes of their names, which can be changed in the Name panel, for example, you can specify the name + 5V (by default, this name displayed on the diagram coincides with the name of the symbol). Names have no fundamental meanings, they are applied only for greater visibility of the scheme.

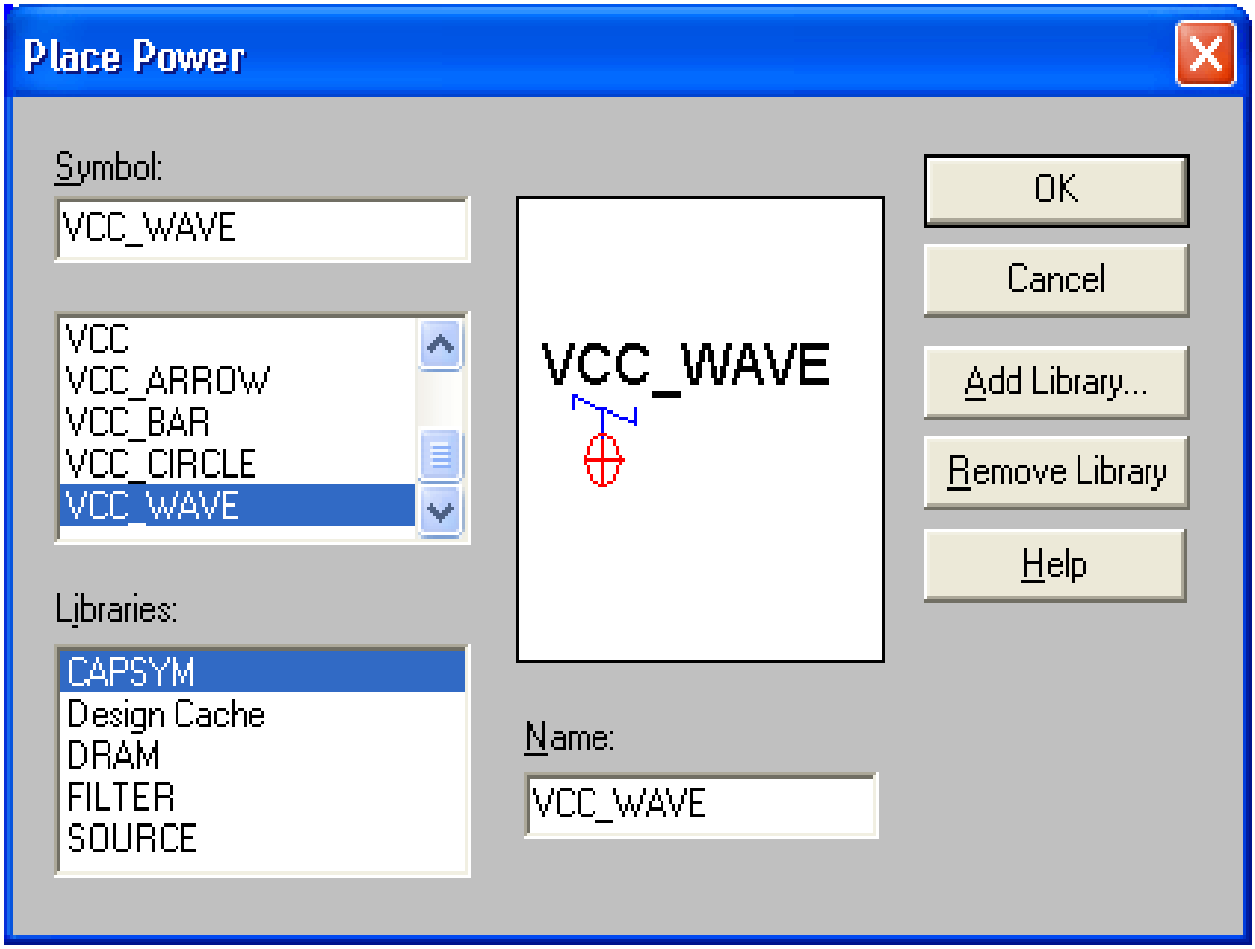


Fig. 5.21. Dialog box for entering power symbols

Table 5.3. List of "ground", power symbols and constant logic signals.

Библиотека символов	Имя символа	Назначение
CAPSYM.OLB	GND_EARTH	"Земля"
	GND_FIELD SIGNAL	"Земля"
	GND_POWER	"Земля"
	GND_SIGNAL	"Земля"
	VCC_ARROW	Источник питания
	VCC_BAR	Источник питания
	VCC_CIRCLE	Источник питания
	VCC_WAVE	Источник питания
SOURCE.OLB (для PSpice)	0	Глобальная "земля"
	$D_HI	Логическая "1"
	$D_LO	Логический "0"

The symbols "ground" and power are connected to the node with the name 0 of the circuit or to the terminals of the components to which they should be connected (to verify this, just look at the netlist files * .net files or simulation tasks * .cir). Therefore, when modeling using the PSpice program, you cannot connect power supply symbols, only the "ground" symbol is used, which has the name "O". In addition to the " ground " symbol, the SOURCE.OLB library also contains the symbols of the constant logic signals "1" and "O". To create your own " ground " and power symbols, use the Design> New Symbol command from the command manager menu.

Placement of no-connections symbols. By command Place> No connect or clicking on the toolbar button places No-connect (NC) symbols are drawn, which are shown on the schematic as "X" symbols attached to the component terminals. Outputs marked with these symbols are not included in error message reports and connection lists. NC symbols cannot be deleted by pressing the [Delete] key, to delete them, you need to place another NC symbol on top of the NC symbol.

Placement of page connector symbols. The Place> Off-Page Connector command or clicking the toolbar button opens a dialog box for drawing page connector symbols on the diagram. In the standard CAPSYM.OLB library there are two- page connector symbols L and R. The names of the page connectors, which are automatically assigned the names of the chains attached to them are entered in the Name panel of the dialog box. Circuits located on the same or different pages of the circuit and having the same names are considered electrically connected.

Placement of electrical circuits. Circle conductors are placed by the Place > Wire command, by pressing SHIFT + W, or by clicking on the toolbar button. The start of entering the chain is marked by clicking the left mouse button, then the cursor changes its shape, taking the form of a cross. The chain is laid by cursor movements. Each bend in the conductor is fixed by clicking the left mouse button. Thus, orthogonal breaks can be made in the chain at angles multiples of 90 °. Entering the conductor at an arbitrary angle is performed by holding down the SHIFT key. The input of the current circuit is completed if its end coincides with the output of the component or any point of another circuit. A double-click on the left mouse button will force the termination of the circuit, after which another conductor can be drawn. The circuit input mode is terminated by pressing the Esc key or selecting the End Wire line in the pop-up menu opened by clicking the right mouse button.

If the circuits start or end at any point on a segment of another conductor or at the output of a component, an electrical connection is established between them. A sign of connecting the chain to the output is a change in its shape - the disappearance of the square at its end. Crossed conductor segments do not merge into each other. Their connection is performed in two ways:

- when laying intersecting conductors, you need to stop at the connection point and double-click with the left mouse button - as a result, the connection will be marked with a special point (junction);
- to connect intersecting conductors, the cursor is placed at the point of intersection and the Place> Junction command is executed, the SHIFT + J key combination or the button on the toolbar is pressed; to cancel the electrical connection, you need to place another similar point on top of the connection point.

If, when placing the components on the diagram, one or more terminals collide, an electrical connection is established between them, and if these components are then moved apart, a conductor is automatically laid.

If a number of circuits are short-circuited when moving a component or a fragment of the circuit, then a warning shown in fig. 5.22 and short circuits are displayed. To cancel this move, click on the OK button and then execute the Edit> Undo Move command. Moving circuits without taking into account their electrical connections is carried out with the ALT key.

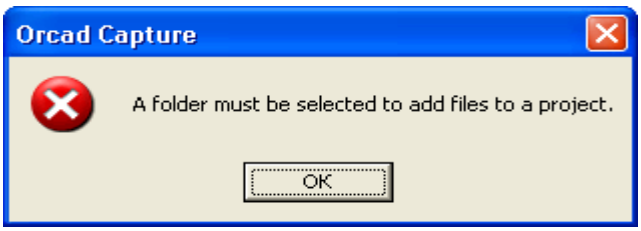


Fig. 5.22. Short circuit warning

When chains are placed, they are automatically assigned system names. such as N01049, which cannot be changed. However, the so-called aliases (Alias) of chains are entered in the connection lists, which are defined for the selected chain by the Place> Net Alias command, which is also initiated by pressing the SHIFT + N combination or by clicking on the toolbar button. Each chain can have several aliases, a current alias is selected from them in the Properties table and is used when compiling the list of connections.

The conductors are depicted by lines of a standard width of 0.2 mm at a scale of 1:1 (unfortunately, this width cannot be changed) in the diagram. Lines of the same thickness depict contour lines of component symbols and their outputs (see fig. 2.4).

Placement of group communication lines (buses). Group communication lines (buses) are entered by the command Place> Bus (SHIFT + B) or by clicking on the toolbar button. They are depicted by wider lines than conductors on the diagram (fig. 5.23). Bends of individual circuits inclined at an angle of 45° are entered by the command Place> Bus Entry (SHIFT + B) or by pressing the button according to the same rules as individual circuits. At the same time, it is convenient to copy circuit segments by dragging them while holding down the CTRL key, keeping the original object unchanged. The names (aliases) of buses and the circuits included in them are assigned using the Place> Net Alias command, and when placing the names of individual circuits, their numbers offered in the dialog box of the command are automatically increased by one, for example, ADDR1, ADDR2, ADDR3, ADDR4. The name of the bus consisting of these conductors is written in the format: ADDR [1..4]. In the diagram buses are depicted by lines of a standard width of 0.8 mm (at a scale of 1:1).

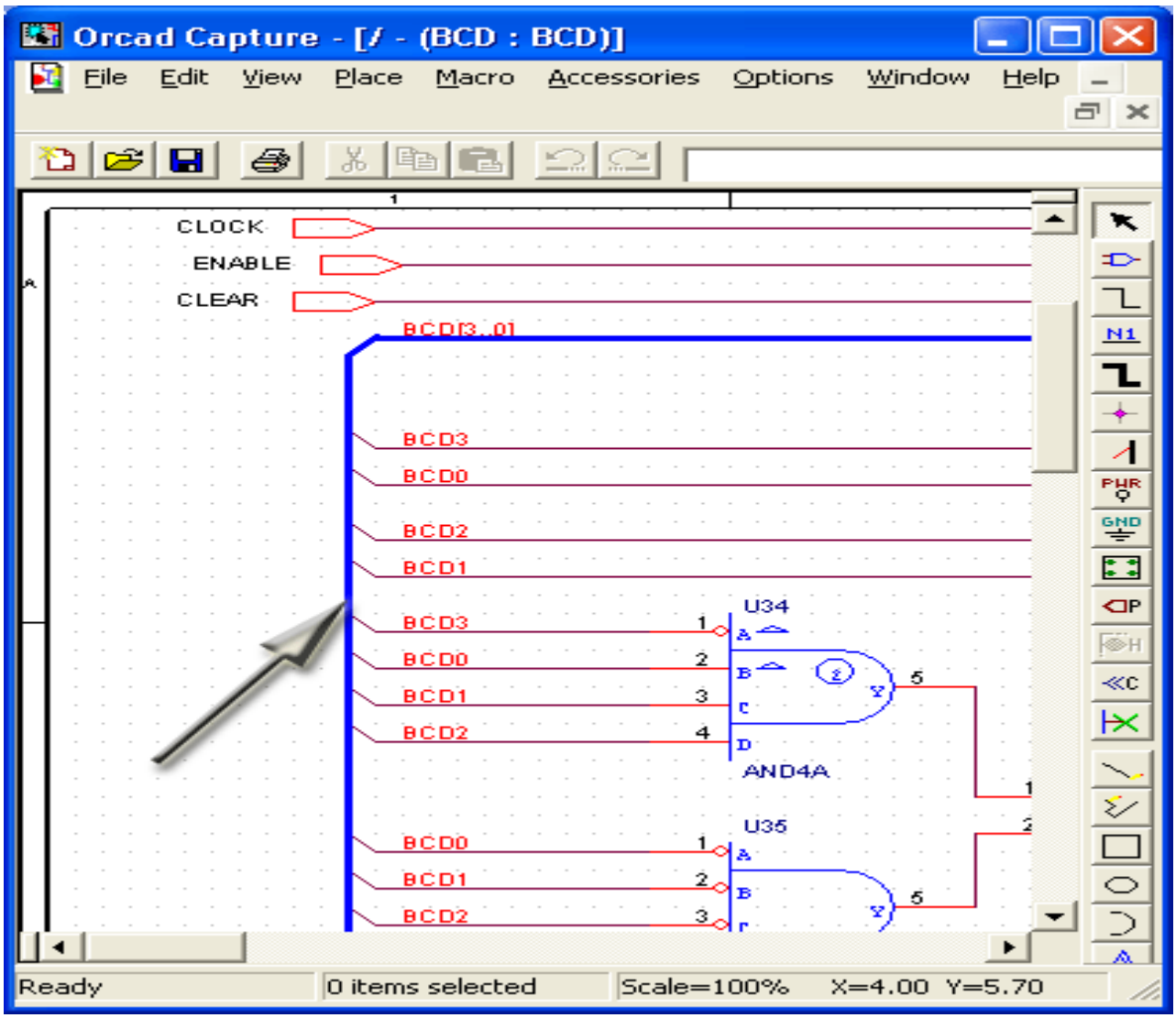


Fig. 5.23. Bus image

Any fragment of the scheme can be designed in the form of a hierarchical block, the symbol of which is a rectangle, and then placed it on the scheme, which allows you to reduce its size. Another use of hierarchical blocks is to use them to represent repeated fragments of circuits: various filters, amplifiers, rectifiers, adders, etc. A hierarchical block is placed on the diagram using the Place Hierarchical Block command or by clicking on the toolbar button. The dialog box of this command is presented. in fig. 5.24, a, and it has the following panels:

- Reference - positional designation of the hierarchical block;

- Implementation Type - a type of hierarchical block, which takes the following values:
    - Schematic View - scheme of the object,
    - VHDL - component description in VHDL language,
    - EDIF - list of connections in EDIF format,
    - Project - FPGA project,
    - PSpice Model - a mathematical model file in PSpice format; and it is necessary to manually place hierarchical outputs in this block,
    - PSpice Stimulus - a file of external influence in PSpice format; moreover, it is necessary to manually place hierarchical conclusions in this block;
- Implementation name - the name of the hierarchical block;
- Path and filename – a full name of the file that contains the description of the hierarchical block (it is not specified if the file is located in the directory of the current project, in this case, the name of the hierarchical block is accepted as the name of the folder);
- Primitive - a block type: Yes - elementary block; No - a block that has a hierarchical structure, Default - is set by default (in accordance with the configuration settings on the Hierarchy tab of the Options> Design Template command;
- User Properties - opening a dialog box for entering additional parameters of the block.

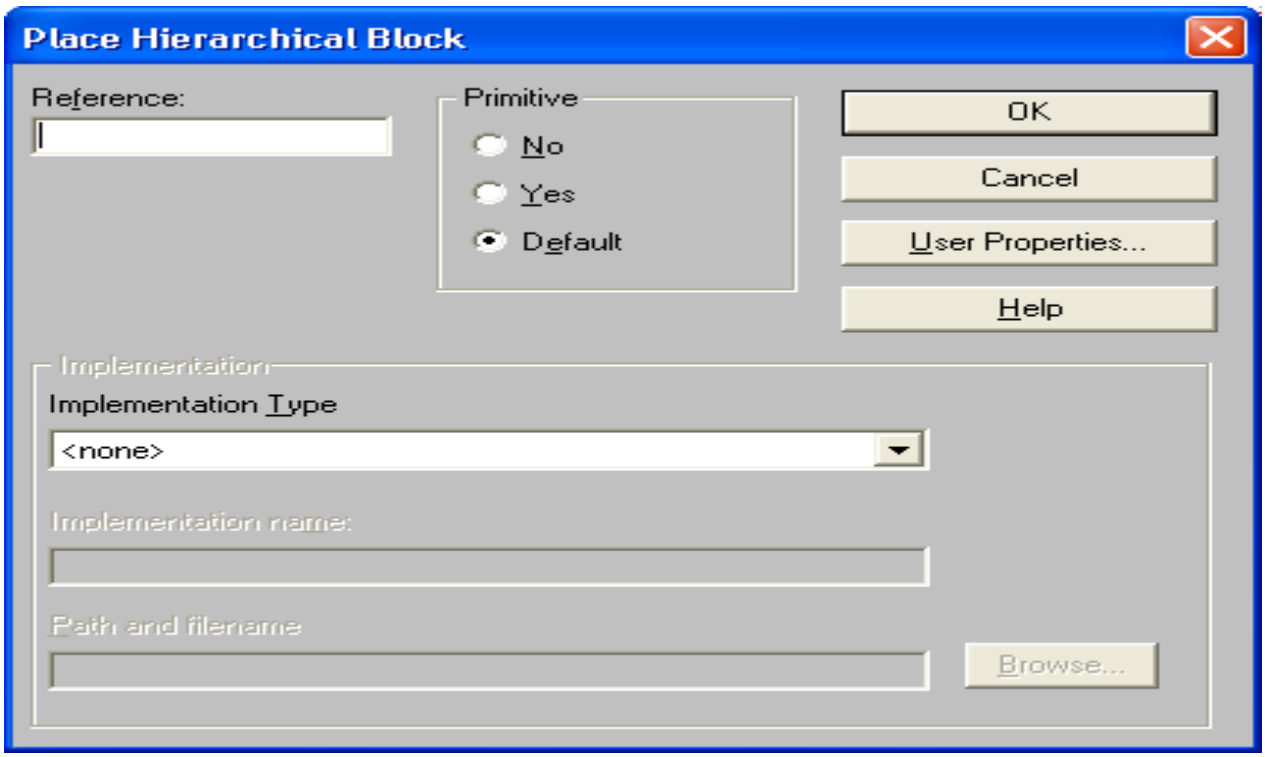


Fig. 5.24, a) Dialog windows for creating a hierarchical block (a) and drawing its outputs (b)

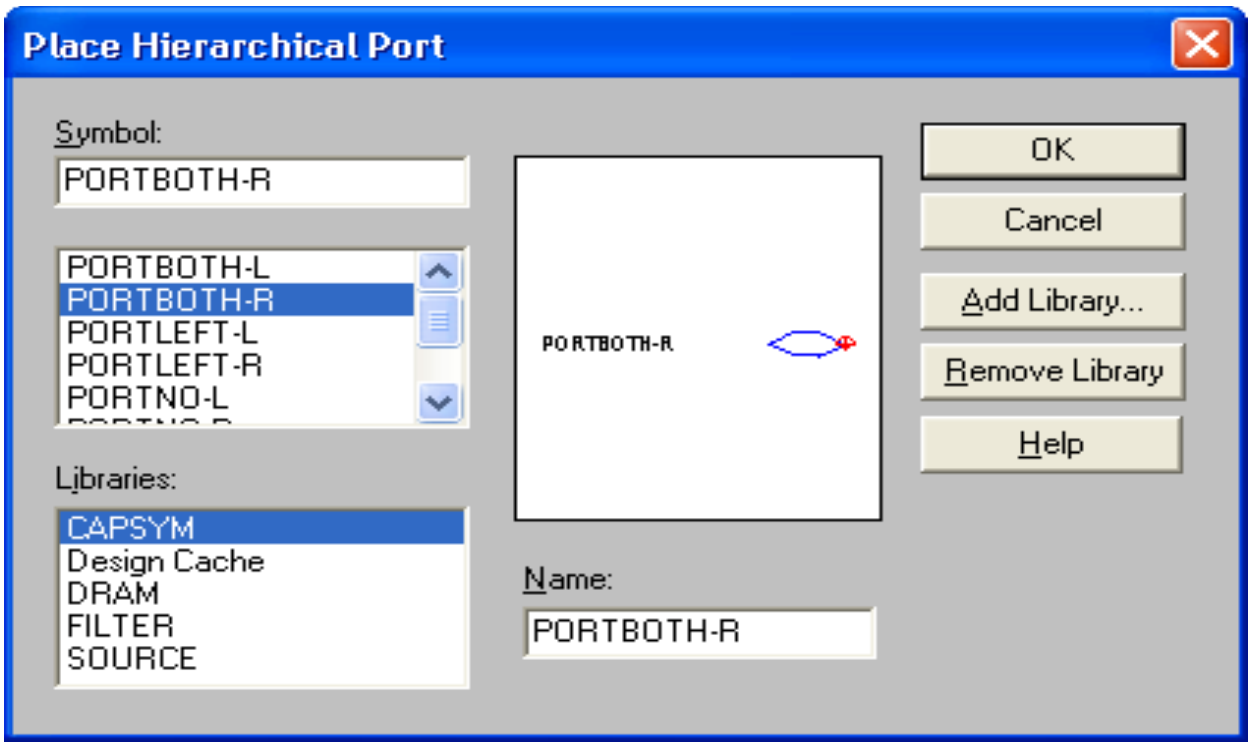


Fig. 5.24, b

After closing this window, the rectangular contours of the hierarchical block symbol are drawn with the cursor on the diagram, and the outputs of this block are entered using the Place> Hierarchical Pin command or by clicking on the toolbar button. In the dialog box (fig. 5.24, b) the following commands are specified:

- in the Name panel - the name of the output;
- in the Type column - the output type:
  - 3 State - the output of a digital component that has three states;
  - Bidirectional - bidirectional output of a digital component;
  - Input - input;
  - Open Collector - the output of a digital component of the open collector type;
  - Open Emitter - the output of a digital component of the open emitter type;
  - Output - output;
  - Passive - output of the passive component;
  - Power - output of connection to the power source;
- the type of circuit that is connected to the terminal is selected on the Width panel:
  - Scalar - a single chain;
  - Bus- a bus.

In order not to open this window every time when placing a new output, you can place all outputs of the block of the same type, and then edit the table of all outputs (fig. 5.25) by selecting the Edit Properties line in the pop-up menu.

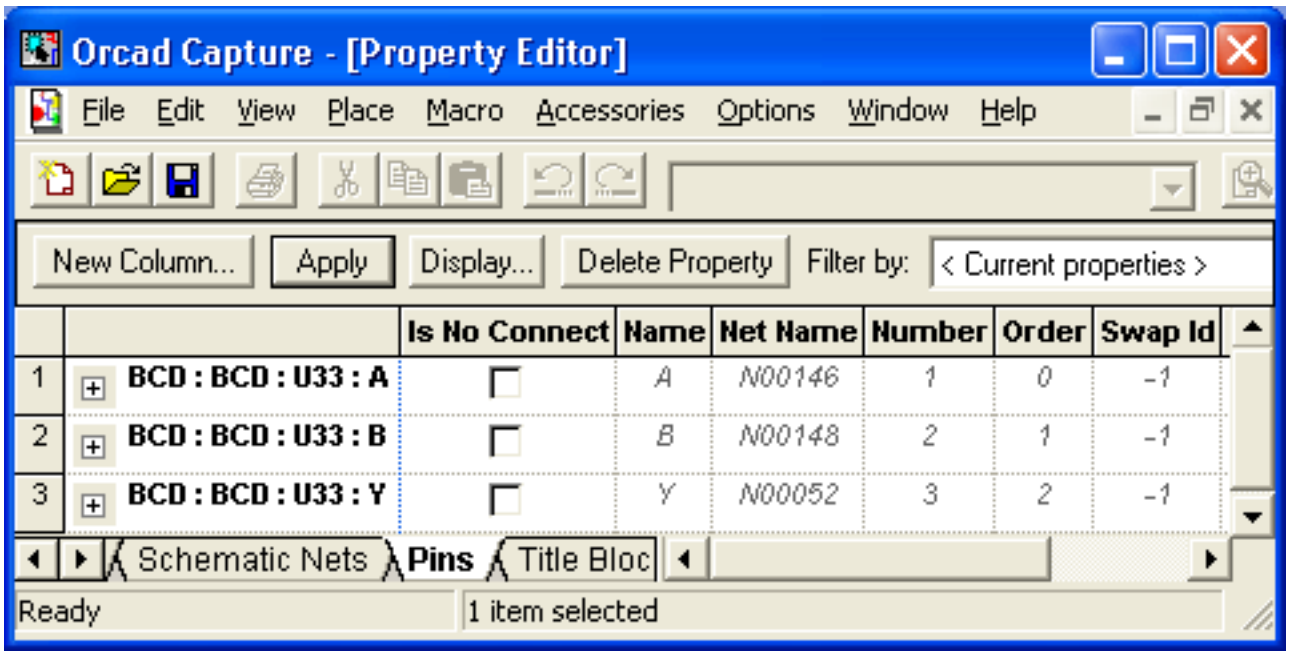


Fig. 5.25. Table of output properties

After completing the command Place> Hierarchical Block, a folder with the specified name is automatically created. The description of the hierarchical block must be placed in this folder in the form of its replacement scheme (if the Schematic View block type is selected) or a text description in the VHDL language. An example of a schematic description of a hierarchical block is shown in fig. 5. 26. Chains connect to the terminals of a hierarchical block are assigned the names that match the names of the corresponding terminals, or the external ports of the circuit of this block are entered by the command Place> Hierarchical Port or by clicking on the toolbar button (the names of the ports must also match the names of the corresponding terminals so that to ensure their electrical connection).

Graphical information is entered on the scheme using the commands Place> Line, Polyline, Rectangle, Ellipse and Arc. This information is of an auxiliary nature, so it is possible, for example, to create electrical circuits. Default graphics execution styles are set on the Miscellaneous tab in the Options> Preferences command window. After drawing segments of lines or arcs using the commands

Place> Line, Place> Polyline and Place> Arc, it is possible to edit them using the Edit Graphic dialog box (Fig. 5.27, a). It selects:

- LineStyle&Width - a type of line (solid, dashed, etc.) and its thickness (0.2, 0.8 and 2 mm);
- Color - a line color.

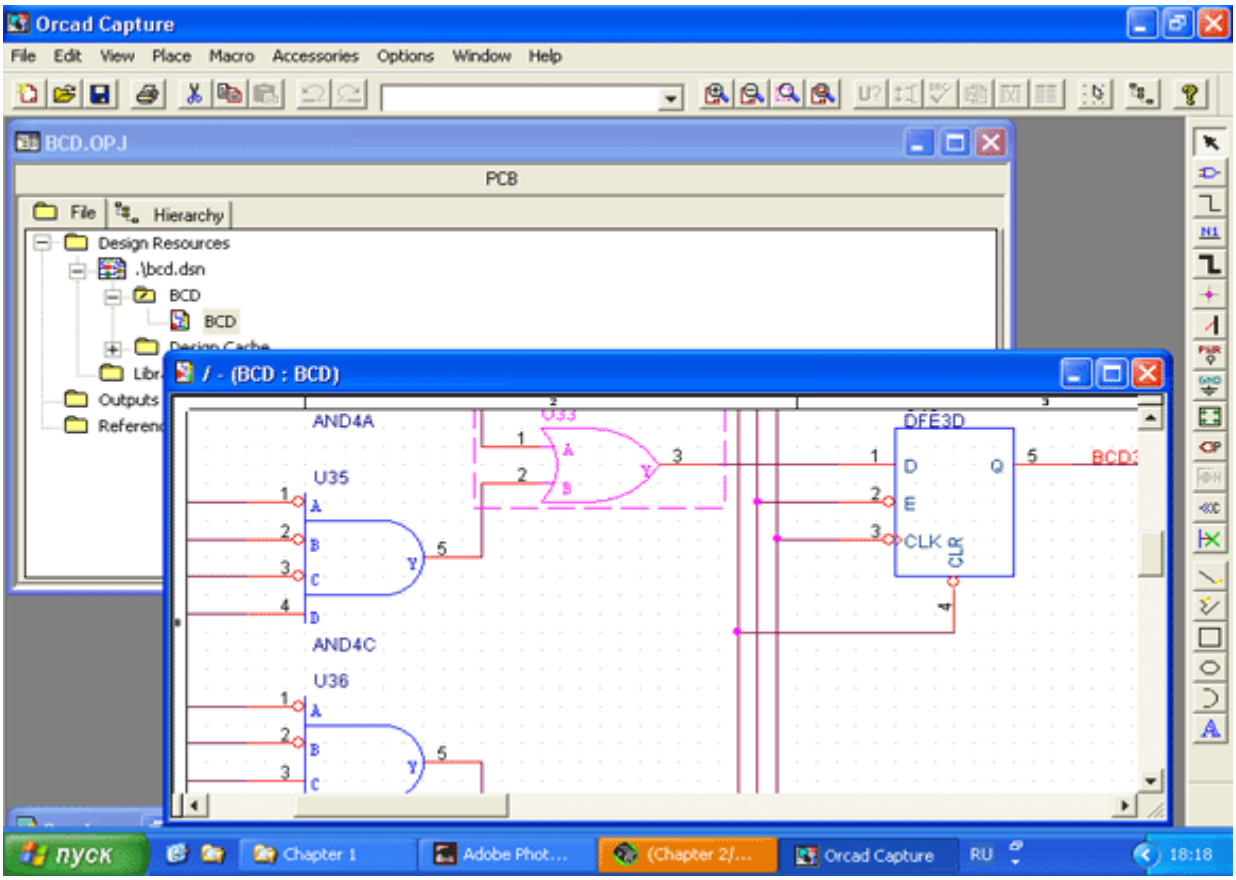


Fig. 5.26. Schematic description of hierarchical blocks

The Fill Style is selected in the Edit Filled Graphic dialog boxes that are opened when editing closed shapes, in addition to the above-mentioned parameters (see fig. 5.27, b).

OrCAD Illustrated Tutorial > Creating a Project in OrCAD Capture > Placing Graphic Objects and Text.

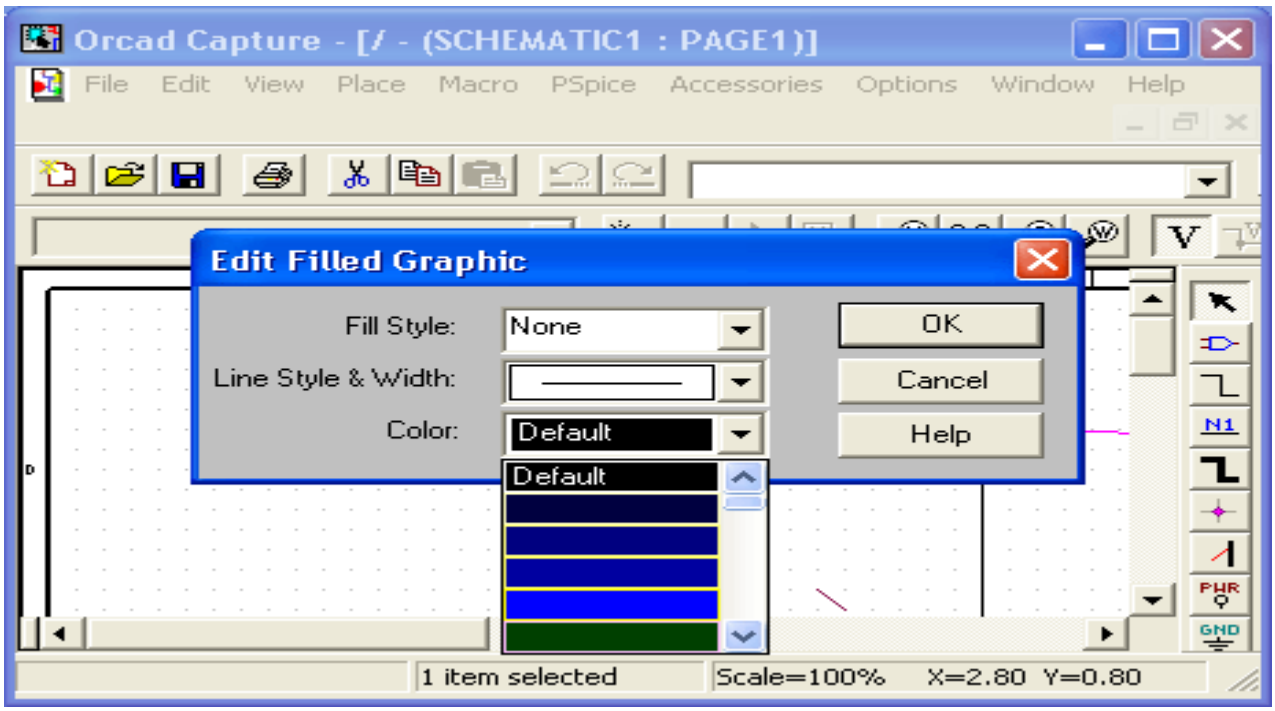


Fig. 5.27, b)

To increase the resolution when drawing graphics, you can disable the mode of cursor snap to grid nodes (Pointer snap to grid option on the Grid Display tab in the Options> Preferences command window), in this case cursor movement step is 0.1 from the grid step.

Drawings previously entered in * .bmp graphic files are applied to the diagram using the Place> Picture command.

Text is applied to the diagram using the Place> Text command or by clicking on the toolbar button. Preliminarily, the text is entered in the dialog box shown in fig. 5.28, a) (a forced transfer of text to a new line is performed by pressing the CTRL + Enter keys), which indicates the orientation of the text and the color of the font. The font type and size is selected in the window (fig. 5.28, b) that opens by clicking on the Change panel (Cyrillic fonts are available, see fig. 5.26). The font is instaledl by the Options> Design Template command.

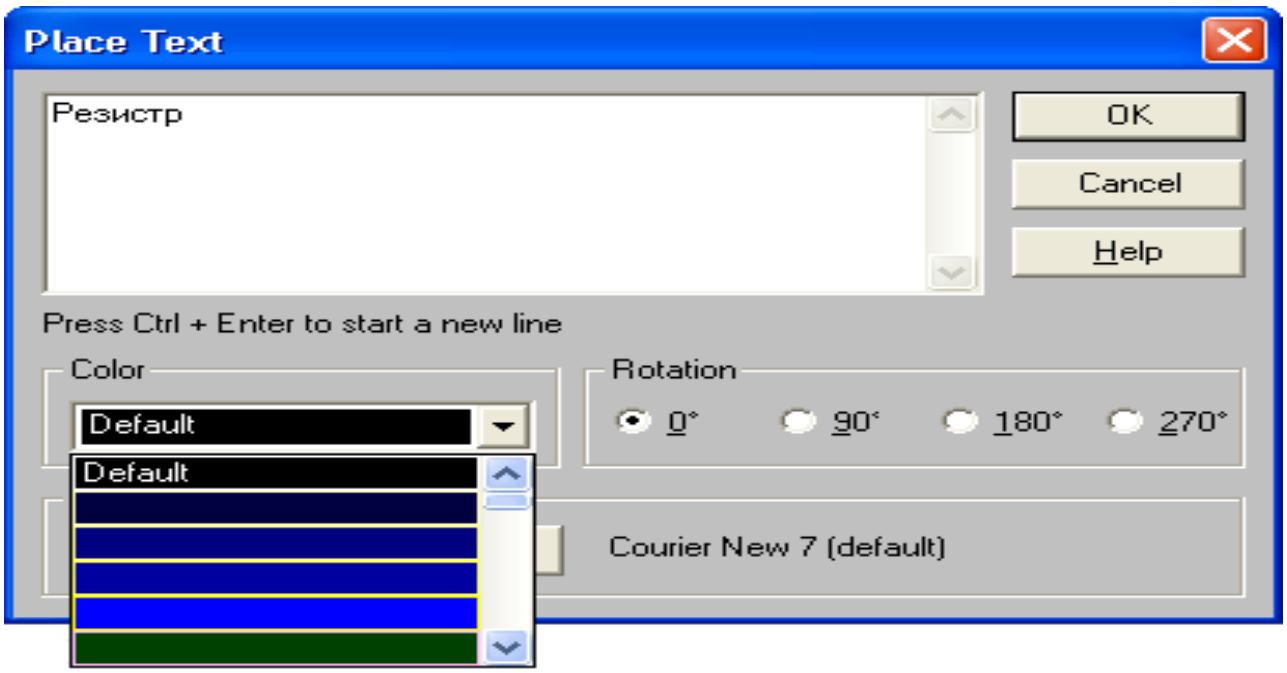


Fig. 5.28, a) Dialog boxes for text input (a) and font selection (b)

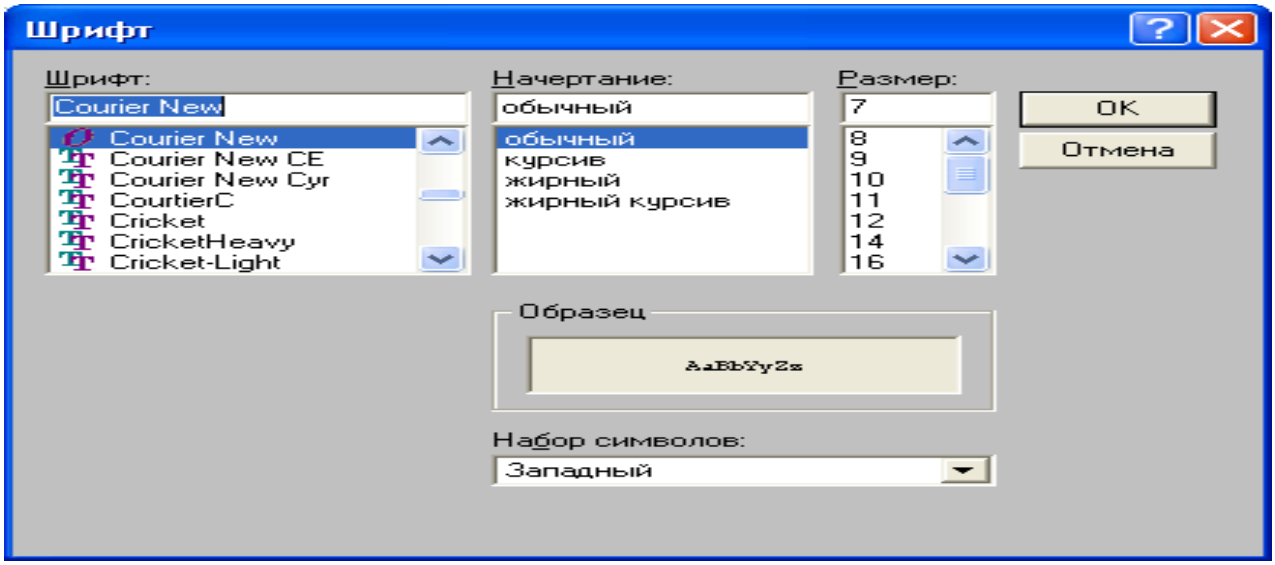


Fig. 5.28, b)

Text import into the dialog box (Fig. 5.28, a) from other Windows programs is usualyl performed by pressing the CTRL + V keys (the text fragment must first be placed in the Clipboard). To export the line of text selected on the diagram to the clipboard for transfer to other programs, press CTRL + C or CTRL + X.

The scheme editor has the ability to record the sequence of execution of individual commands in a file called a macro command file, and then re-execute it.

For example, you can write commands for laying the circuit and placing its name in such a file. The created macro command file is written to the temporary memory. Such a file can be executed only during the current Capture session. To give this file a unique name, you must specify it in the Configure Macro dialog box. Due to the fact that macro command files can be used only within one page of the scheme, the following commands cannot be written in them:

- transition to another level of the Ascend and Descend hierarchy;
- editing components Place> Edit Part.

The coordinates of the objects recorded in the macro file are calculated relative to the position of the cursor when executing the last command before writing to this file. Writing to the macro file is performed in the following sequence:

1. By clicking the left mouse button, the point on the page of the scheme is marked, in relation to which the coordinates of the macro file will be measured;
2. By Macro>Record command, which is duplicated by pressing the F7 key, a line of macro file recording tools opens, which contains three buttons and is shown in fig. 5.29;
3. By clicking the right button in the toolbar, the mode for writing commands to a macro file is activated and these commands are executed; while pressing the middle button pauses command recording. The creation of the macro file is completed by pressing the left button.

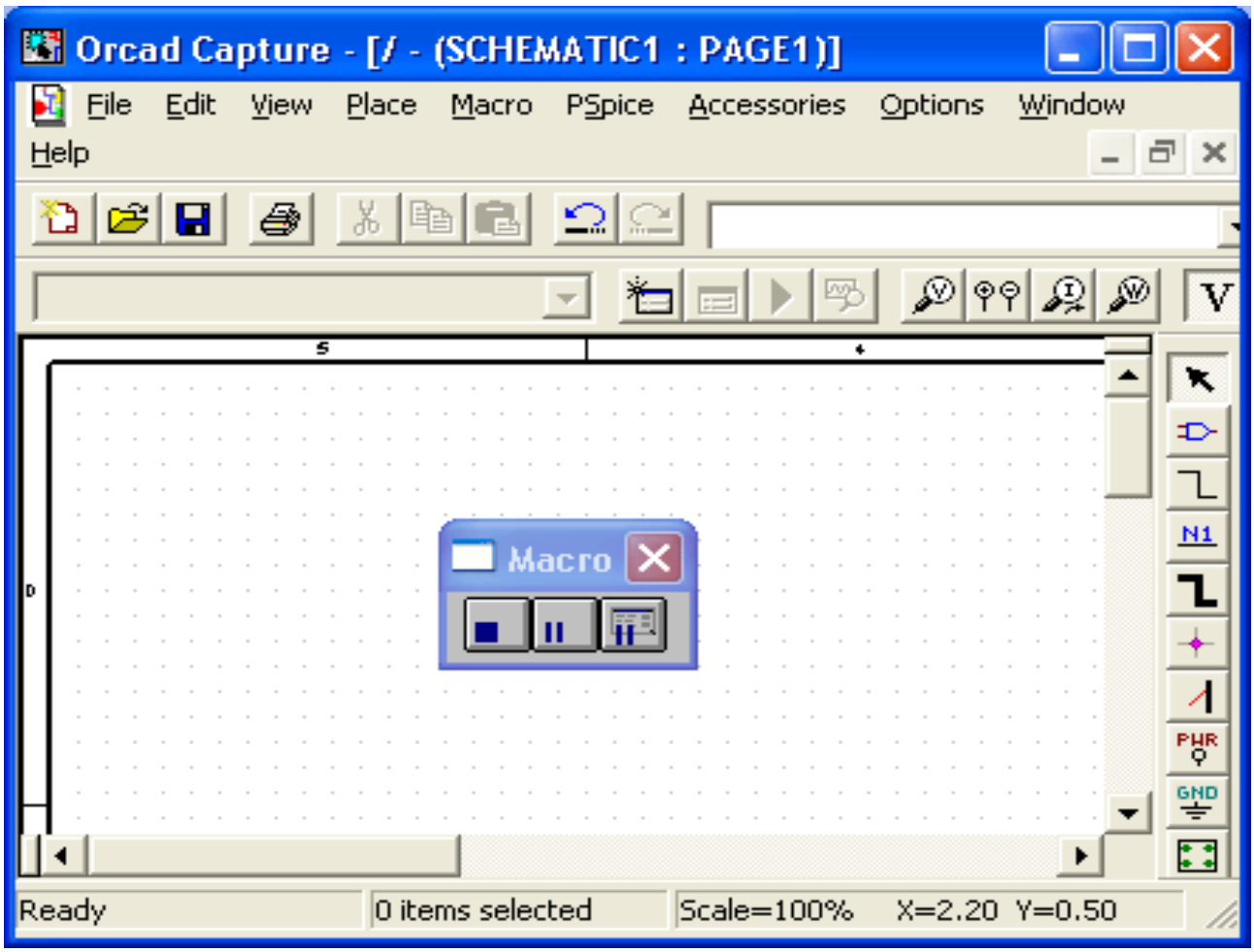


Fig. 5.29. Macro file recording control buttons (Stop, Pause, Start recording)

To execute the last macro file, by clicking the left mouse button, a point on the diagram is indicated, to which the origin of the coordinates of the macro file will be connected, and execute the Macro> Play command, duplicated by pressing the F8 key. To assign a macro file name and select a macro file for execution by the Macro> Configure command, duplicated by pressing the F9 key, the macro file configuration dialog box shown in fig. 5.30 is opened. This window contains the following panels:

- Macro Name - the name of the macro file;
- Configured Macros - image of a list of available macro files, which indicates the name of the macro file to be executed;
- Close - closing the dialog box;

- Record - closing the dialog box and starting to record commands in a macro file;
- Play - execution of a macro file;
- Add - adding another name to the list of macro files;
- Remove – removing a macro file name from the list;
- Save - saving changes to a stream macro file with the same name;
- Save As – saving changes to a stream macro file with a new name;
- Keyboard Assignment - assignment "hot" keys to execute a macro file, for example, Ml, M2 or CTRL + 1;
- Menu Assignment - menu specification associated with the stream macro file;
- Description - description of the macro file.

Changing the view of the current page of the scheme is carried out by changing the image scale using the View>Zoom commands, panning (changing the center point of the image without changing the scale) using the View>Zoom>Selection command or moving to the specified point using the View>Go Te command. Zoom commands do not require much explanation. Let's consider the Go To transition commands in more detail, which have three dialog boxes shown in fig. 5.31.

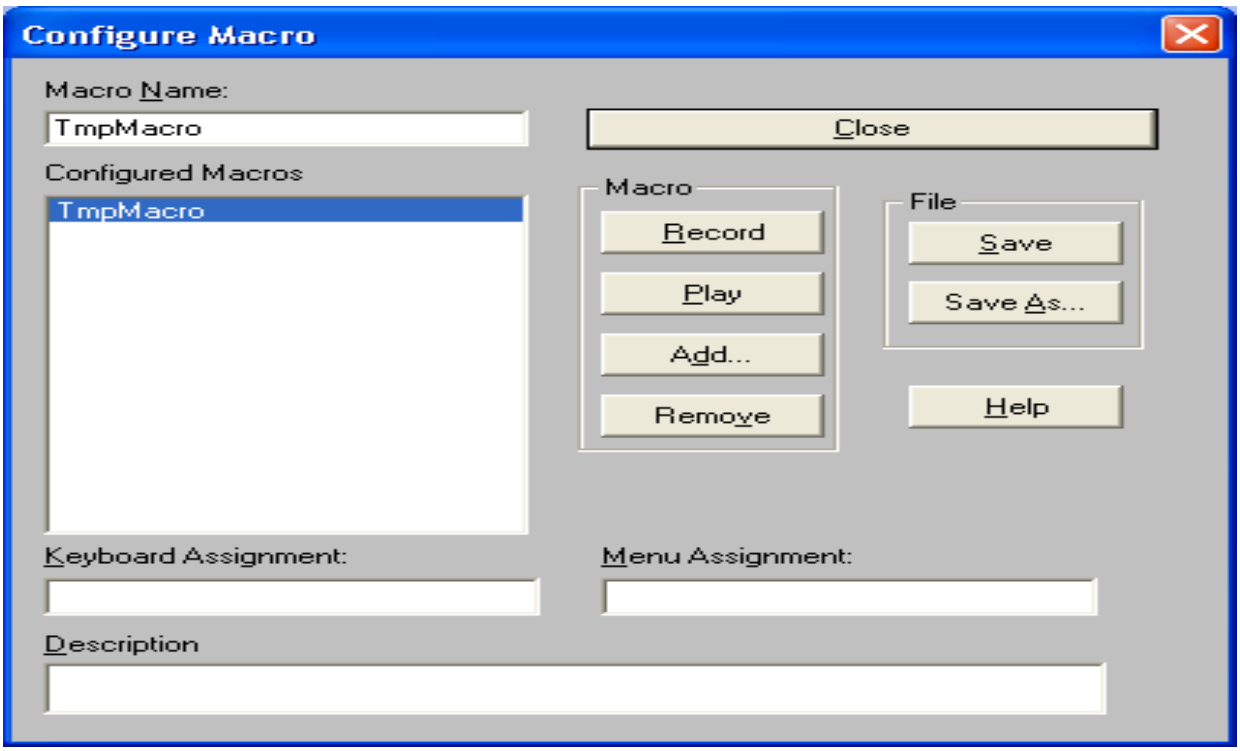


Fig. 5.30. Macrofile configuration dialog box

Note

Examples of useful macro files are located in the directory \ CAPTURE \ MACROS.

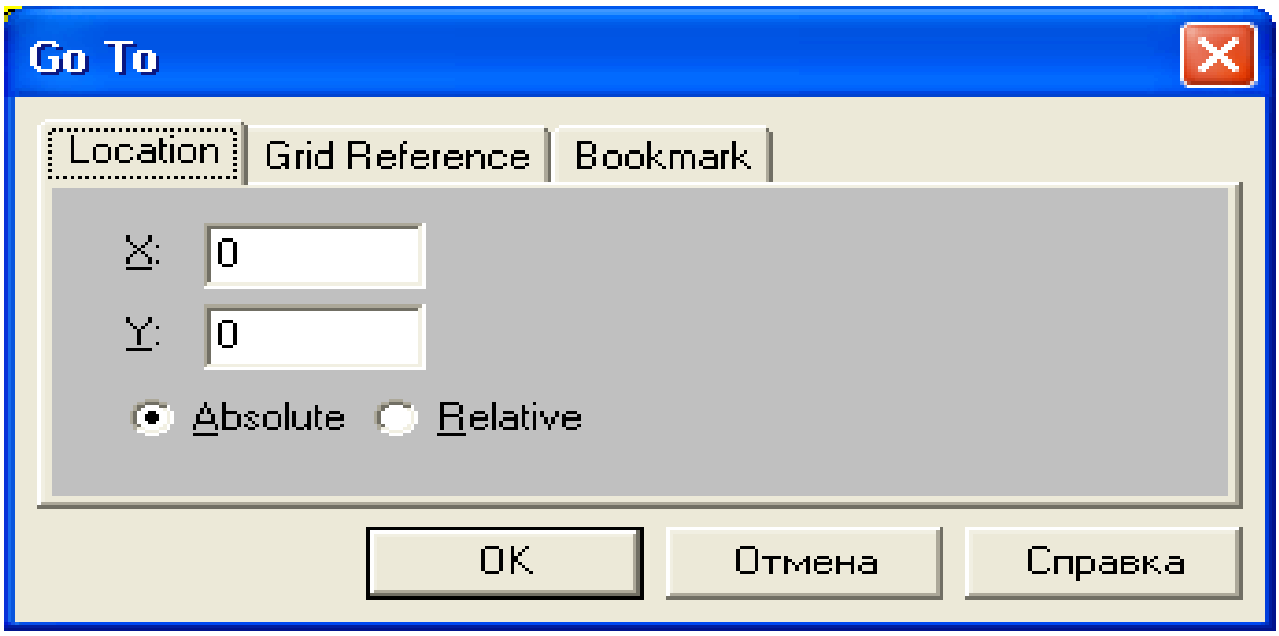


Fig. 5.31. Go To command dialog boxes

Moving to the point with the specified X, Y coordinates is performed using the Location dialog box (fig. 5.31). The transition to the point which coordinates are measuredd on the drawing frame is performed using the Grid Reference dialog box. Finally, moving to the point specified in advance by the Place> Bookmark command is performed using the Bookmark dialog box.

In addition, the Edit> Find command (fig. 5.32) is used to search for various objects on the diagram.

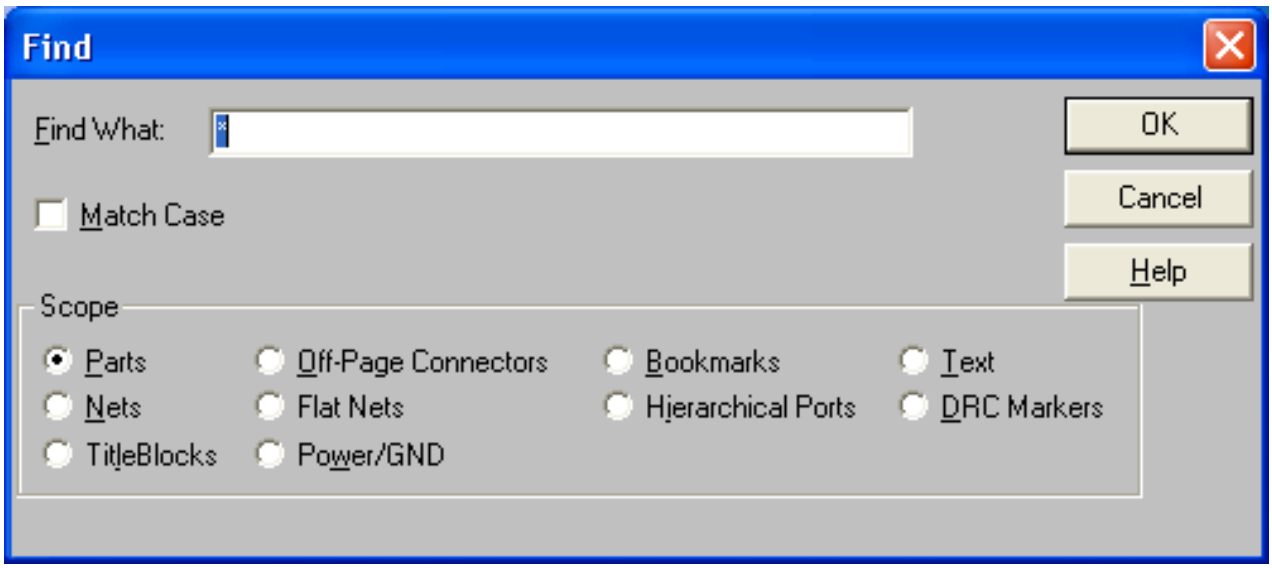


Fig. 5.32. The object search dialog box on the diagram

The symbol libraries (*.olb files) of the Capture program of the OrCAD 9.2 system contain more than 30,000 elements. When creating a project, it is necessary to think in advance which libraries can be used in each specific case. Otherwise, for example, after creating a schematic diagram of the device, it is not possible to develop a printed circuit board due to inconsistencies between the symbol libraries and component housings. The brief information about placement included in the standard delivery of libraries is given in the table 5.4.

The Capture \ Library \ PSpice directory contains libraries of symbols * .olb and mathematical models of components * .lib, used in modeling by the PSpice program, and almost all symbols of the graphic editor PSpice Schematics and their corresponding mathematical models are included here.

A number of symbol libraries from the directory \ Capture \ Library \ PSpice do not contain information about the packaging of components, links on their body and numerical values of the parameters of mathematical models (these values are entered directly on the diagram):

- Abm.olb - functional blocks (adder, multiplier, linear inertial link, integrator, differentiator, limiter, etc.);
- Analog.olb - discrete analog components (R, R_var, C, L, E, etc.);
- Breakout.olb - blanks symbols of semiconductor devices and other components;
- Source.olb - sources of analog and digital signals, the parameters of which are specified in a text form;
- Sourcestm.olb - sources of analog and digital signals created using the Stimulus Editor program;
- Special.olb - symbols for assigning special modeling directives (these include the specification of PARAM parameters, the WATCH label, etc.).

Other libraries correspond to components of certain types, they are coordinated with the libraries of mathematical models and component bodies (these libraries are located in the \ Capture \ Library \ PSpice and \ Capture \ Library subdirectories):

- Anlg_dev.olb - operational amplifiers and other ICs of AnalogDevices
- Bipolar.olb - bipolar transistors;
- CD400.olb - digital to MOSFET valves;
- Lin_tech.olb - operational amplifiers of Linear Technology;
- Siemens.olb - semiconductor devices of the Siemens company

- 7400.olb, 74ac.olb, etc. - digital TTL-IS;

Table 5.4. OrCAD standard libraries.

Этап проектирования	Расширения имен файлов библиотек	Имя подкаталога расположения библиотек
Создание схем (**OrCAD Capture**)	**olb** – символы компонентов	\Capture\Library\PSpice
Создание схем (**PSpice Schematics**)	**sib** – символы компонентов **plb** – упаковочная информация	\PSpice\Library
Моделирование схем (**OrCAD PSpice**)	**lib** – математические модели компонентов	\Capture\Library\PSpice
Разработка печатных плат (**OrCAD Layout**)	**lib** – типовые корпуса (**Footprints**) компонентов	\Layout\Library (см. каталог библиотек, в файлах Liblist.txt, Layllb.txt)

5.3. Concept of symbols, components and their libraries

Component symbol libraries are files with the.olb extension that contain all the information needed to create schematics and transfer data to other OrCAD programs. Before proceeding to the description of the rules for working with libraries, we will explain the main terms adopted in OrCAD.

Physically existing transistors, capacitors, integrated circuits (ICs), etc. They are called components. Part is a conditional graphic image (symbol) of a component on the schematic diagram. Some components are multi-sectioned, consisting of several sections. If all sections of such a component are identical, for example, a digital IC 4NI-I, it is called homogeneous, otherwise heterogeneous. Information about the packaging of a component, which includes the number of sections of the component, the number of pins of individual sections, the presence of logical equivalent sections and their pins (they can be rearranged during auto-routing of PP connections) is called Package. It is accepted that the term Part denotes both the symbol of a separate section of the component and the symbol of

the entire component as a whole in the OrCAD Capture. Component symbol libraries are separate files with the olb extension.

The graphic projection of the physical body of a component on a printed circuit board is called a footprint. Footprint libraries of component bodies are separate files with the extension lib.

Symbol library files are opened in the project manager by the command File> Open> Library. After clicking the "+" icon on the line with the name of the library, its catalog is displayed as shown in fig. 5.33. By selecting individual components in this directory, they can be deleted and moved to other libraries in the usual way; to move a component from one library to another, it is necessary to simultaneously open the catalogs of two libraries in the project manager and drag the symbol icon from one library to the other, placing it on the line with its name.

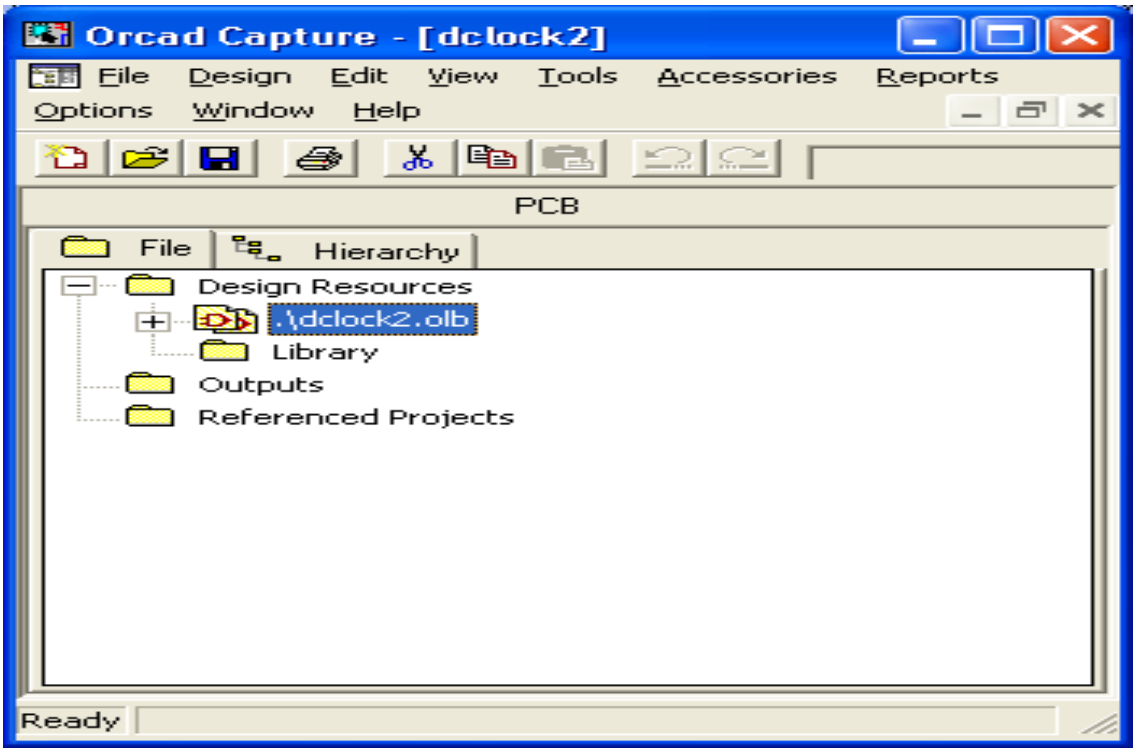


Fig. 5.33. Library catalog

When the first symbol is placed on the diagram, a so-called design cache is created, into which this symbol is copied from the library file. As a result, in the Design cache section of the project manager, the symbols of all components placed on the project diagram are added, keeping their connection with the symbol libraries. This allows you to perform a synchronous change of all instances of any symbol located in the project by changing it in the library. To do this, select the symbol of the component in the Design cache section and execute the Design> Replace Cache command. The name of the selected symbol is displayed in the Part Name line of the dialogue window of this command, shown in fig. 5.34. After that, the Part Library line indicates the name of the library (using the Brows viewer) in which it is located. After clicking the OK button, all instances of this symbol in the current project will be replaced with the library symbol. The Design> Replace Cache command updates the selected symbol, and all the parameters entered by the user are saved.

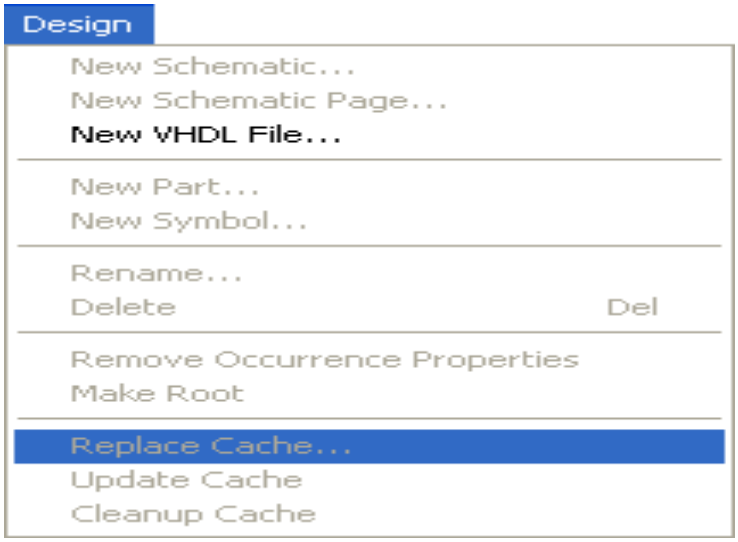


Fig. 5.34. Replacing the project cache

It is possible to create component symbols and then place them in the existing or new libraries in OrCAD Capture. To create or edit symbols, the Part Editor is used which can be accessed in one of three ways.

1. To create a new symbol, a new library is created or an existing library is opened and then the Design>New>Part command is selected.
2. To edit an existing symbol, the symbol library is opened in the project manager (fig. 5.33) and then the desired symbol is selected by double-clicking the cursor.
3. To edit a symbol placed on the diagram, it is selected with a single click of the cursor and then the Edit> Part command is executed.

Creating a new symbol. Symbols are created by two different commands depending on their purpose.

The Design > NewSymbol command (fig. 5.35) creates auxiliary symbols of four types:

- Power - symbol of connection of "ground" and "power" chains;
- Off-PageConnector - the symbol of the circuit page connector;
- Hierarchica lPort - a symbol of a hierarchical block;
- Title Block - the symbol of the title Block ("corner stamp"), its example, made according to ESKD, is shown in fig. 5.36.

The name of the symbol is indicated in the column Name (fig. 5.35), and its type is selected in the column Symbol Type. Symbols of these types are placed on schematic diagrams and do not correspond to physical existing components. The types of these auxiliary symbols are taken into account only when executing the Place> Power, Place> Ground, Place> Off-Page Connector, Place> Hierarchical Port, Place> Title Block commands - only the list of components of the corresponding type is placed in the selected library directory in the dialog boxes of the commands.

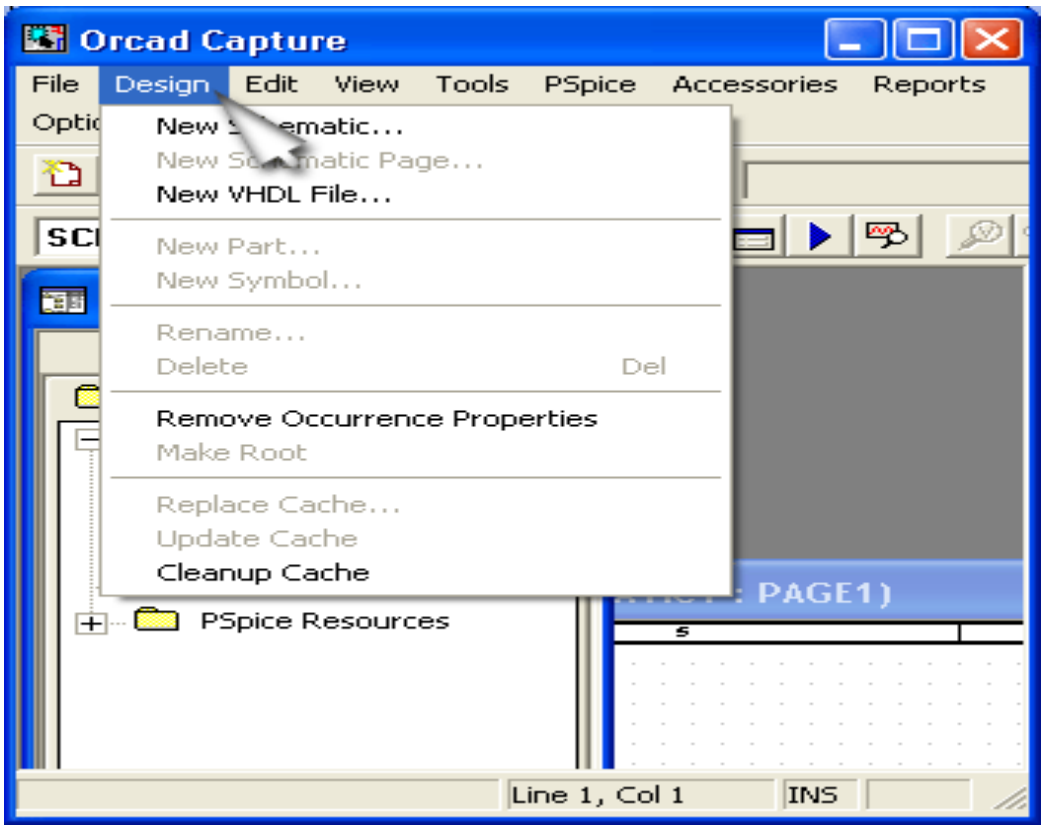


Fig. 5. 35. Design> New Symbol and Design> New Part dialog boxes

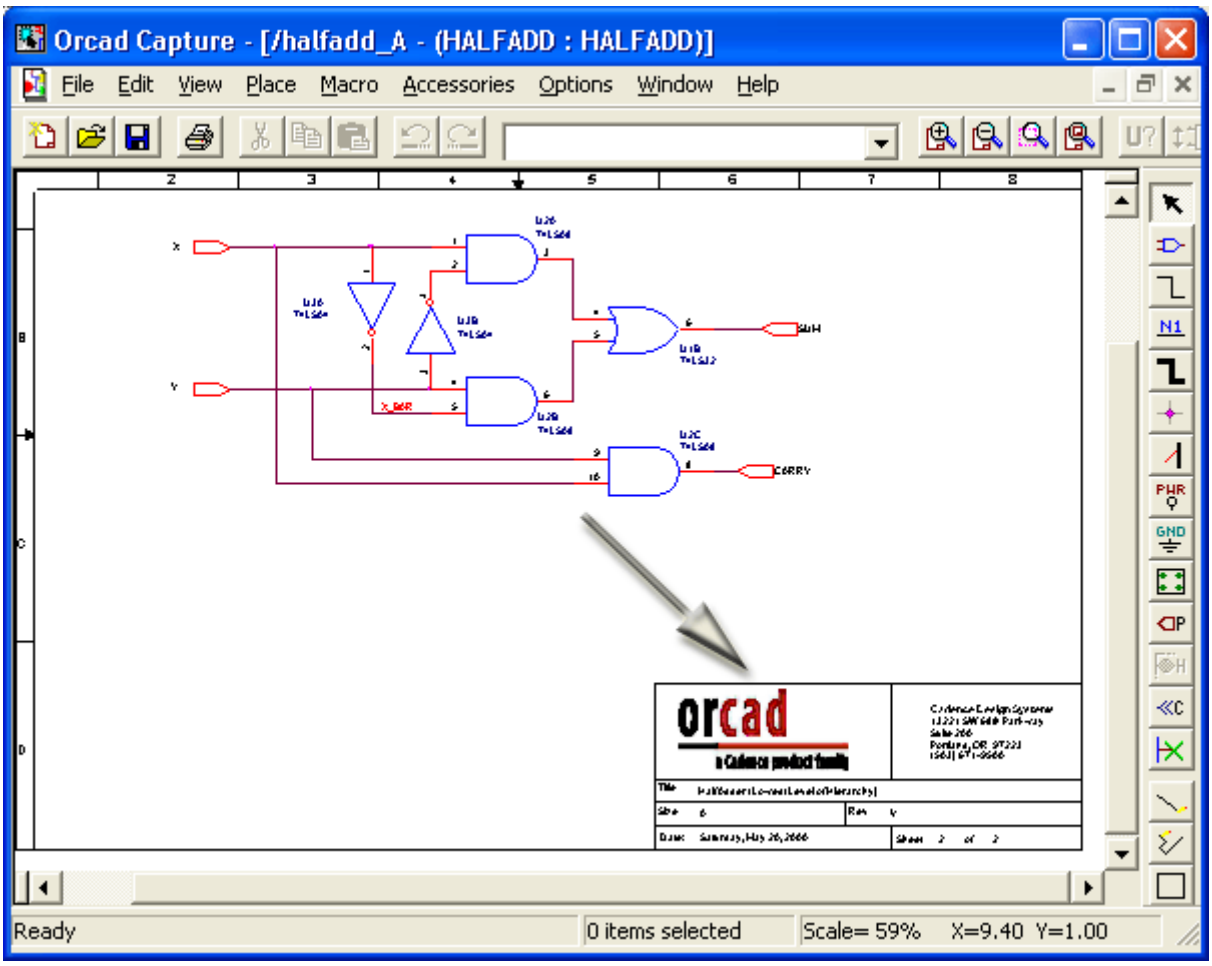


Fig. 5.36. An example of the symbol of the main inscription. (Title Block)

Symbols for all other components, most of which have physical existing bodies, are entered by the Design> New Part command. The following data is entered in the dialog window of this command:

- Name - the name of the symbol;
- Part Reference Prefix - positional designation prefix (for example, R for resistor, C for capacitor, DA for analog IC, DD for digital IC, etc.);
- PCB Footprint - the name of the typical component body, for example, DIP16, SOI24, if it exists (this parameter is required only when transferring the scheme for the development of a printed circuit board, it is not needed when performing simulation);
- Create Convert View - the need to create a second symbol image (for example, a DeMorgan equivalent for digital logic elements);
- Parts per Package - the total number of sections in the component body;
- Homogeneouse or Heterogeneouse - a choice between components with sections of the same or different type (for example, IC 133LAZ, containing 4 logical elements 2AND-NOT belongs to the Homogeneouse class, and IC 564LP2, containing 2 logical elements 3OR -NOT and an element NOT belongs to the Heterogeneouse class);
- Alphabetic or Numeric - a choice between section designations of multi-section components with letters of the Latin alphabet, for example DD1A, DD1B, DD1C, etc. (Letters of the Latin alphabet can be used to designate sections of components containing up to 26 sections in one case) or numbers, for example DD1-1, DD1-2, DD1-3;
- Part Aliases - definition of symbol aliases to reduce volume. libraries (for example, you can create an LA3 component and assign it aliases 133LA3, K155LA3, 530LA3);

- Attach Implementation - connection of an additional symbol description using an equivalent circuit, VHDL file, connection list, other project or in the form of a PSpice model;
- Pin Numbers Visible - display of pin numbers on the circuit.

After clicking on the OK panel of the dialog boxes of the Design> New Part or Design> New Symbol commands, the Part Editor workspace is opened (fig. 5.37), on which the dimensions of the symbol are limited by a dashed-dotted rectangle (the dimensions of this rectangle are changed in the usual way by "dragging" its corners).

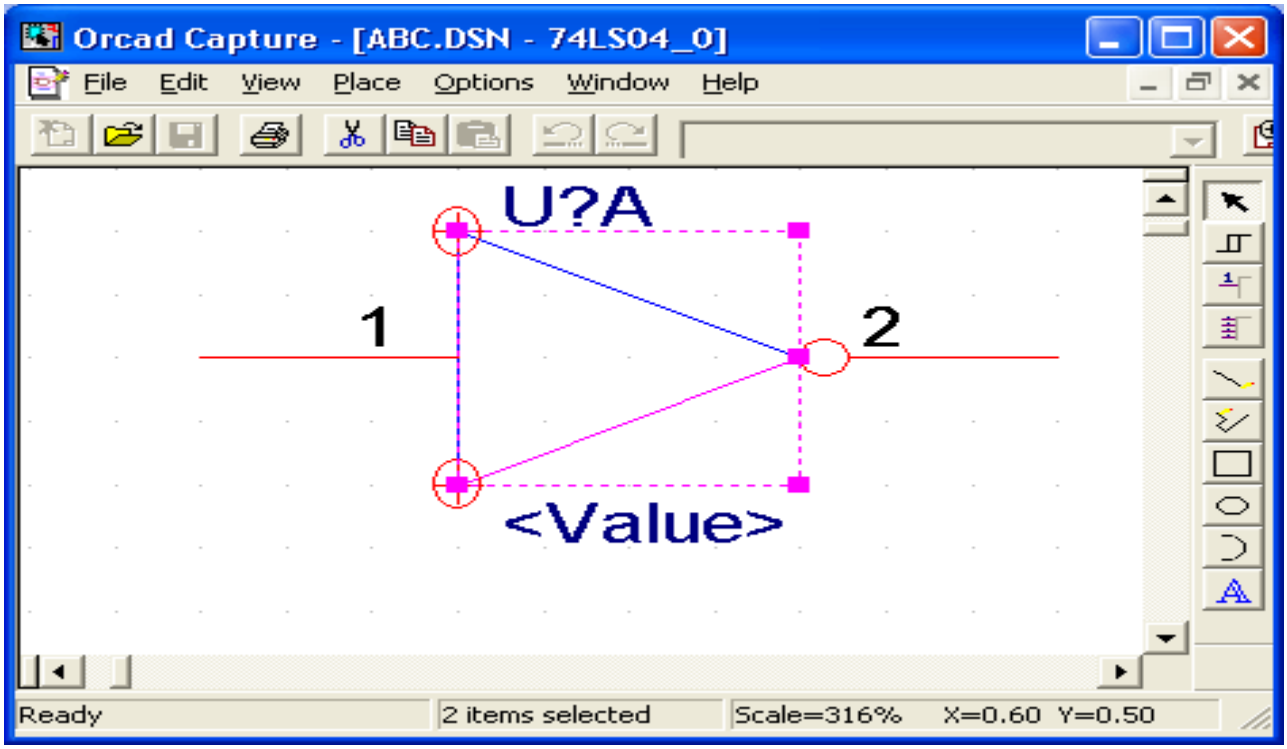


Fig. 5.37. Symbol preparation

The pins of the component must be placed outside this rectangle, touching it. The component pins are placed using the Place > Pin command, the dialog box of which is shown in fig. 5.38, a. It shows the following information:

- Name - the name of the output;
- Number - output number;

- Shape - output form (see Table 5.5);
- Type - type of output (see Table 5.6), which is used only when checking the correctness of the circuit assembly using the Tools> Design Rules Check (DRC) command;
- Scalar or Bus - a choice between a single output or a bus;
- Pin Visible - displaying the output on the diagram (only for Power type outputs), in the Edit Part window such outputs are displayed without indicating their names and numbers;
- User Properties - opening a dialog box for viewing and editing the output characteristics of the component before placing it on the workspace

The <Value> attribute is automatically placed below the contour of the component (its location can be changed by placing it inside the contour), if its value is not defined, then the name of the component is automatically indicated on the diagram as its value.

Table 5.5 Graph of pins.

Форма (Shape)	Описание
Clock	Вход синхронизации
Dot	Признак логического отрицания
Dot-Clock	Вход синхронизации с инвертированием
Line	Стандартный вывод, длина которого равна трем шагам сетки
Short	Короткий вывод, длина которого равна одному шагу сетки
ZeroLength	Стандартный вывод нулевой длины

Table 5.6. Types of pins.

Тип вывода	Описание
3-State	Тристабильный вывод, имеющий три возможных состояния: логическое состояние низкого уровня, догическое состояние высокого уровня и состояние большого выходного сопротивления (Z-состояние, это состояние эквивалентно разрыву цепи). Например, 8-разрядный регистр-защелка 74LS373 (КР1533ИР22) имеет тристабильные выводы
Bidirectional	Двунаправленный вывод (может быть как входом, так и выходом компонента)
Input	Вывод подачи входного сигнала
Open Collector	Выход вентиля с открытым коллектором (к нему подключается резистор нагрузки)
Open Emitter	Выход вентиля с открытым эмиттером (к нему подключается резистор нагрузки)
Output	Выход компонента
Passive	Вывод пассивного компонента (резистора, конденсатора, диода и т.п.)
Power	Выводы для подключения цепей "земли" и "питания". Например, для ИС серии 133 питание подключаются к выводу 14, а "земля" – к выводу 7. Имена этих выводов должны совпадать с именами соответствующих цепей

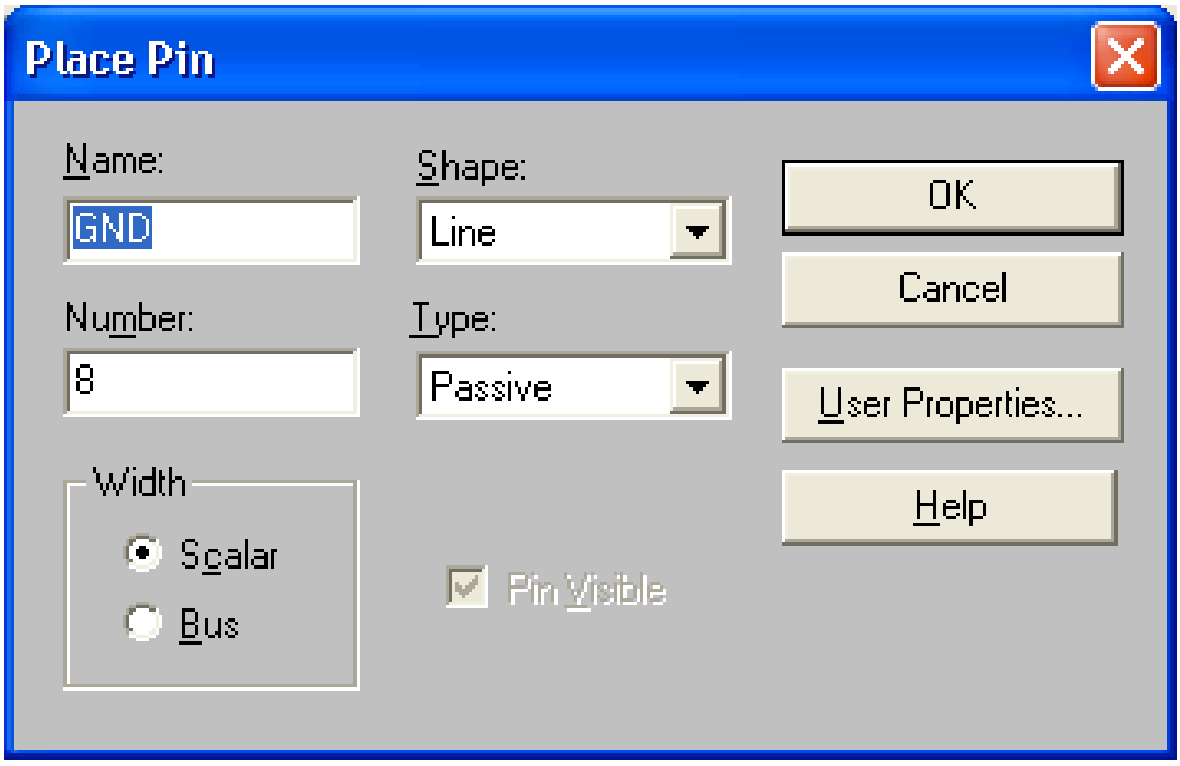


Fig. 5.38, a) Dialog window for placement of a separate component output. (A) and array of pins (b)

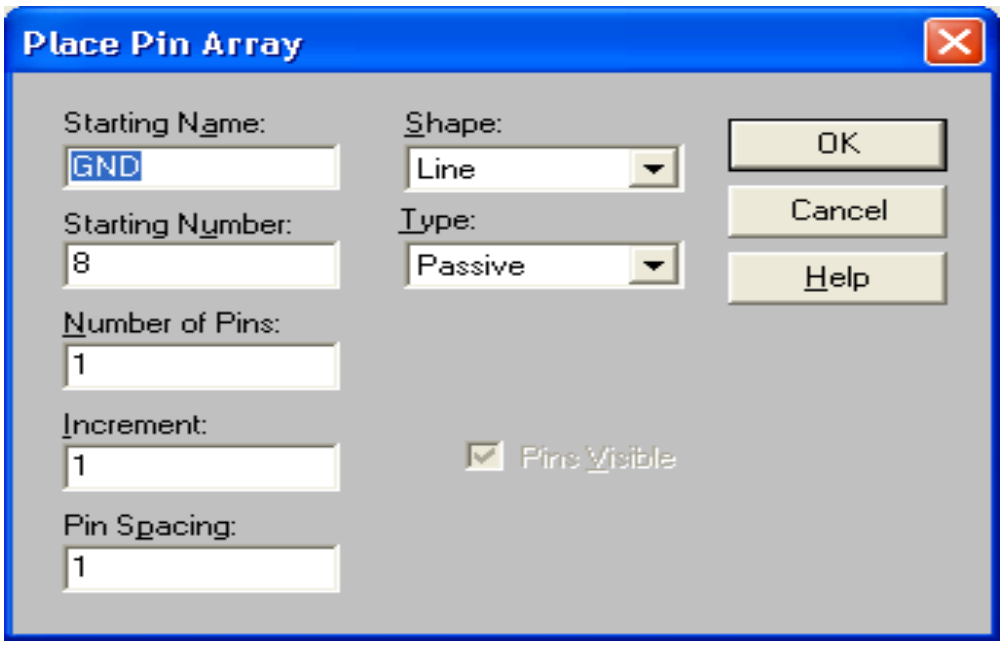


Fig. 5.38, b)

Pin arrays are placed using the Place › Pin Array command, the dialog box of which is shown in fig. 5.38, b). It contains the following information:

- Starting Name – the name of the first output of the array. If the name of the output ends with one of the digits 0...9, then in the names of subsequent outputs, the value specified by the Increment parameter is added to it sequentially. If it is necessary to put a negative sign in the form of a horizontal line above the name, then a slash "\" is entered after each symbol of such a name. For example, entering the characters R\E\S\E\T\ determines the name RESET;
- Starting Number – the number of the first output of the array;
- Number of Pins – the number of pins in the array;
- Increment – the increment of the names of the outputs of the array that are inserted automatically (if the name of the first output ends with a digit);

- Pin Spacing is the distance between adjacent pins of the array in units of the grid step;
- Shape – output form (Table 5.5);
- Type – output type (Table 5.6);
- Pins Visible – display of circuit pins (only for Power type pins).

Sections of both homogeneous and non-homogeneous components can have common outputs, usually these are the outputs for connecting the "ground" and "power" circuits, i.e. outputs of the Power type. Usually, these pins are invisible and they are considered to be connected to circuits which names coincide with the pin names. For heterogeneous components, it is enough to place the "ground" and "power" terminals on at least one section, for homogeneous components, these terminals are automatically placed in all sections (at the same time all their copies have the same names and numbers), so they are always made invisible on the diagram. To make all "ground" and " power" outputs visible (for documentation purposes), you need to select the desired project name in the project manager by clicking the cursor and select the Design Properties command in the Options menu, then select the Display Invisible Power Pins option on the Miscellaneous tab.

After drawing the pins of the section, its contour is drawn and additional text inscriptions are applied (see fig. 5.39, a). It is convenient to apply the functional symbols shown in the table5.7. by the Place> IEEE Symbols command. The image of the next section is opened by the command View> Next Part, - for homogeneous components, it is enough just to apply the numbers of the pins (selecting them sequentially by clicking the cursor), as shown in fig. 5.39, b; for homogeneous components, the image of each section is redrawn. Viewing images of all sections of multi-section components is performed using the View> Package command (see fig. 5.39, c), the transition to editing a separate section is carried out by clicking the cursor.

Component parameters are entered using the Options> Part Properties command, the dialog box of which is shown in fig. 5.16, b. The package parameters of the

component are entered using the Options> Pakage Properties command, the dialog box of which is shown in fig. 5.40. Compiling all these parameters again is quite a painstaking task, therefore, when creating a new component, it is more appropriate to copy a component of the same type to the symbol library using Windows tools and then edit its parameters.

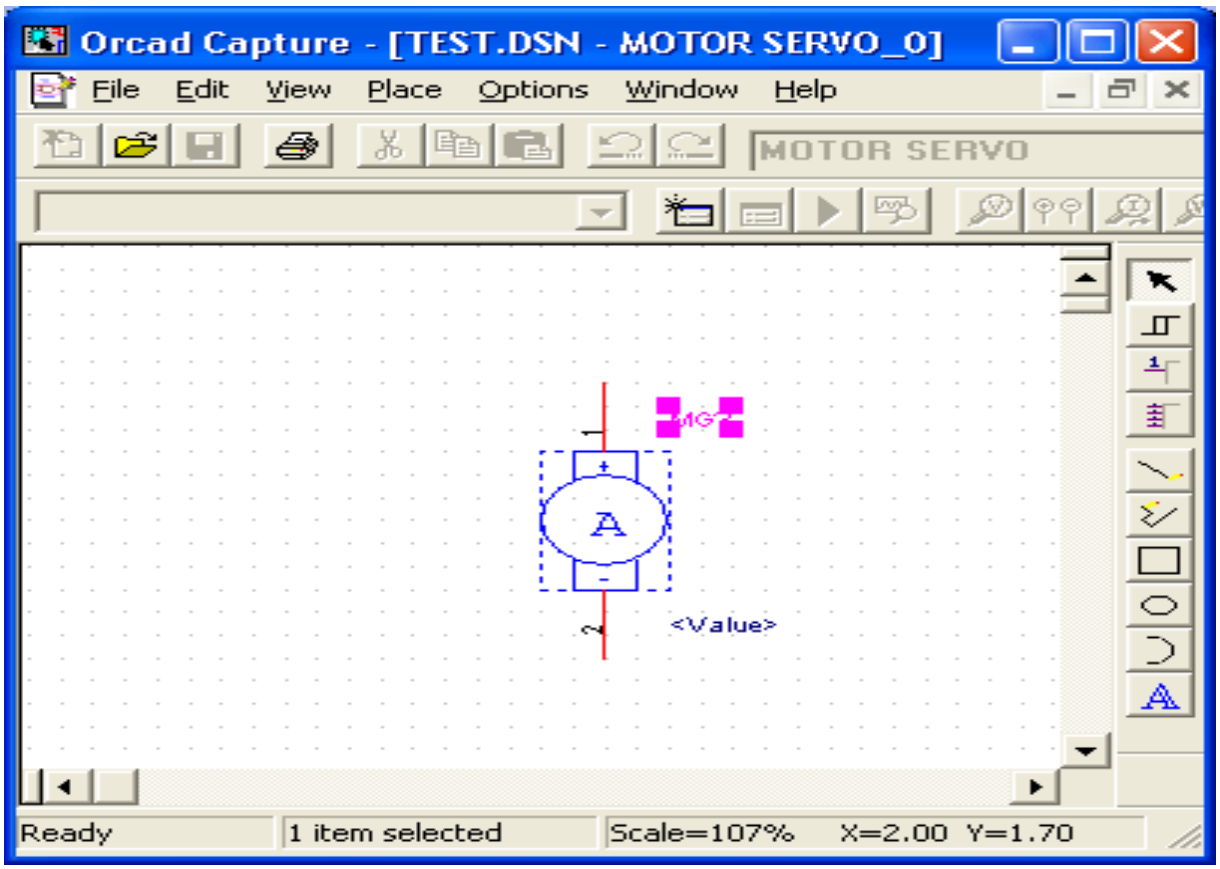


Fig. 5.39, a) Entering graphics and numbers of pins of individual sections of a homogeneous component (a, b) and viewing the packaging of a three-section component (c)

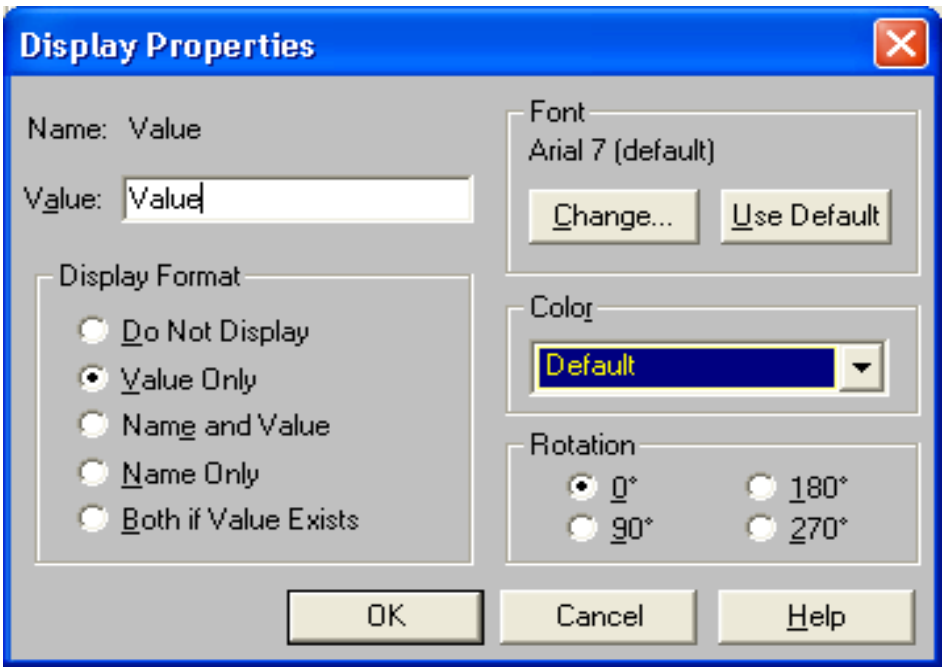


Fig. 5.39, b

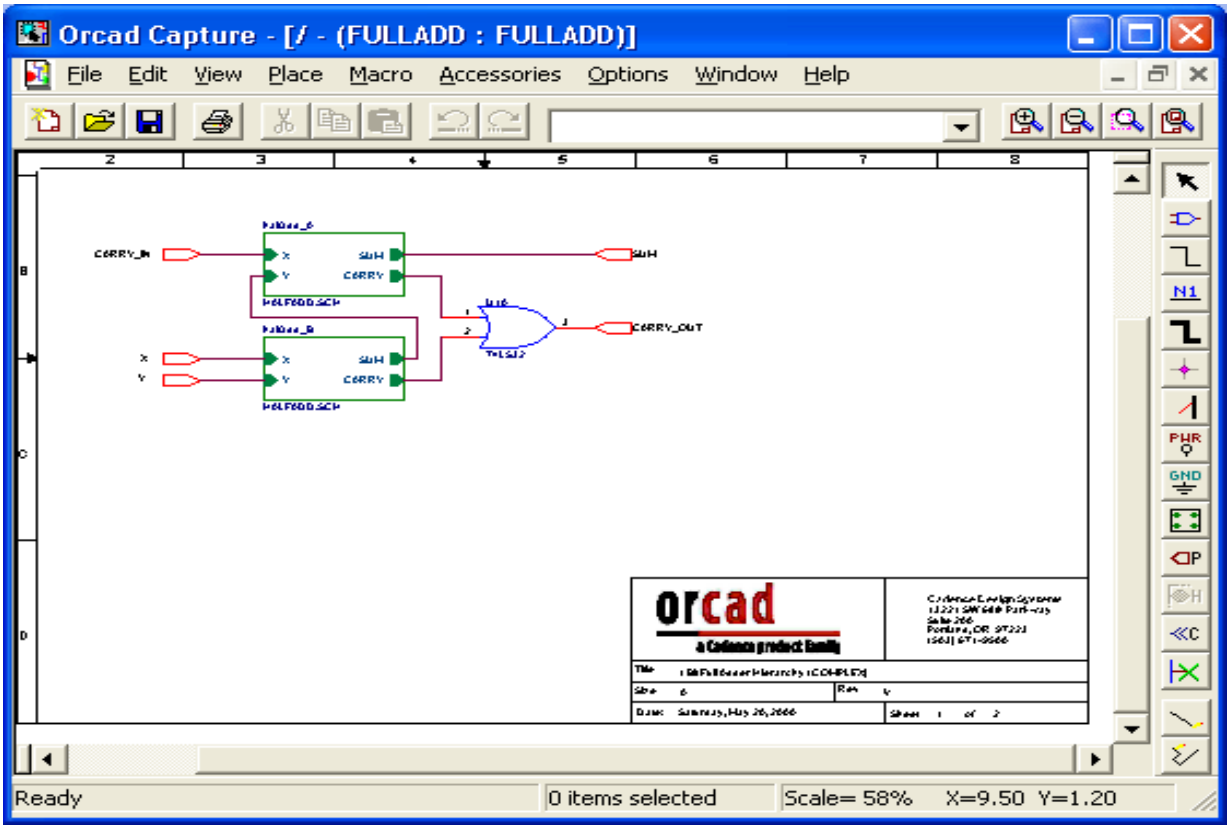


Fig. 5.39, c)

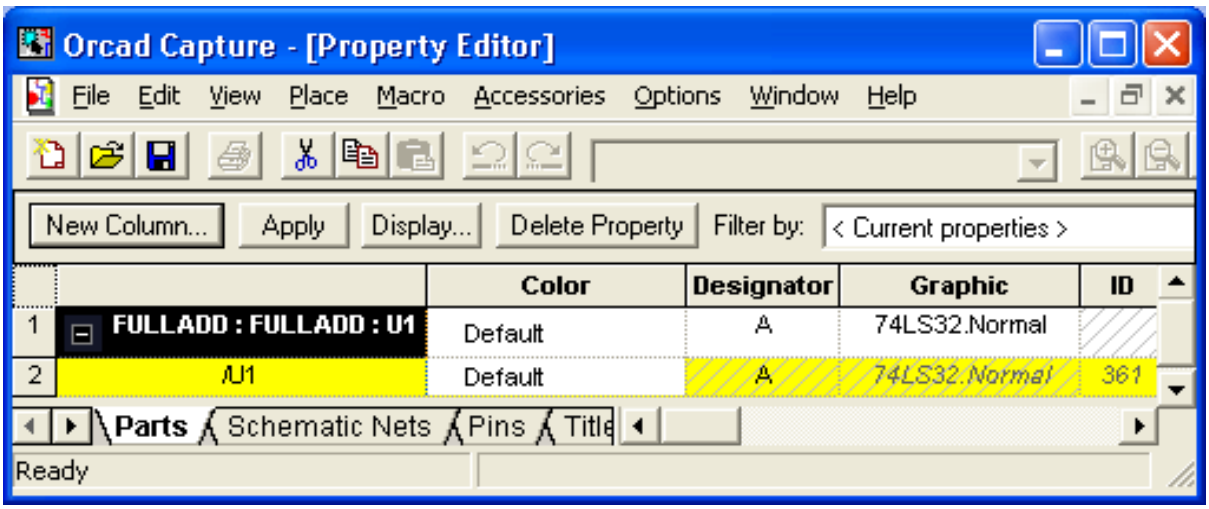


Fig. 5.40. Entering the component packaging settings

The geometric dimensions of the symbol are related to the grid spacing adopted during its construction. If you change this step when entering a symbol on the diagram (it is considered equal to the Pin-to-Pin Spacing parameter on the Page Size tab, fig. 5.15, c), then the sizes of all symbols will change proportionally. Therefore, when creating symbol libraries, it is recommended to select and set the same value of the Pin-to-Pin Spacing parameter in advance.

Remark

As component names in OrCAD Capture, it is allowed to enter Cyrillic characters, for example 133IP7, however, it is not recommended to do so, as there are no guarantees that an error will not occur in the future, for example, when transferring data to another OrCAD module or to another design system. As for the names of the pins, Cyrillic characters are not allowed in them. In general, in order to avoid misunderstandings, it is recommended to use Cyrillic characters only in text inscriptions in imported CAD.

Table 5.7. IEEE symbols.

Символ	Символ
3 State	LE
Active Low Left	NE
Active Low Right	Non Logic
Amplified Left	Open Circuit H-type
Amplified Right	Open Circuit L-type
Analog	Open Circuit Open
Arrow Left	Passive Pull Down
Arrow Right	Passive Pull Up
BiDirectional	Pi
Dynamic Left	Postponed
Dynamic Right	SHIFT Left
GE	SHIFT Right
Generator	Sigma
Hysteresis	

To continue designing after creating a schematic description of the project, the Tools> Create Netlist command of the project manager is executed. When modeling with OrCAD PSpice, this command is loaded automatically; to transfer data to the PP development program OrCAD Layout and others (a total of about 40 user-selected formats is provided for compiling a list of connections), this command is executed manually, having previously highlighted the project name in the project manager.

Before modeling, it is necessary to exclude repetitions of the positional markings of components, and before developing the PP, it is also necessary to pack the component sections in the case. These operations are performed using the

Tools> Annotate command of the project manager, the dialog box of which is shown in fig. 5.19.

Before creating a list of connections, it is advisable to execute the Tools> Design Rules Check (DRC) command to detect errors in the scheme (when starting the PSpice program for simulation, this command is loaded automatically, but in any case, its configuration must be specified beforehand). The inspection report is entered in the * .drc file and is duplicated in the Session Log protocol file (the locations of errors are marked on the diagram with special DRC markers at the user's request).

Two types of design violations are included in the reports:

- Errors - errors that must be corrected;
- Warnings - warnings that can lead to errors during project modeling (it is not necessary to react to them).

After running the Design Rules Check command, a dialog box for the check rules task opens, which has two tabs (fig. 5.41).

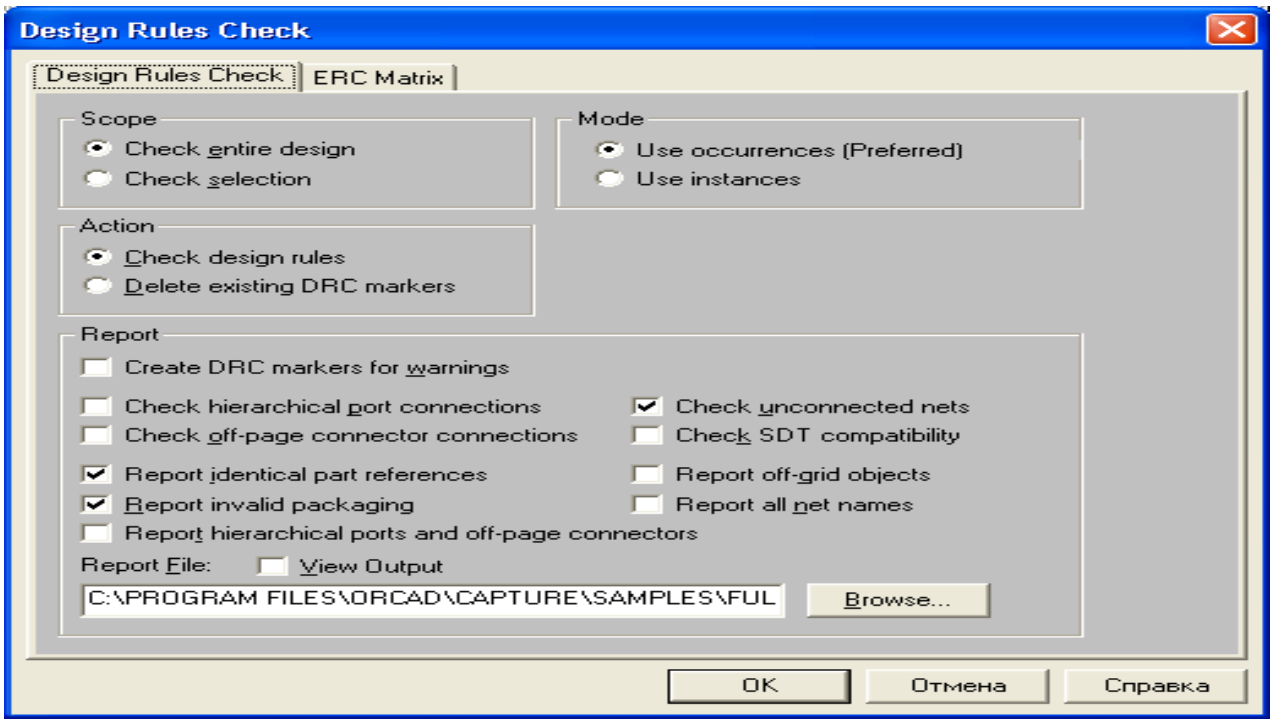


Fig. 5.41, a) Configuration task of the Tools> Design Rules Check command

The Design Rules Check tab (fig. 5.41, a) sets what information is included in the inspection report:

- Scope - checking the entire project (Check entire design), a selected page or several pages (Check selection);
- Action - checking compliance with all design rules (Check design rules) or removing previously applied DRC markers from the scheme;
- Report (selection of information included in the inspection report):
- Create DRC markers for warnings - placement of DRC symbols to warn of possible errors in accordance with the rules specified in the ERC table (DRC markers are always placed in the places where unconditional errors occur):
    - Check hierarchical port connections - checking the coincidence of the names of hierarchical pins and the corresponding hierarchical ports in their equivalent circuits, as well as the coincidence of their total number and types of all pins;
    - Check off-page connector connections - a check of the coincidence of the names of inter-page connectors (connected to chains with the same names) located on different pages of the scheme;
    - Report identical part references - inclusion a list of components with the same positional designations in the report;
    - Report invalid packaging - inclusion a list of components that have the same body, but different packaging information in the report;
    - Report hierarchical ports and off-page connectors - compiling a list of all ports of hierarchical blocks and off-page connectors;
    - Check unconnected nets - detection of nets, each of which is not connected to at least two outputs of components or is not connected to external signal sources, as well as nets that have the same names on different pages of the circuit, but to which inter-page connectors or hierarchical ports are not connected;

- Check SDT compatibility - compatibility check with the graphic editor of schematic diagrams of OrCAD SDT for DOS (this compatibility is necessary if it is planned to save the project diagram in OrCAD SDT format);
- Report off-grid objects - compiling a list of names and coordinates of objects located outside the grid nodes;
- Report all net names - compiling a list of all network names.

- Report File - assigning the file name to the report (by default, its name matches the project name, drc name extension);
- View Output - viewing the results of the check on the screen.

Check rules are set on the ERG Matrix tab, which are recorded in the form of an Electrical Rules Check matrix (ERC, fig. 5.41, b). The rows and columns of the matrix indicate the types of pins of the components and various ports (see Table 2.5). An unpainted cell means permission to connect the corresponding pins, warnings are marked with the symbol W, errors - with the symbol E. For example, according to the the ERC matrix shown in fig. 5.41, b the Output-Input connection is allowed, a warning will be displayed about the Open Emitter-Open Collector connection, and the Power-Output connection will be considered an error.

Therefore, before executing the Tools> Design Rules Check command, it is necessary to edit the content of the ERC matrix in accordance with the features of the current project.

An example of a report file about the results of a project check is given below.

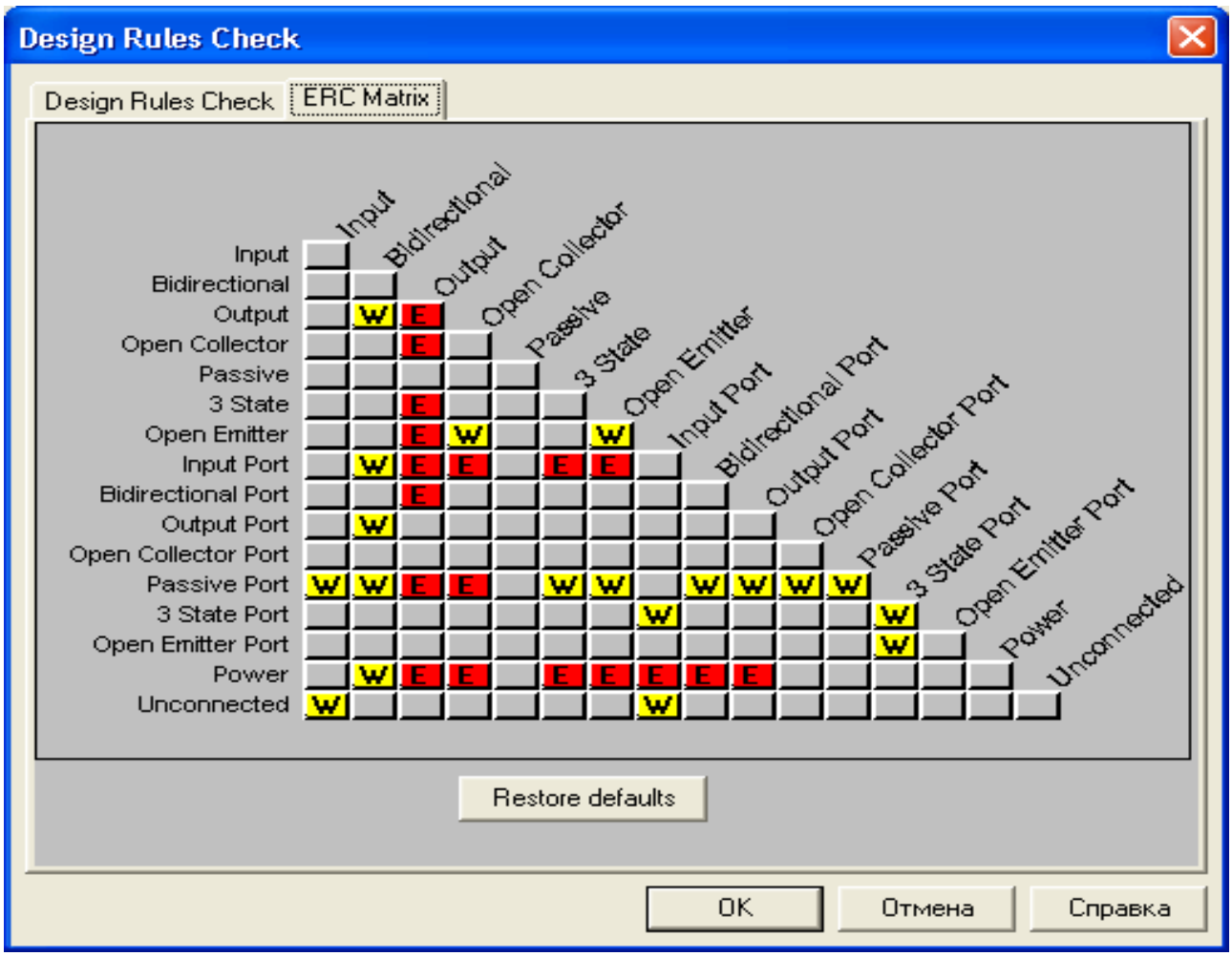


Fig. 5.41, b)

After correcting the detected errors, you can execute the Tools> Create Netlist command to create a list of project connections. There are 9 tabs for choosing the format of the list of connections (fig. 5.42) in the dialog window of this command. The first 8 tabs are related to certain formats:

- EDIF 200 – Electronic Data Interchange Format, which has two varieties; with support (hierarchical netlist) or without support (flat netlist) of a hierarchical structure (depending on the selected system configuration);
- PSpice (files * .net), SPICE (files *.cir) - PSpice and SPICE modeling program format (hierarchical blocks are represented as macromodels using the SUBCKT directory);

- VHDL (files * .vhd), Verilog (files *.v) - description of digital devices in VHDL and Verilog languages;
- Layout - a list of project connections in the format of the printed circuit board development program OrCAD Layout (binary files *.mnl);
- INF - data transfer to the old version of the DOS simulation program OrCAD Digital Simulation Tools 386+ (*.inf files).

Depending on the selected format, the user must adjust the form of the output file in accordance with the specifics of the project (for example, for the OrCAD Layout format, see fig. 5.42, enabling the Run ECO to Layout option will result in the automatic transfer to the OrCAD Layout printed circuit board development program of a new (changed) .mnl file, while the old board must be loaded in Layout earlier). By clicking on the ninth button Other, you can select about 40 more formats, the most famous of which are: Allegro, EEDesigner, Intergraph, HiLo, Mentor, PADS, P-CAD, Scicards, Tango, VST. It should be noted that the list of connections in the format of the Allegro PCB Layout program (Unix and NT) of the Cadence company can also be compiled using the Accessories> Allegros> Allegro Netlist command.

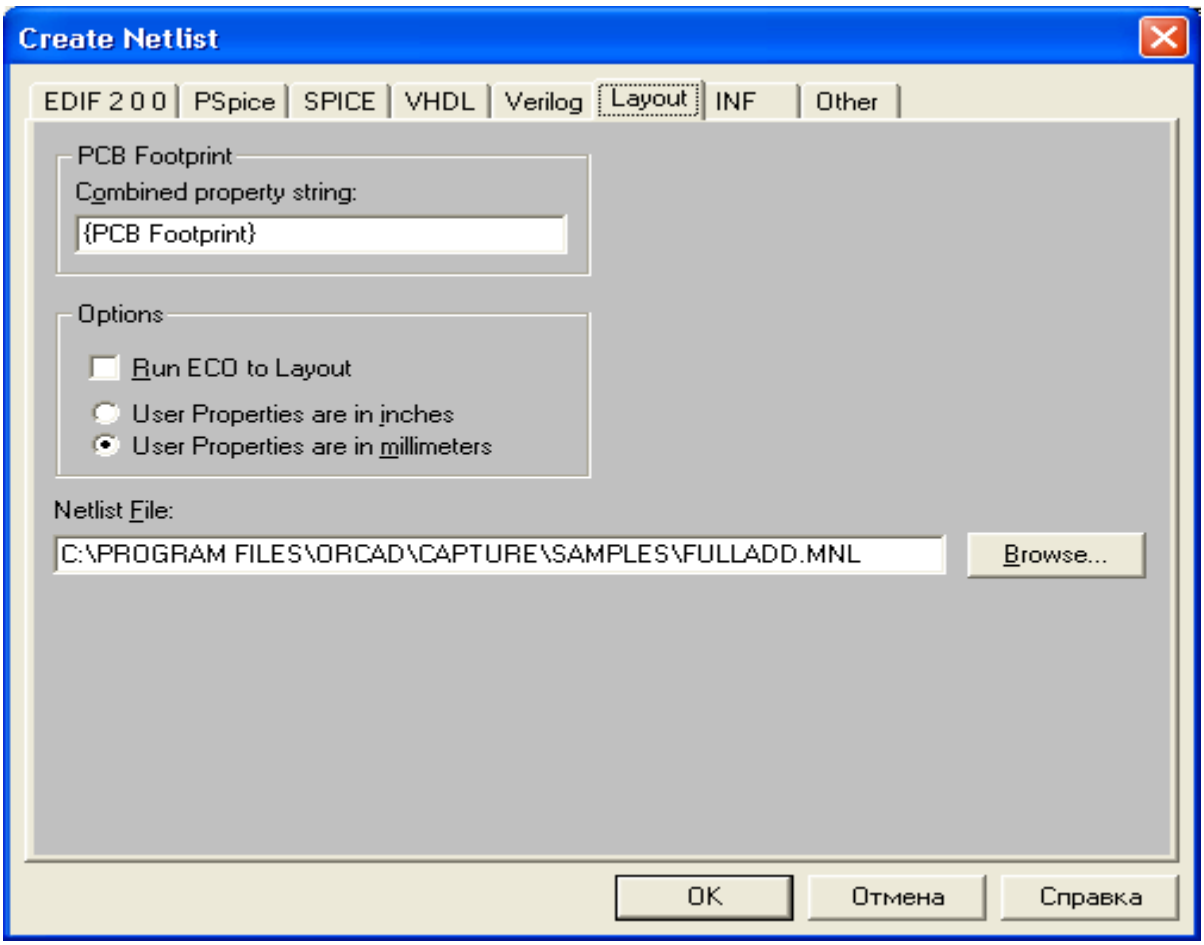


Fig. 5.42. Selection of the format of the list of connections in the dialog box of the Create Netlist command

There are three commands for creating a project specification: Tools> Bill of Materials, Tools> Cross Reference and Reports> CIS Bill of Materials in OrCAD Capture. In turn, the Reports> CIS Bill of Materials command has two subcommands Standard and Crystal Reports (the latter is only in Capture CIS version). Dialog windows of report creation commands are presented in the figure 5.43.

The following options are selected in the dialog window of the Tools> Bill of Materials command (fig. 5.43, a):

- Scope - drawing up a report for the entire project (Process entire design) or its selected part (Process selection);

- Line Item Definition (assignment of the list of variables placed in the report):
  - Header - names of graphs in the report (they must be consistent with those listed in the Combined property string); graph names are separated by "\ t" symbols and are set as text variables, in which Cyrillic characters are allowed;
  - Combined property string - the task of listing the values placed in the graphs of the report; the names of the values are enclosed in parentheses and are separated from each other by "\ t" symbols; for example, to include the positional designations of components and their names in the report a reference line {Reference} \ t {Value} is compiled; these values are separated by tabulation marks in the report;
  - Place each part entry on a separate line - the output of the data of each component in a separate line;

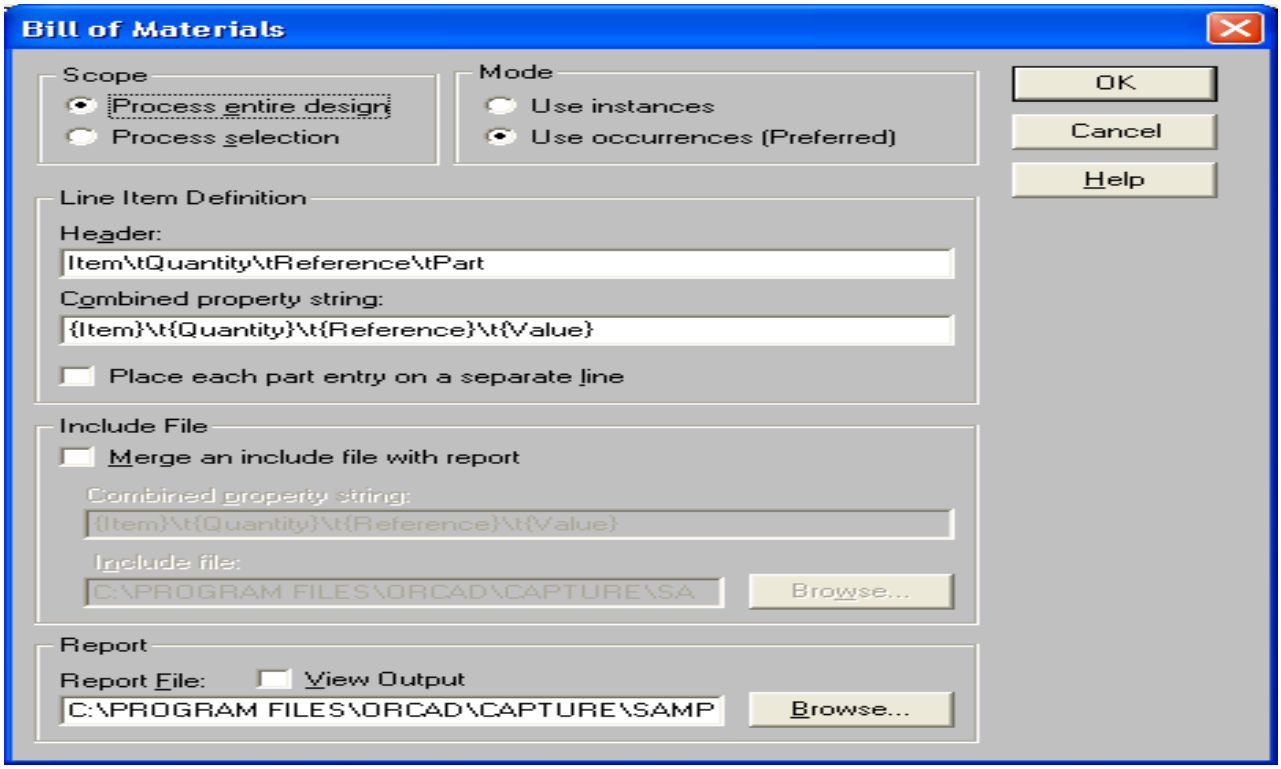


Fig. 5.43, a Dialog windows of the Tools> Bill of Materials (a), Tools> Cross Reference (b), Reports> CIS Bill of Materials> Standard (c) and Reports> CIS Bill of Materials> Crystal Reports (d) commands

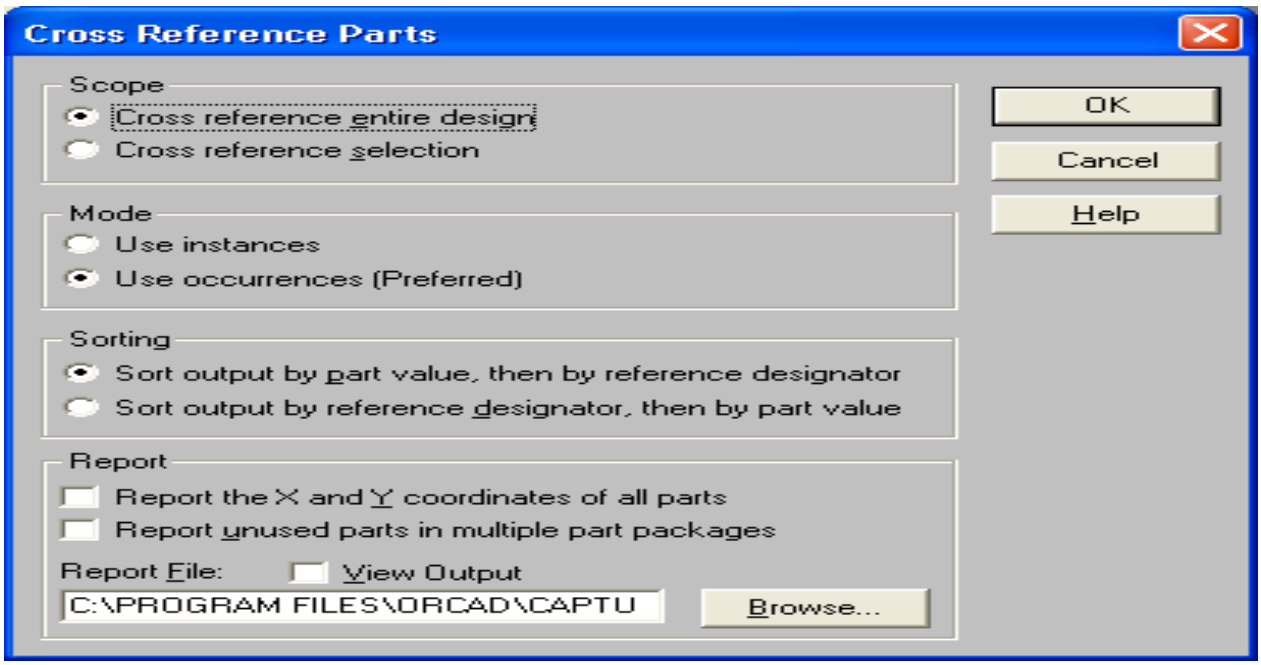


Fig. 5.43, b)

Fig. 5.43,c)

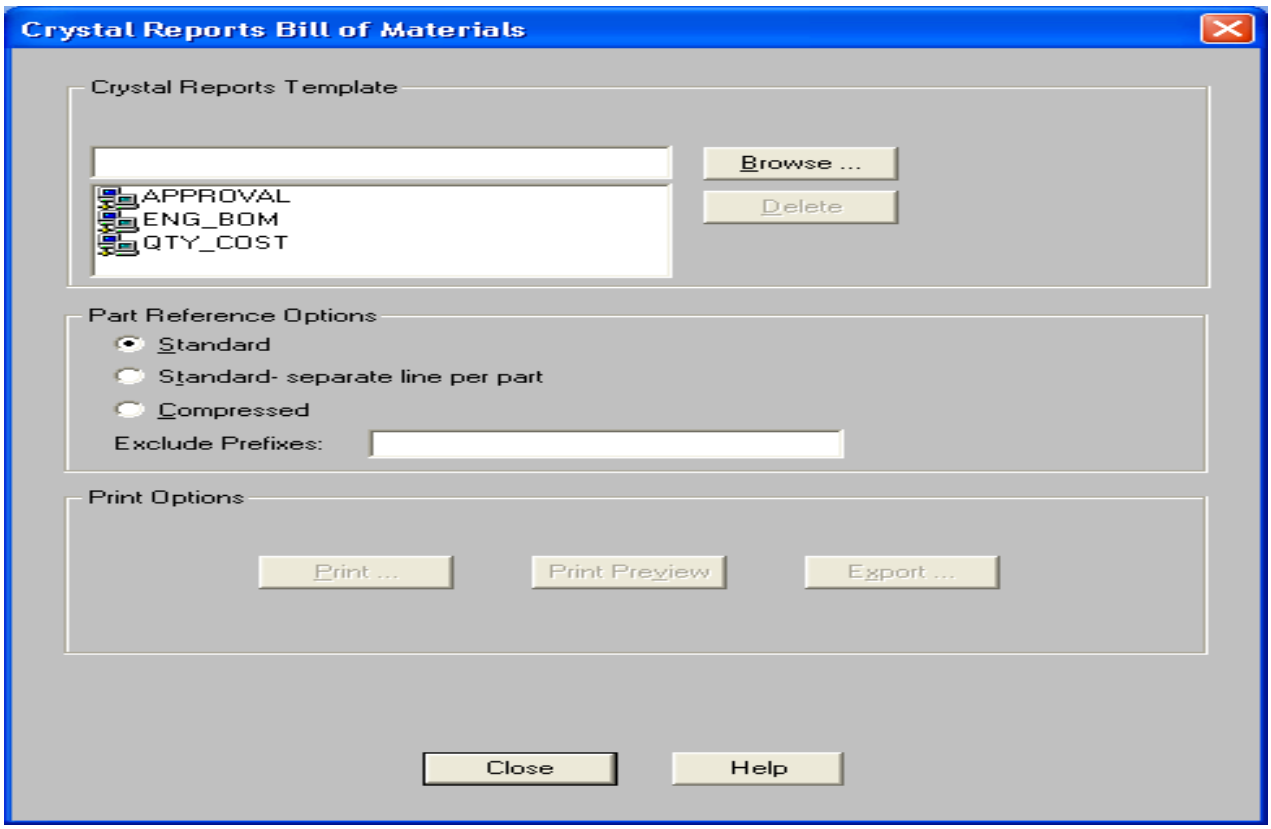


Fig. 5.43, d)

- Include file:
  - Merge an include file with report - inclusion additional information placed in the Include file in the report;
  - Combined property string - task of listing the values enclosed in the Include file in apostrophes (see below);
  - Include file - the name of the inclusion file;
- Report File - the name of the report file (by default, the name of the project with the extension .bom is specified as its name);
- View Output - output of the output file for preview to the screen.

If the content of the first column of any line of this file matches the data of the Include File Combined property string, then the content of the other part of this line of the Include file is added to the report file.

Let us give an example of a fragment of a project report compiled by the Tools> Bill of Materials command (see fig. 5.43, a) using the Include file (its text is given above):

Bill Of Materials September 22.2000 21:21:33 Pagel		
№п/п (Item)	**Поз. обозн. (Reference)**	**Номинал Описание компонента ГОСТ (Value)**
1	C1	1 uF Керамический конденсатор
2	R1	1k МЛТ-0.255%
3	U1	74107 Warning: The part is not in the include file
4	DD1	133LA3 4 логических элемента 2И-НЕ
		И63.088.023ТУ7

The following options are selected in the dialog window of the Tools> Cross Reference command (fig. 5.43, b):

Scope (selection of the project or its part):

- Cross reference entire design - drawing up a report for the entire project;
- Cross reference selection - drawing up a report for the selected part of the project;

- Sorting:
  - Sort output by part value, then by reference designators - sorting of lines from the report initially by <Value> parameters of components and by reference designators;
  - Sort output by reference designators, then by part value - sorting of lines from the report first by positional designations of components and then by parameters<Value>;
- Report:
  - Report the X and Y coordinates of all parts - conclusion of the report of the X, Y coordinates of the location of all components on the diagram;

- Report unused parts in multiple part packages - conclusion of the report of information on unused sections of multi-section components;

- Report File - the name of the report file (by default its name is the project name with the extension.xrf);
- View Output - output of the output file for preview to the screen.

The following options of the standard report are selected (the command is available only for Capture CIS) in the dialog window of the command Reports > CIS Bill of Materials > Standard (fig. 5.43, c):

- Template Name - selection from the list of the name of the report template or entering the name of a new template (the Seagate Crystal Reports program is required to create new templates (see the websites www.orcad.com/cis/crystal.htm, www.seagatesoftware.com, www.softline .ru), moreover with OrCAD 9 compatible versions of this program are not less than 6;
- Report Properties (adjusting report parameters):
  - Selected Properties (selection of parameters included in the report): Availability, Datasheet, Distributor, Distributor Part Number, Rating, Schematic Part Path, Source Library, Source Package, Tolerance, Voltage;
  - Output Format - list of selected parameters;
- Keyed - marked parameters are grouped in the report on one line (see below);
- Part Reference Options (choice of report file format):
  - Standard - standard form - grouping of marked parameters on one line;
  - Standard-separate line per part - each component is placed on a separate line;
  - Compressed - compressed form - information about the same components is placed on one line;
  - Exclude Prefixes - a list of prefixes for positional designations of components that are not included in the report.

An example of a fragment of a project report compiled using the Reports> CIS Bill of Materials> Standard command is given in fig. 5.44, a.

The following options are selected in the dialog box of the Reports> CIS Bill of Materials> Crystal Reports command (fig. 5.43, d):

- Crystal Reports Template - a binary report template file compiled by the Crystal Reports program (standard templates have the names ENG_BOM, QTY_COST, APPROVAL);

Позиц. обозн.	Наименование	Номинал	Документ	Группа
C1	ОС К53-18	-3В 33 икФ+-2%	ОЖО.454.136 ТУ ОЖШ4201 ТУ	Конденсаторы
C2	К10-73-16.М47	2.5 мкФ	ЯАВЦ.673511.004ТУ	Конденсаторы
DA1	3ОД101АОСМ		ТТ0.336Ш2ТУП0.070.052	Микросхема аналоговая
DA2	3ОД101А ОСМ		ТТ0.336.012ТУ ПО.070.052	Микросхема аналоговая
DD1	ОСМ 537РУ8А		ВК0.347.243-08ТУ ПО.070.052	Микросхема цифровая

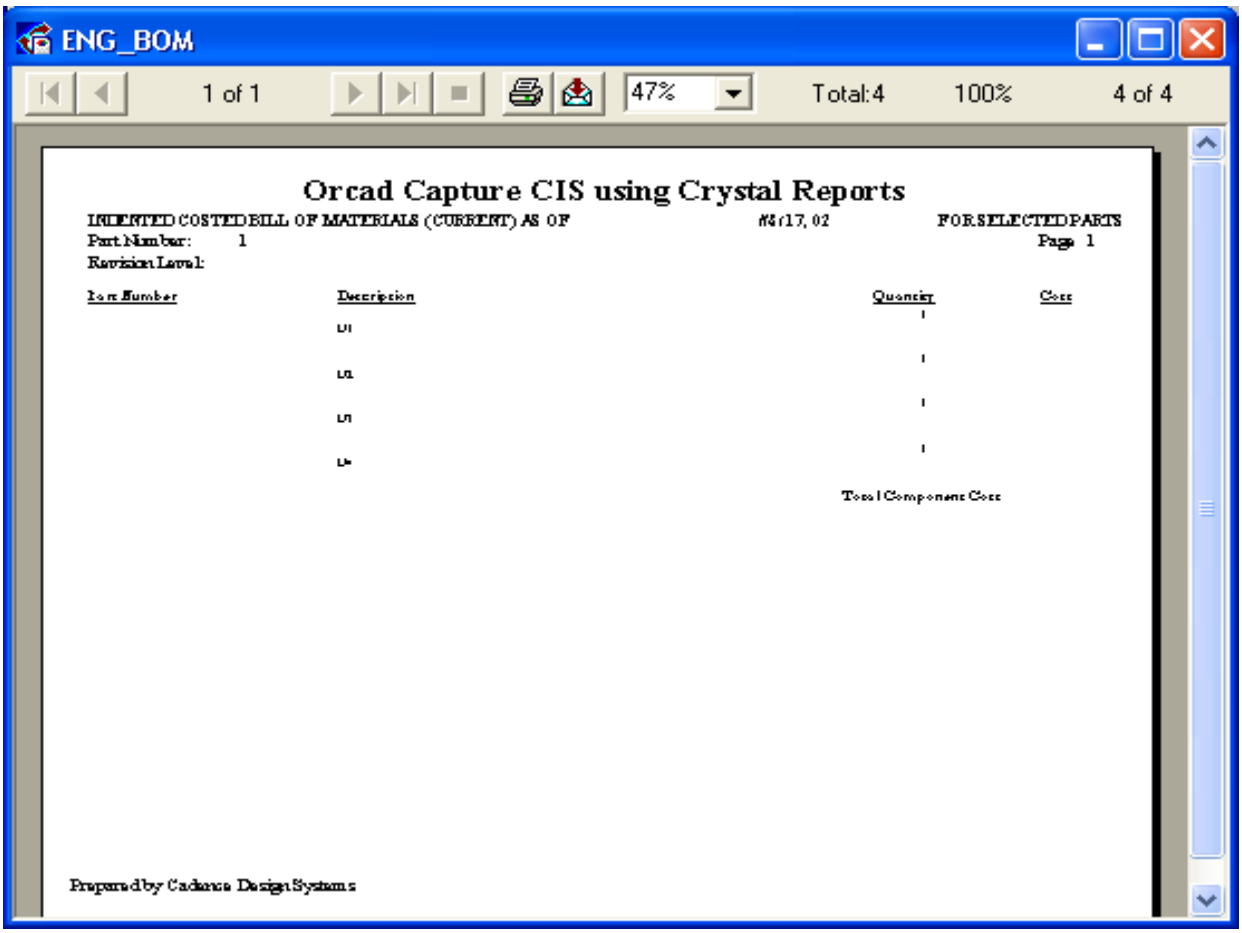


Fig. 5.44, a) Project reports in the standard form of OrCAD CIS (a), in the form of Crystal Reports with the ENG_BOM template (b) and the template compiled according to ESKD (c)

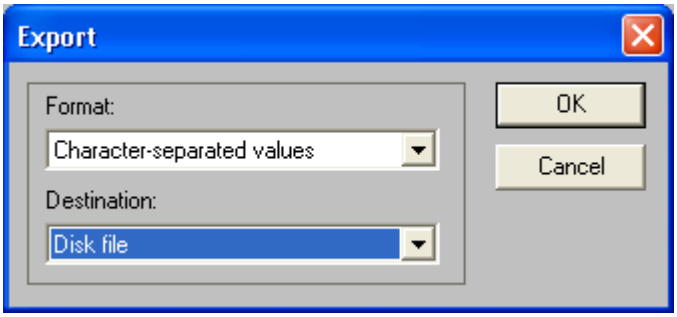


Fig. 5. 44, b)

Part Reference Options (choosing the report file format):

- Standard - standard form;

- Standard-separate line per part - each component is placed on a separate line;
- Compressed - compressed form - information about the same components is placed on one line;

Print Options (selecting the print format for the printer):

- Print - direct output to the printer;
- Print Preview - preview;
- Export (report export):

Format: Character-separated values, Comma-separated values, Crystal Reports, Data Interchange Format, Excel, HTML, Lotus 1-2-3, ODBC, Paginated Text, Record style (columns of values), Rich Text Format, Tab-separated text, Tab-separated values, Text, Word for Windows document;

Destination: Disk file - disk file; Exchange Folder - Exchange folder; Microsoft Mail - e-mail.

An example of a fragment of a project report compiled using the Reports> CIS Bill of Materials> Crystal Reports command using the ENG_BOM template is shown in fig. 5.44, b. We remind you that changing the template of the CIS Bill of Materials>Standard and Crystal Reports reports is done using the Crystal Reports program, which is supplied separately.

The File> Import Design command imports schemes or libraries from the PSpice Schematics program into OrCAD Capture, as well as schemes recorded in EDIF 2.0.0 and PDIF (P-CAD 8.x - 2001) formats. In the dialog box of this command (fig. 5.45) there are three tabs PSpice, EDIF and PDIF, respectively.

Schematics imported from PSpice Schematics (PSpice tab) are suitable for modeling without additional editing. In the Open panel the name of the original scheme * .sch file is indicated, and in the Save As panel - the name of the * .obj project in the OrCAD Capture format. In addition, you must specify the name of the configuration file * .ini of the Schematics program on the MSIM.INI panel.

When selecting the Translate Hierarchy option, diagrams of hierarchical blocks are created on individual pages; when selecting the Consolidate all Schematic files into one Design file option, all schematic pages are included in a separate project.

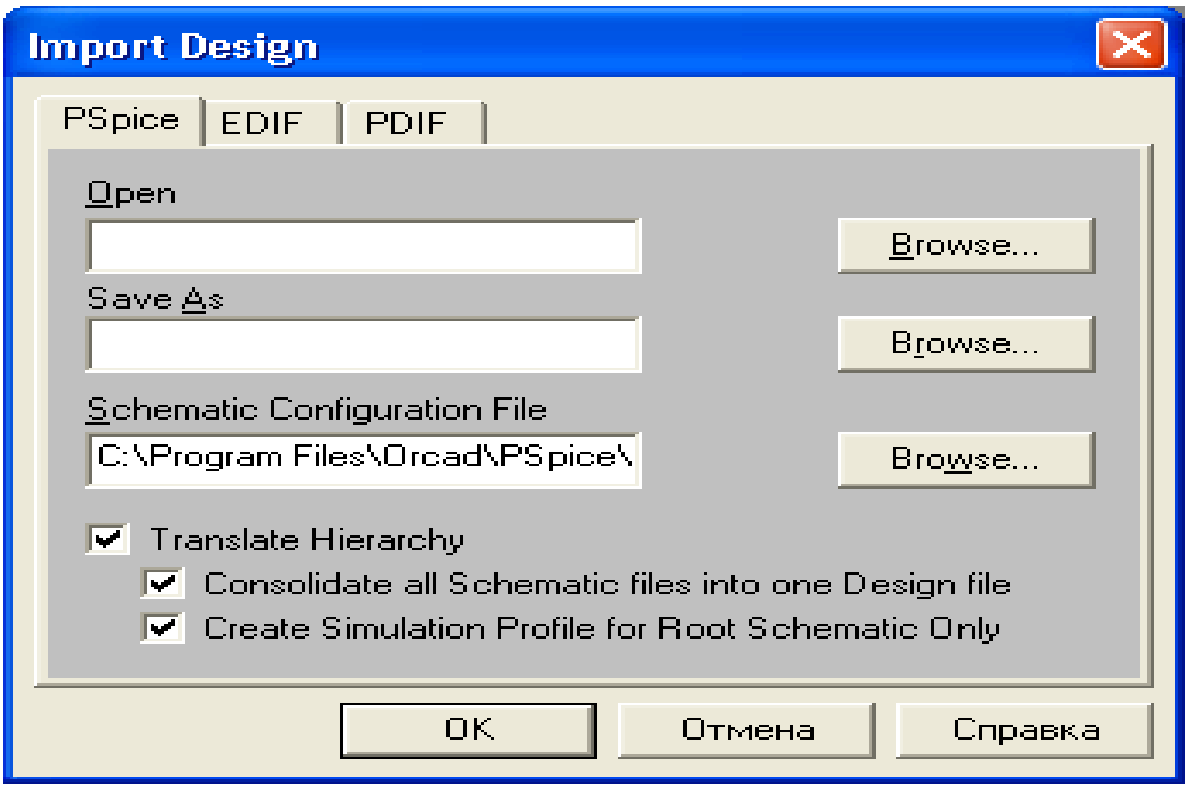


Fig. 5.45, a) Import of schemes from PSpice Schematics (a) and P-CAD (b) programs

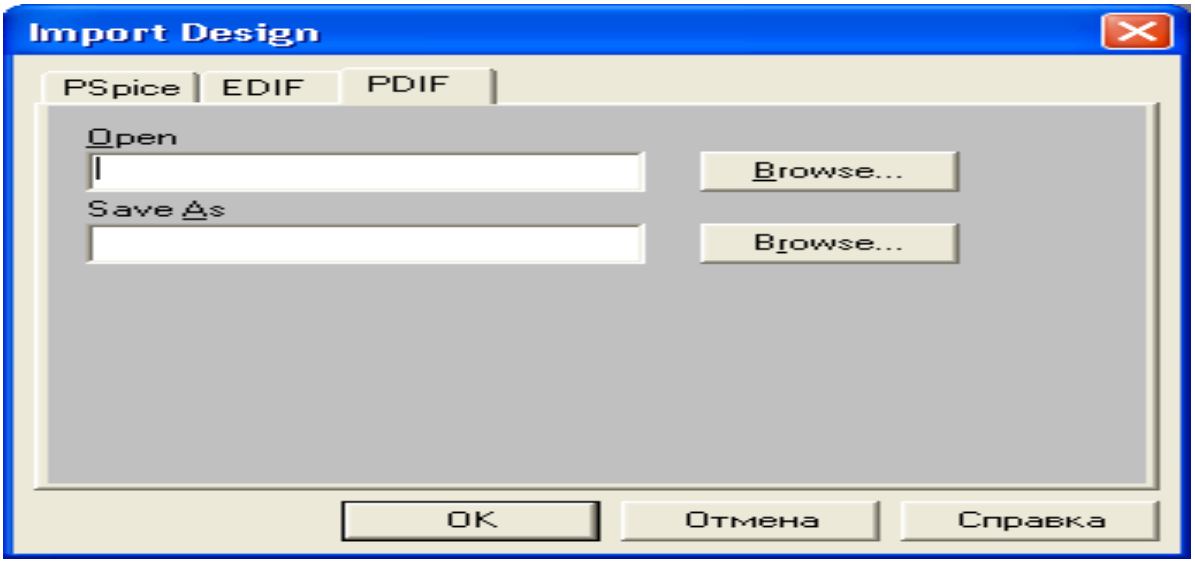


Fig. 5.45, b)

To import schemes from P-CAD 8.x, you must first create a *.pdf text file using P-CAD and specify its name in the Open panel (see fig. 5.45, b). In addition, it is necessary to specify a list of P-CAD and OrCAD scheme circuit name correspondences in the pcadi.ini conversion configuration file. An example of such conversion of schemes from P-CAD 8.7 to OrCAD Capture 9.2 is shown in fig. 5.46.

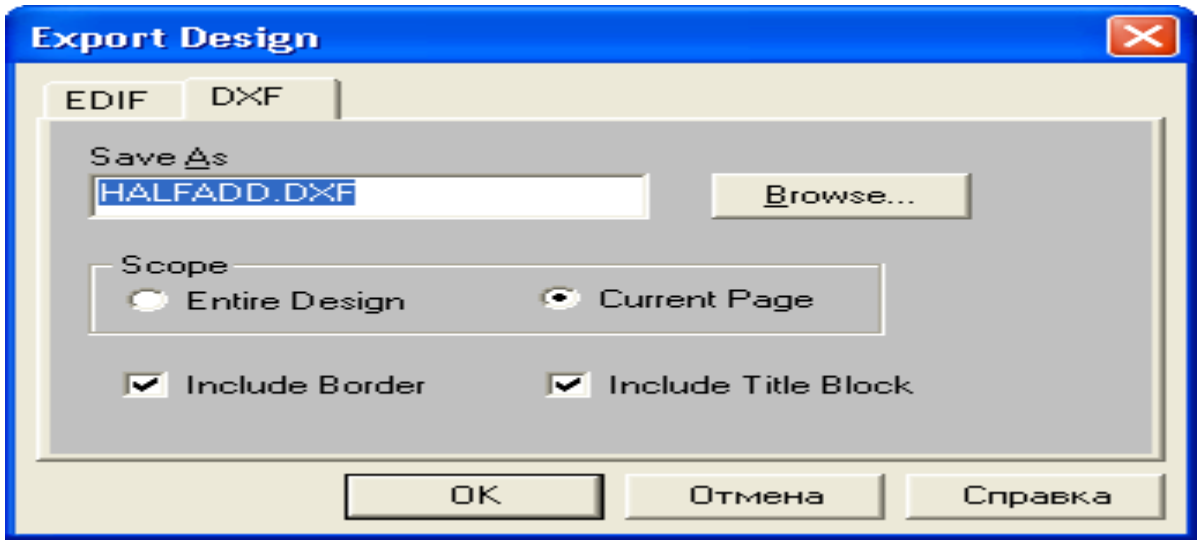


Fig. 5.46. Scheme conversion from P-CAD to OrCAD Capture

Schemes are exported using the File> Export Design command in EDIF 2.0.0 and DXF (AutoCAD) formats.

Export and import of the parameters of the components included in the project is performed using the Tools> Export (Import) Properties command.

SECTION VI . EXAMPLES OF IMPLEMENTATION OF VARIOUS FUNCTIONAL DEVICES.

6.1. Adders.

An adder is a combinational device designed to perform the operation of arithmetic addition of numbers presented in the form of binary codes.

Adders are one of the main components of an arithmetic logic device (ALD). The term "accumulator" covers a wide range of devices, from the simplest logic circuits to the most complex digital devices. Common to these devices is the arithmetic addition of numbers presented in binary form.

The adder adds two numbers, working according to the same rules as we do when we perform the "column" addition (fig. 6.1).

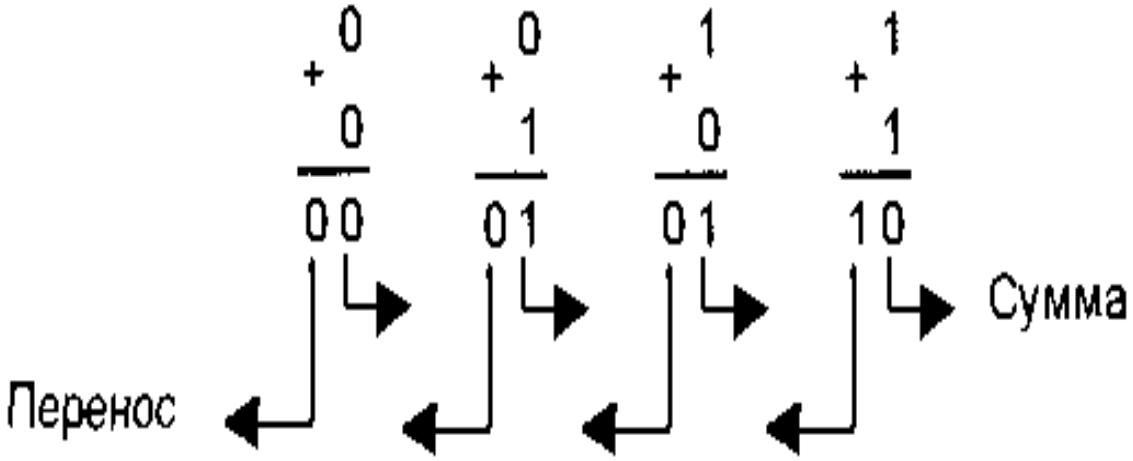


Fig. 6.1. The principle of adding two numbers.

When adding two numbers in any positional system, three numbers are added in each digit:

- Numbers of the summand;
- Numbers of a given digit of the addend;
- Digits 1 or 0 (transfer from the least significant digit).

As a result of the addition for each digit, the numbers of the sum and the digits 1 or 0 of the transfer to the next most significant digit are obtained.

The classification of adders can be performed according to various criteria:

1) According to the number of outputs adders are distinguished:
   - Half- adders designed for adding two single- digit codes;
   - single-digit adders , designed to add two single- digit codes (in contrast to a half-adder, the value of the transfer from the most significant digit is taken into account);
   - multi-bit adders, designed to add two multi-bit codes.

2) According to the structure half-adders can be:

- serial, in which the addition operation is performed in digits, starting with the least significant one;
- parallel, in which all bits of input codes are summed up simultaneously;
- combinational - do not have their own memory;
- accumulative- provided with its own memory, in which the results of calculations are stored; in this case, each new summand is added to the value already avaible in the device.

3) By timing method:

- synchronous;
- asynchronous

A binary half-adder.

A half adder has two inputs A and B for two terms and two outputs: S (sum) and P (carry). The operation of the device is reflected in the truth table 6.1 .

Table 6.1. The truth table of a half adder

Inputs	Outputs

A	B	P	S
0	0	0	0
0	1	0	1
1	0	0	1
1	1	1	0

The logic structure of a half-adder is such that the S output state represents the sum bit, and the P output represents the carry bit. This follows from the table. 4.1. The operation of the half-adder is described by the following equations:

$$S = AB \vee A\,B = A + B; P = A * B$$

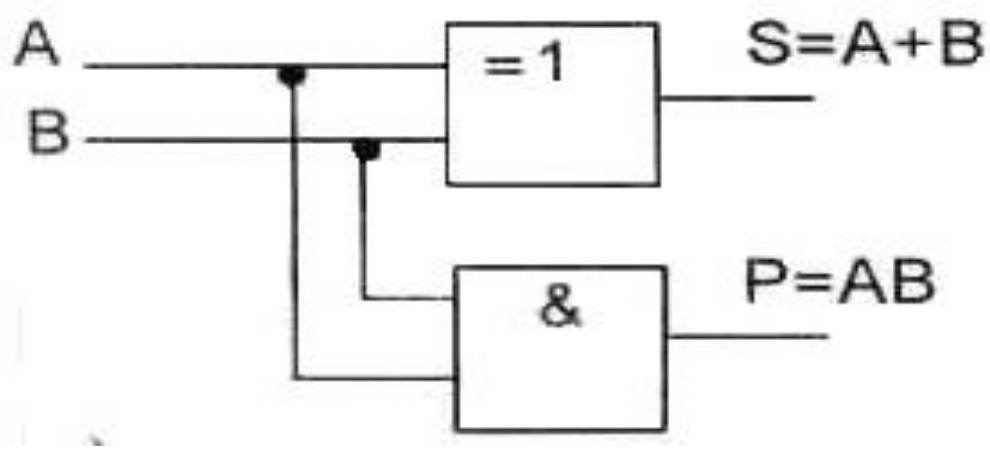


Fig. 6.2. The logic structure of a half-adder

Full one-bit binary adder.

A full one-bit binary adder (fig. 6.3.) has three inputs: a, b for two terms and **p** for carrying from the previous (least significant) digit and two outputs: **S** – sum, **P** – carry to the next (most significant) digit. The full binary adder is denoted by the letters SM. Its work is reflected in the truth table (table 6.2).

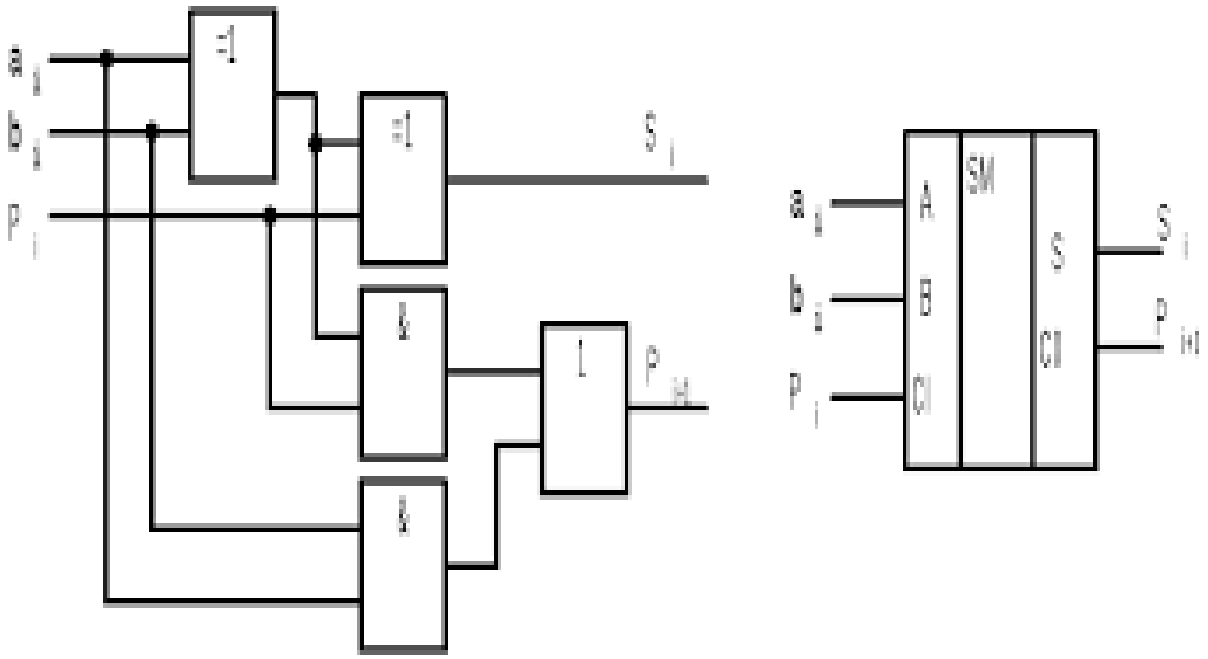


Figure 6.3. The logic structure of a full adder

Table 6.2. The truth table of a full adder

a_i	b_i	P_i	P_{i+1}	S_i
0	0	0	0	0
0	1	0	0	1
1	0	0	0	1
1	1	0	1	0
0	0	1	0	1
0	1	1	1	0

1	0	1	1	0
1	1	1	1	1

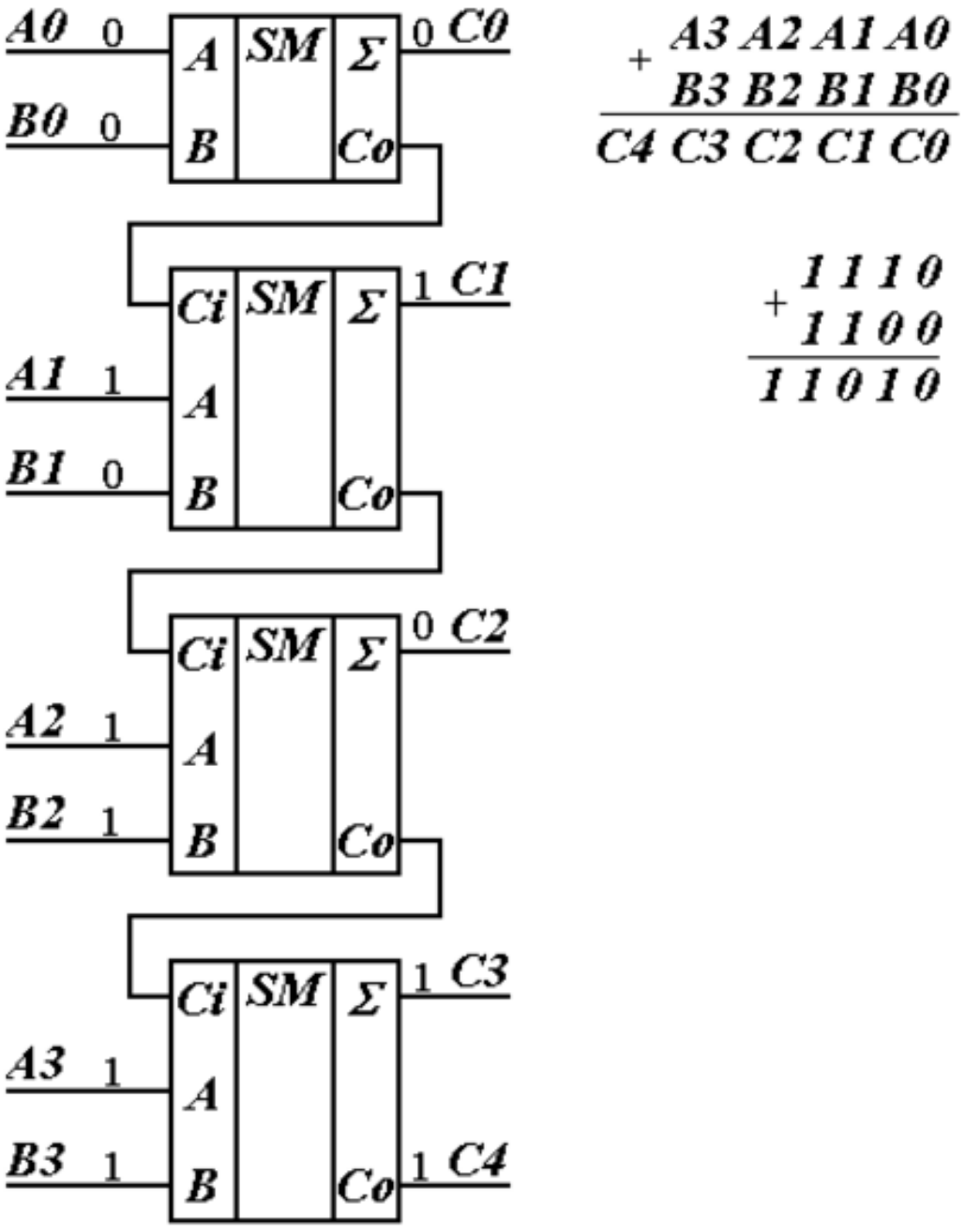


Fig. 6.4. The principle of operation of a multi-bit adder

Implementation of a 16-bit adder using CAD ALTERA MAX + plus II 10.2

The general circuit diagram of the implementation of the 16-bit adder is built on the basis of the main 3 elements (which henceforth contain various logical elements):

- Input buffer;
- 16-bit adder;
- Output buffer.

The diagram shows:

aa [16..1] is the first 16-bit number;

bb [16..1] – the second 16-bit number;

clk – clock frequency;

yy [16..1] is the result of addition;

y17 – carry bit.

The input is the global clock frequency clk. The two numbers aa and bb are fed to vhbufer, which separates the 16-bit numbers bit by bit. The bitwise addition takes place in the sum16 block, and in the future, the opposite action to vhbufer occurs in vyhbufer (fig. 6.5).

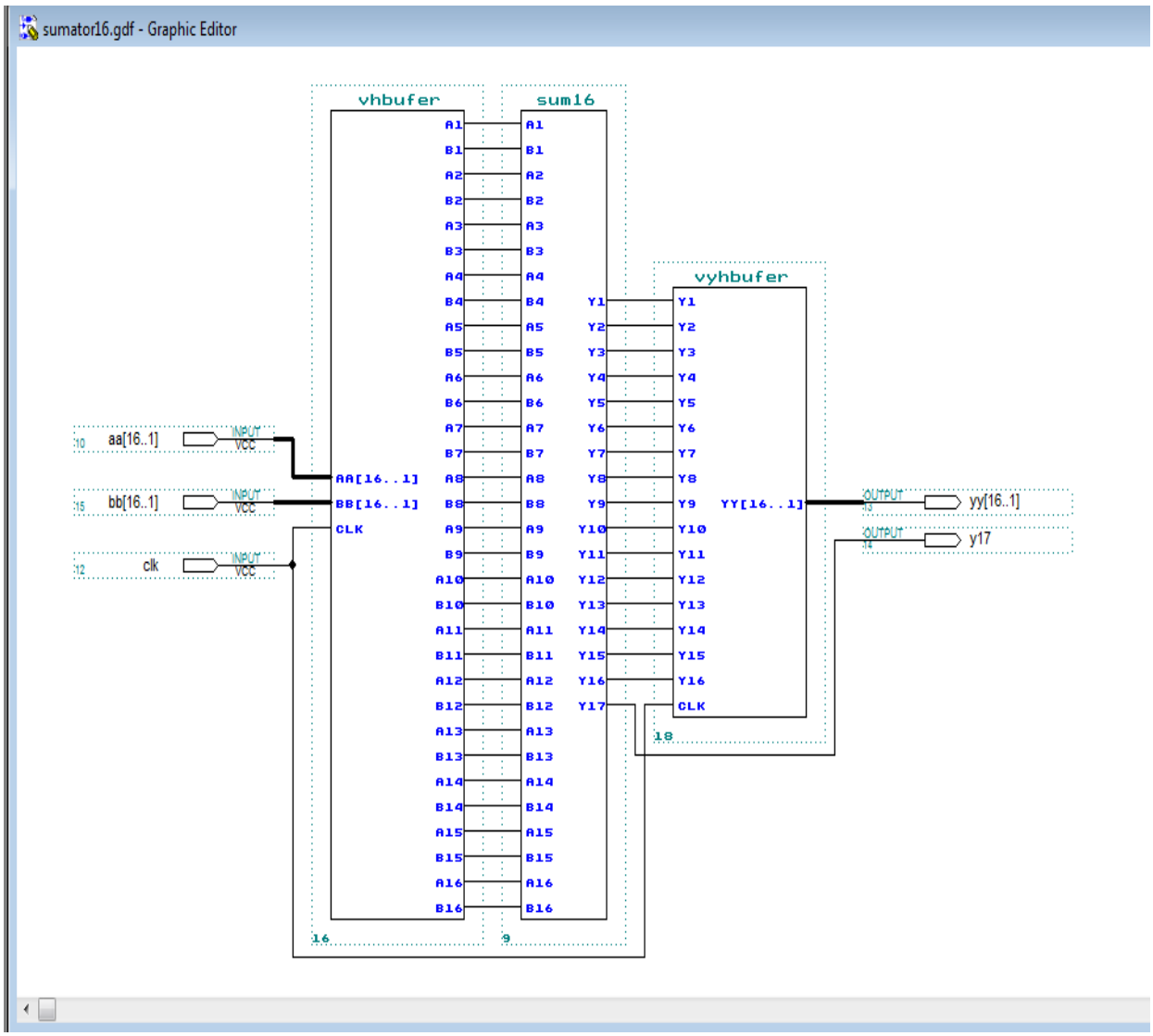
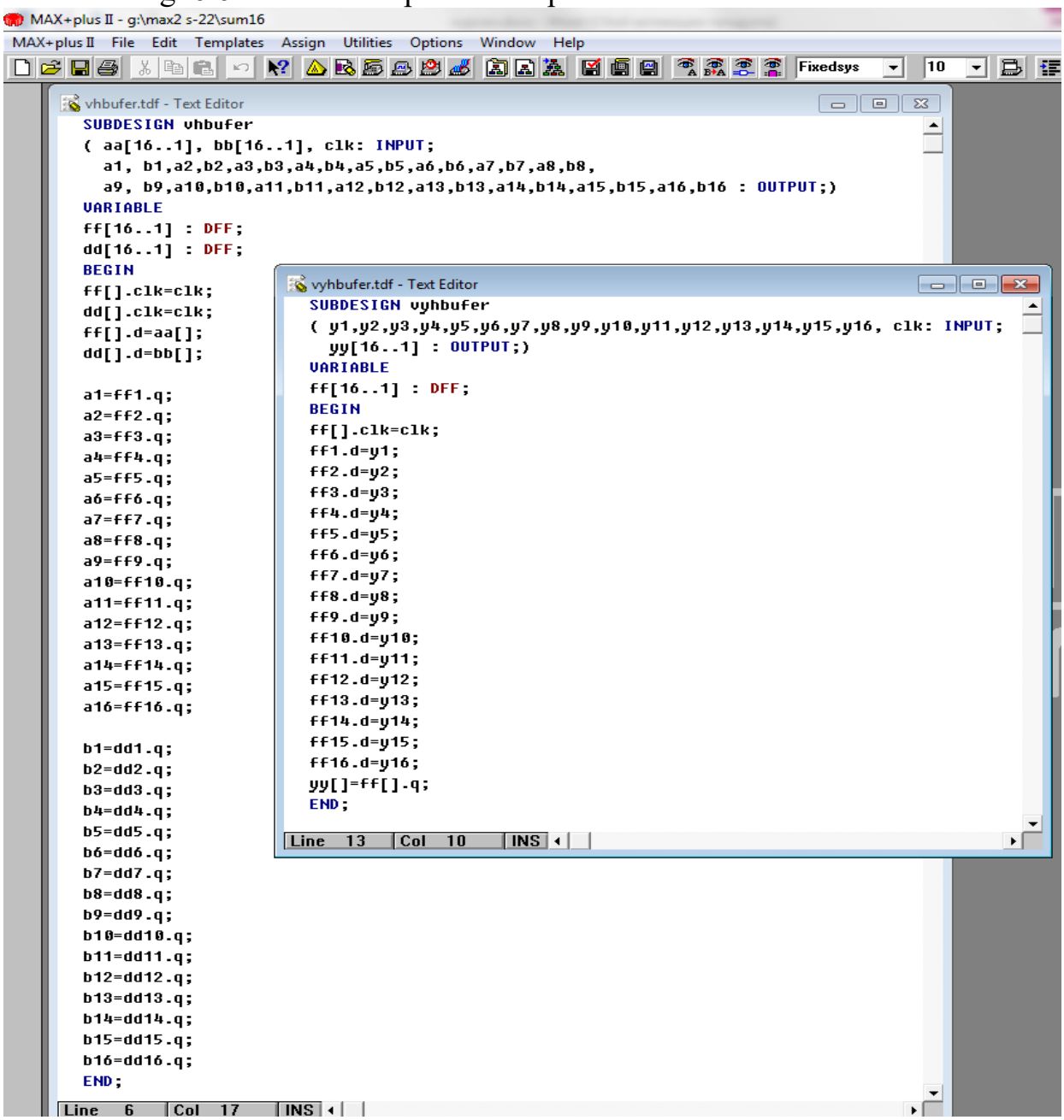


Fig. 6.5. Functional diagram of a 16-bit adder.

The description of the blocks of input and output buffers is given below. The implementation of these blocks is based on the operation of D triggers (*DFF* is a synchronous D trigger (D - FlipFlop)). Each bit of a 16-bit number is written as each bit of a trigger.

Blocks of input and output buffers are necessary to simplify the implementation of a 16-bit adder using CAD ALTERA MAX + plus II 10.2 BASELINE in the graphics editor (instead of a bulky image with many lines, we use buses) (fig. 6.6).

Fig. 6.6. Blocks of input and output buffers.



Sum block16 is implemented using adders and a half-adder (fig. 4.7). As it was mentioned earlier, a half-adder has two inputs and two outputs: sum and carry. The adder has 3 inputs (the carry bit from the previous bit) and 2 outputs.

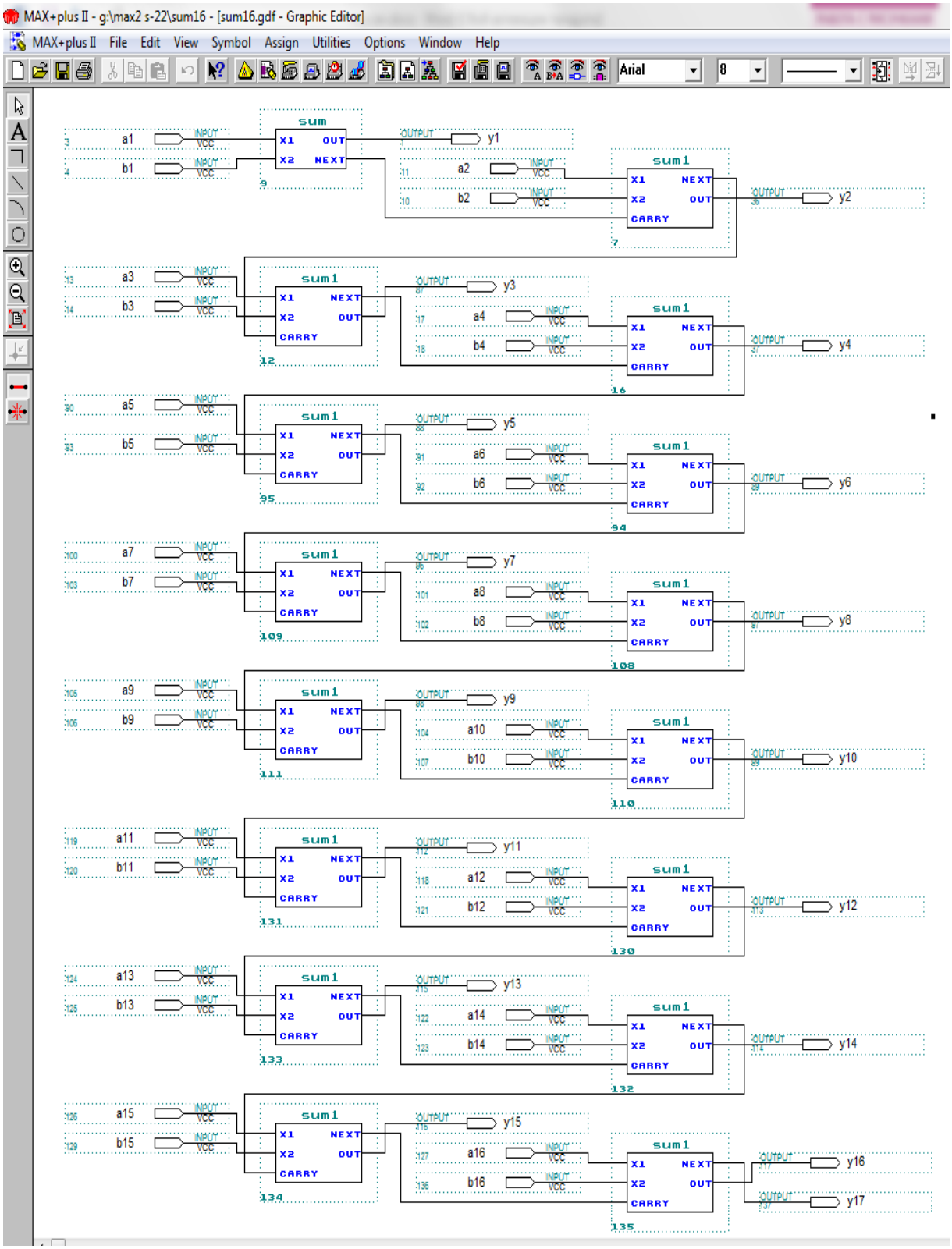


Fig. 6.7. 16-bit adder

A half-adder works by operating 2 logic elements: AND (logical AND) and XOR . The correct work can be verified using a truth table. A full adder can be built on the basis of half adders (fig. 6.8).

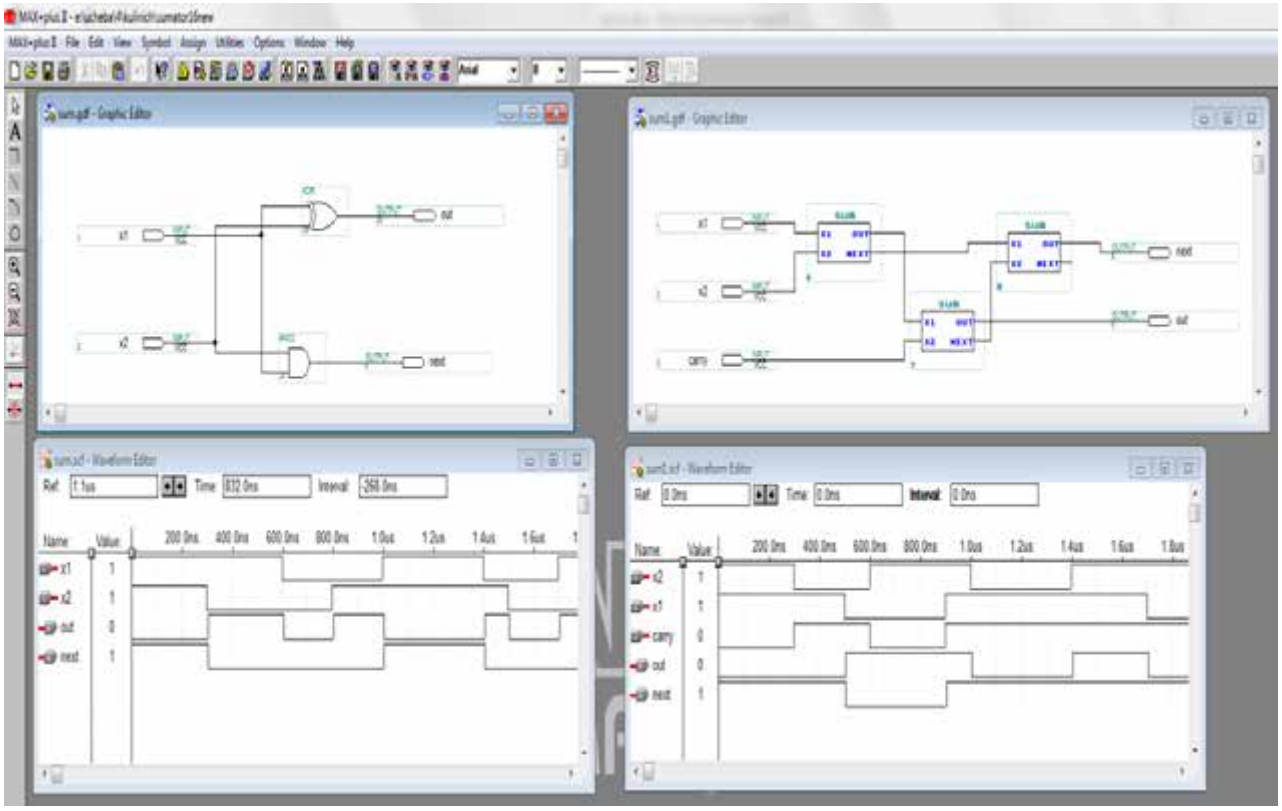


Fig. 6.8. Schematic diagrams of a half adder and a full adder.

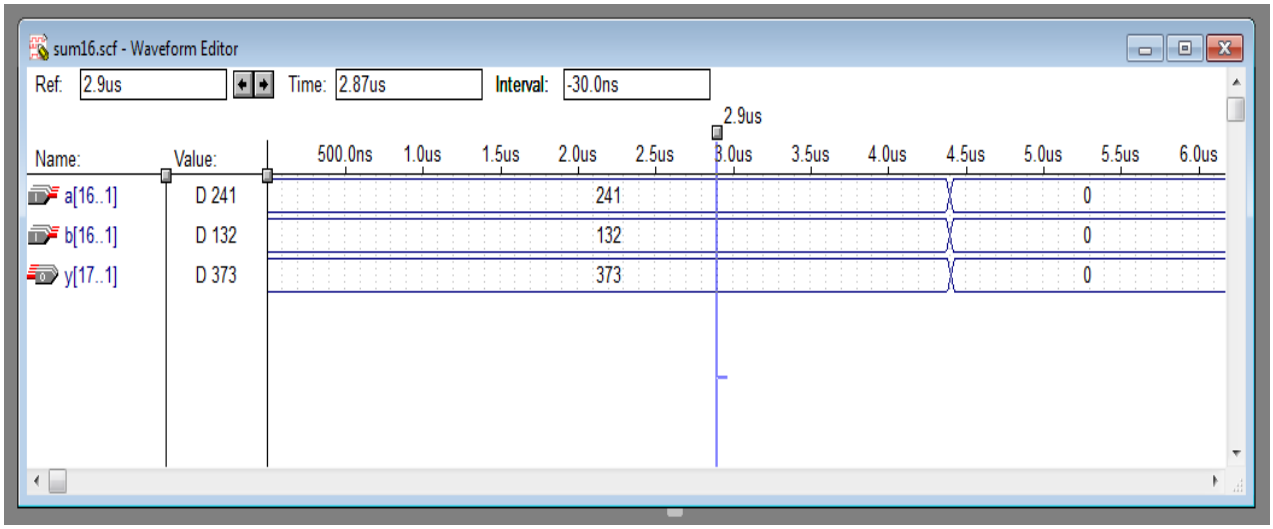


Fig. 6. Diagrams of the functioning of a 16-bit adder.

6.2. Multipliers.

Multiplication in the binary system is performed similarly to multiplication in the decimal system according to the given diagram (fig. 6.10):

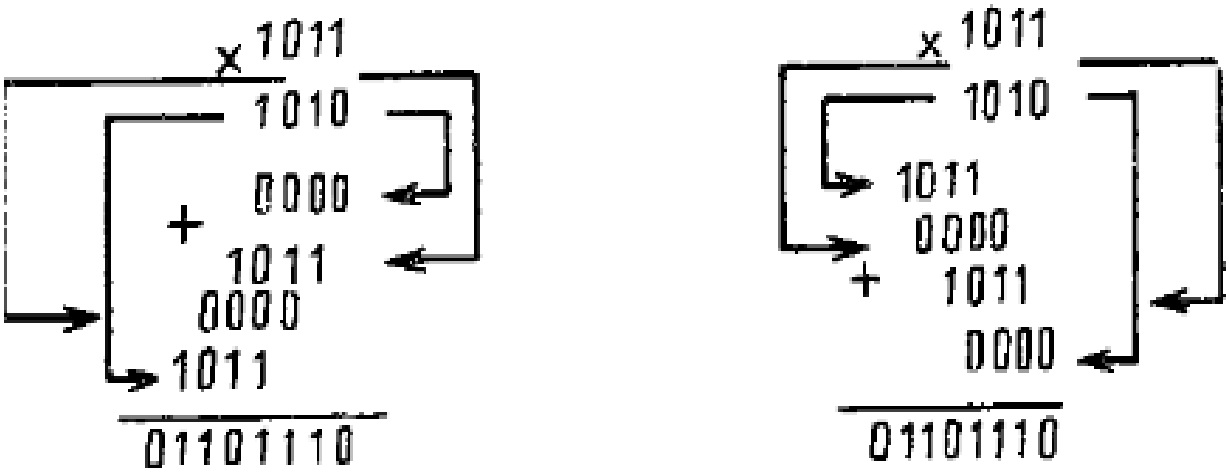


Fig. 6.10. The principle of binary multiplication.

Let us consider the structure of a multicycle multiplier that multiplies 24-bit binary numbers. When multiplying, it is necessary to form 4 lines of the partial sum. Strings are formed using the AND element. For example, X1Y0 means a logical AND between X1 and Y0. To form the multiplication, adders are also needed, and the device provides a shift of the partial sums relative to each other, as in fig. 6. 11.

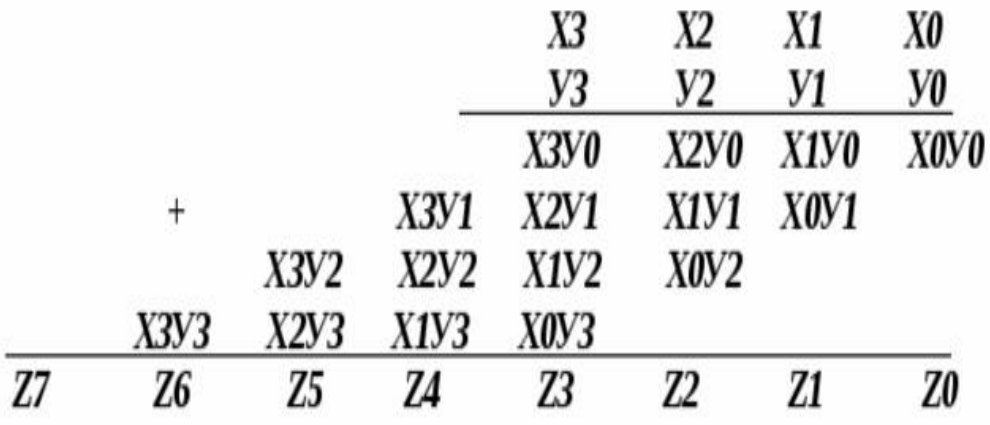


Fig. 6.11. A multiplier of 4-bit numbers.

To implement the above functions, you can apply the following two- stroke multiplier circuit:

As can be seen from the figure, the multi-clock multiplier consists of the shift register of the 1st multiplier, designed to store the second multiplier, a signal TACT 1, which shifts the shift register 1 by 1 digit to the right. Schemes AND are intended for the formation of lines of partial sums. Adders are designed for adding partial sums. The product shift register is designed to store multiplication and shift of partial sums by 1 digit to the right (fig. 6.12).

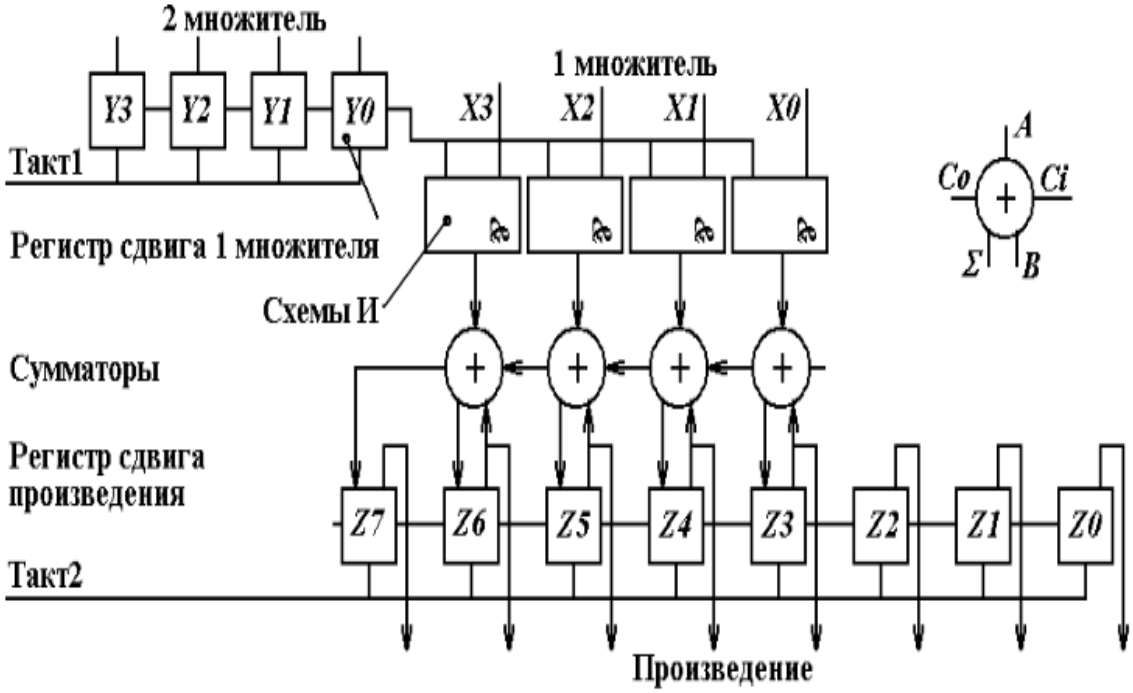


Fig. 6.12. Functional diagram of a multi-clock multiplier.

Implementation of the of 8-bit multiplier using CAD ALTERAMAX + MAX plus II 10.2 BASELINE

The implementation of a multiplier of 8-bit numbers in a graphical form is a rather cumbersome and complex task (which will be demonstrated later). It is much easier to program this task. It will take less time, effort and memory.

To begin with, it should be said that we provide two 8-bit numbers aa [7..0] and bb[7..0] as input, the clock frequency clk, and the bus code[3..0] with the help of which we will facilitate the task of writing a software implementation.

In variables, we set the module's internal variables intended for use in the logic description section. We will work with D- triggers (*DFF* is a synchronous D-flip-flop): 2 registers on the D-flip-flops, where we actually write our 8-bit numbers bit by bit, and 3 registers for writing intermediate results (intermediate multiplications). Figure 6.13.

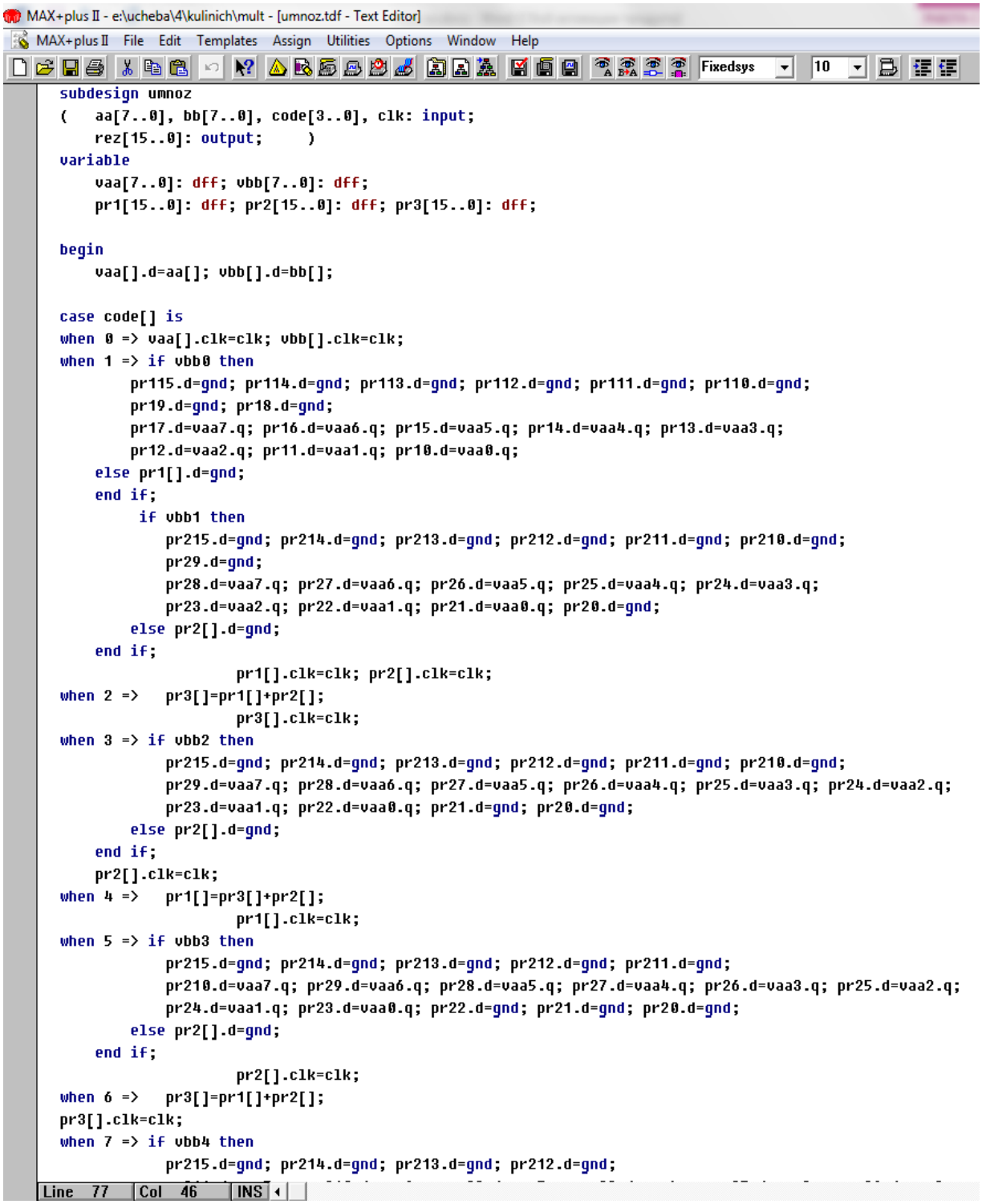


Fig. 6.13. Implementation of the multiplier in the AHDL language.

We sequentially describe all our actions using the *case operator*. The global clock frequency clk is given to the input. Next, using the logical if operator and each digit of the 2nd multiplier, starting with the smallest one, we write the result (bitwise) into the D-trigger.

When we get 2 intermediate results, we sum them up and rewrite them in another trigger. Thus, 3 additional D -triggers are enough to avoid confusion and overload the program. We continue to do the same with each digit of the 2nd multiplier. We add up all the intermediate results and get the result (fig. 6.14). We can see the correct operation of the program on the function diagrams (fig. 6.15).

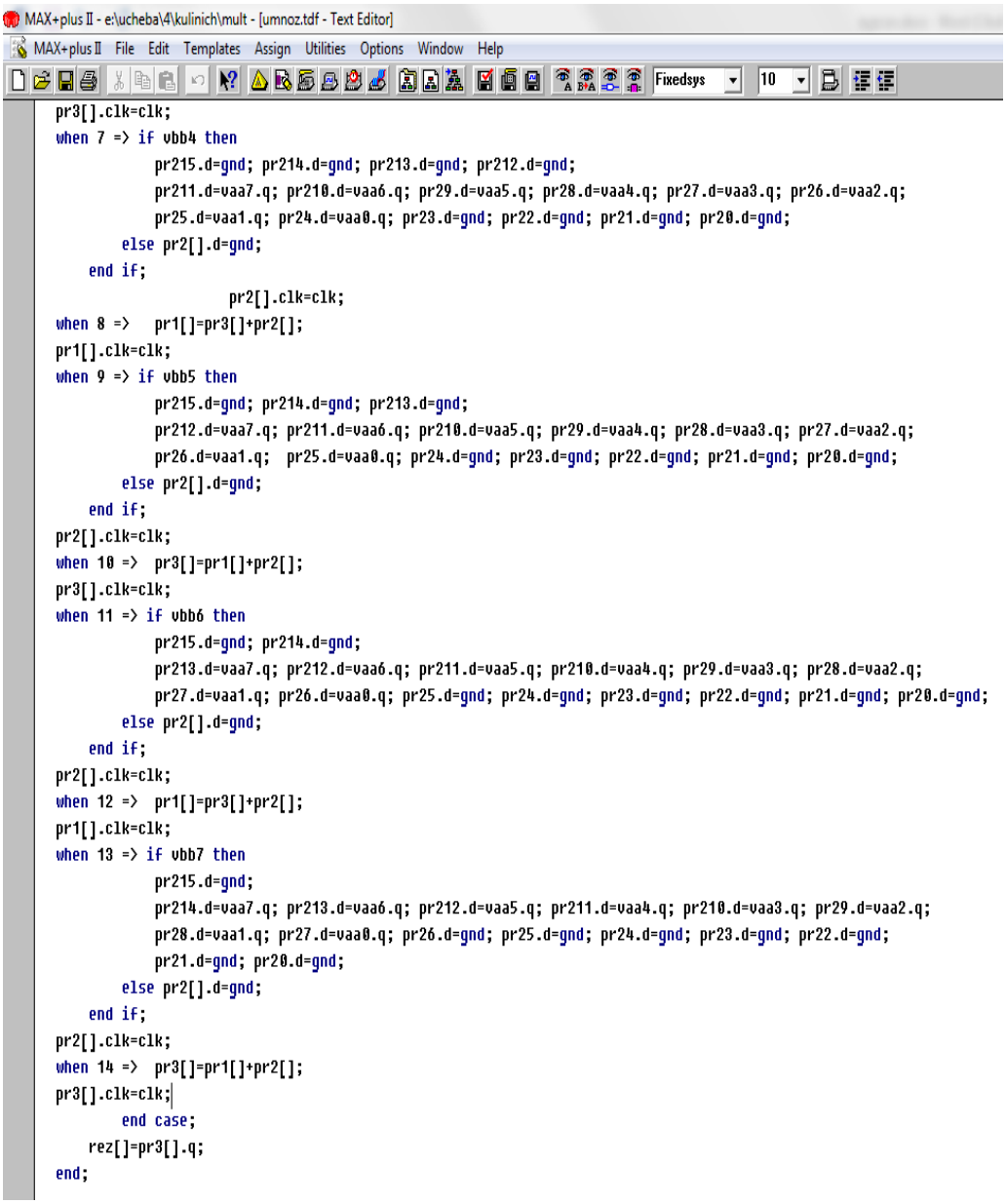


Fig. 6.14. Implementation of the multiplier in the AHDL language .

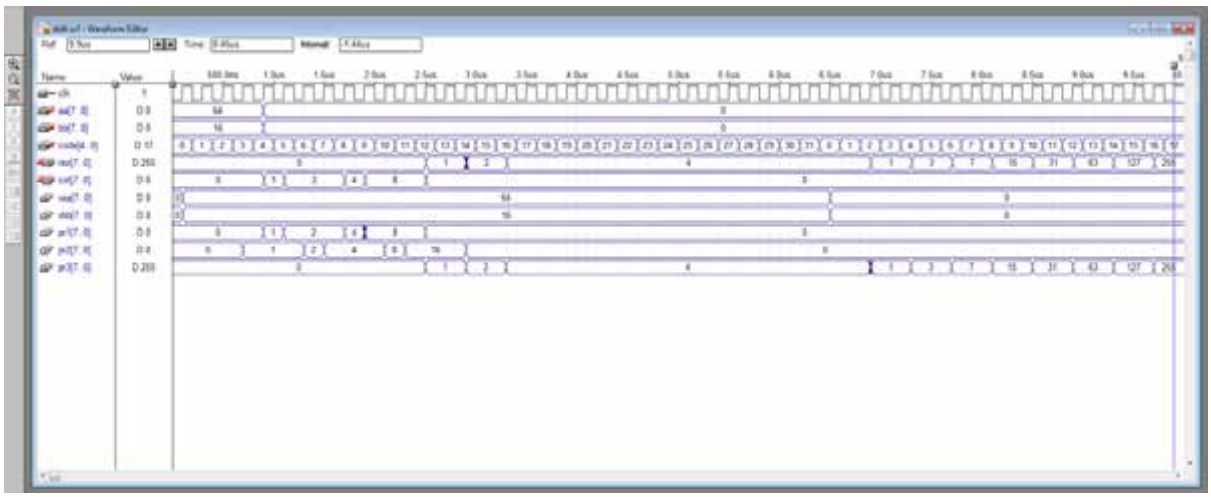


Fig. 6.15. Diagrams of the operation of the multiplier.

Implementation of the multiplier of 8-bit numbers using a graphic editor

Another option for constructing a multiplier is much more cumbersome and based on the use of a graphic editor. For example, the multiplication of 2 16-bit numbers will be implemented. Figure 4.16 shows how the project looks like as a whole. It consists of bufer 1616255, mnsum 1616, mnsum 16161, bufer 162, bufer 1627. But this is only the beginning, each of these blocks has a number of sub-blocks.

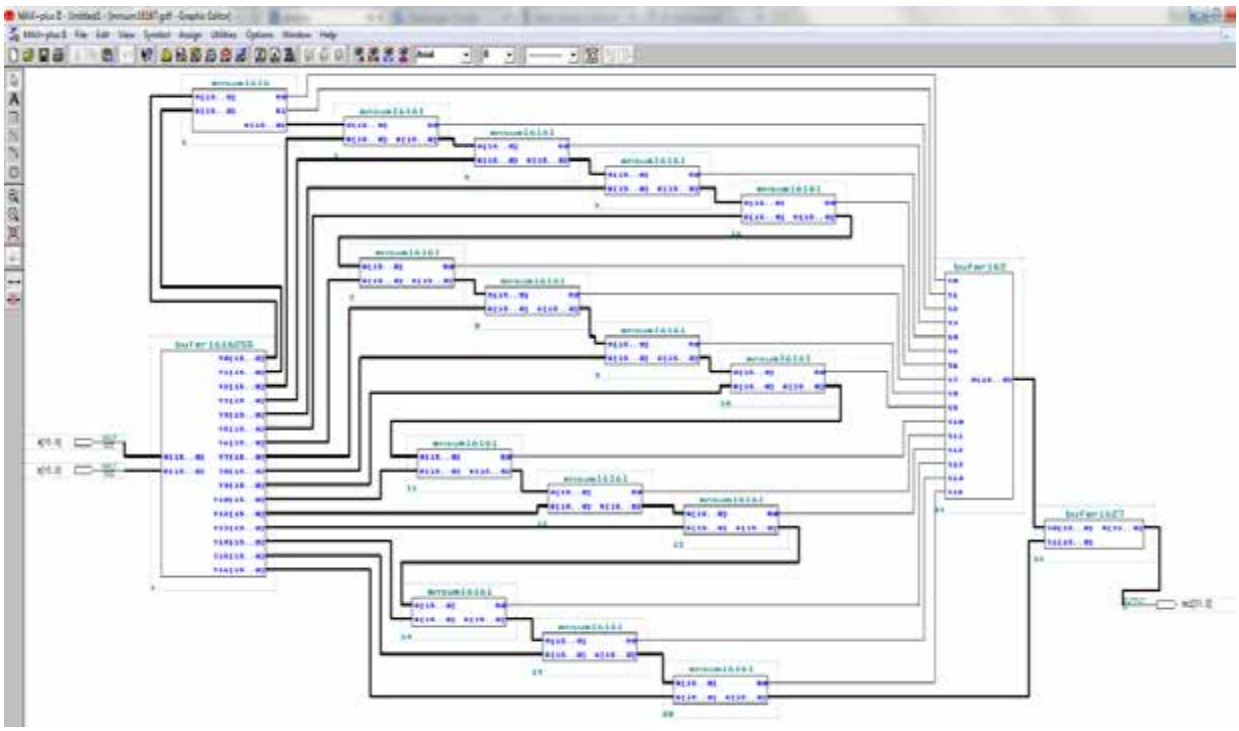


Fig. 6.16. Multiplier block diagram.

It is shown below that bufer 1616255 consists of bufer 1614 and bufer 163. They are designed:bufer 1614 - for bitwise distribution to 16 different outputs (the same function as in the adder above) and written as a program, bufer 163 - it demonstrates multiplication of 1cell of the 2nd multiplier by a whole row and also contains buffer 161 and buffer 162 (fig. 6.17, 6.18).

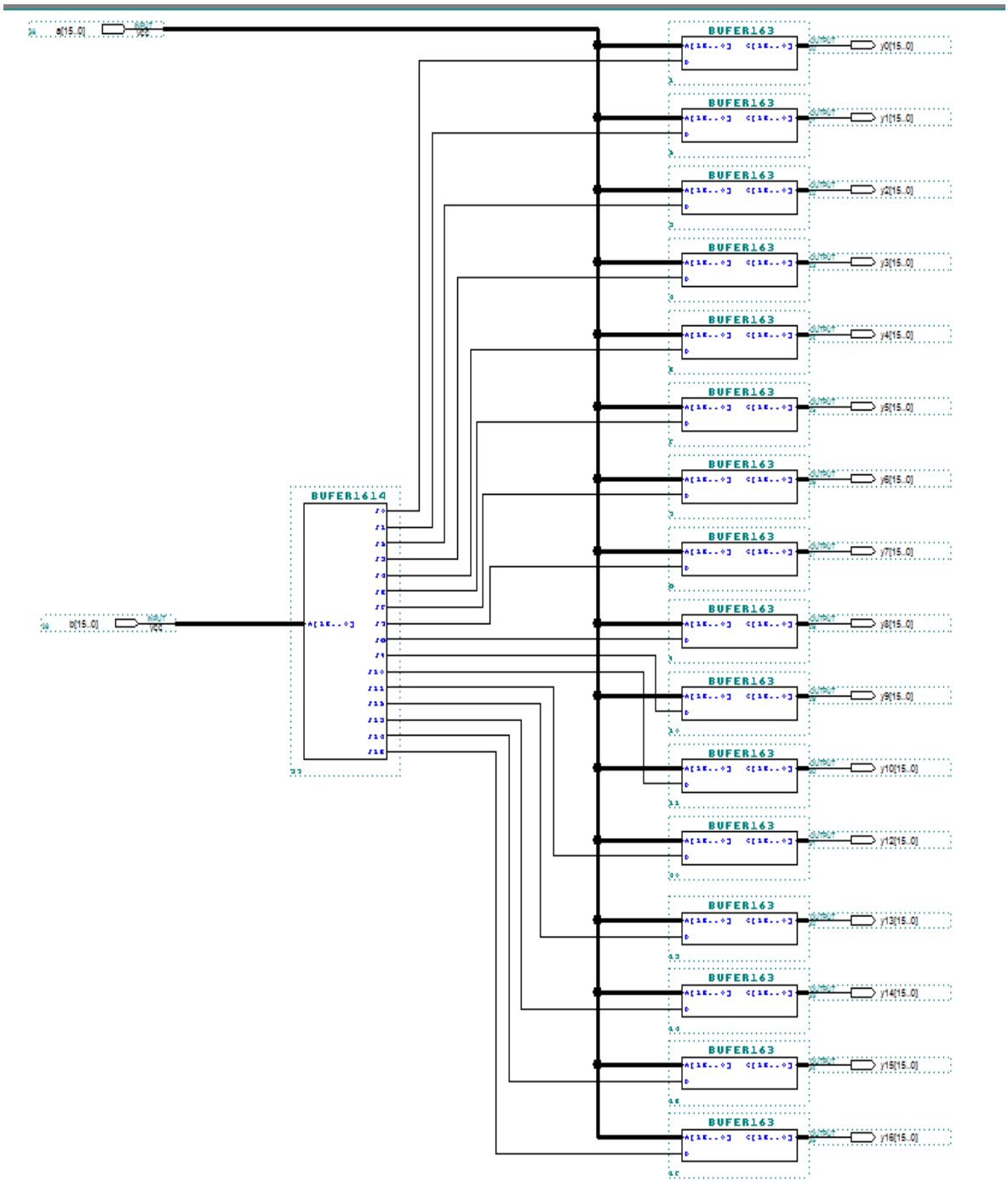


Fig. 6.17. Functional diagram of the buffer.

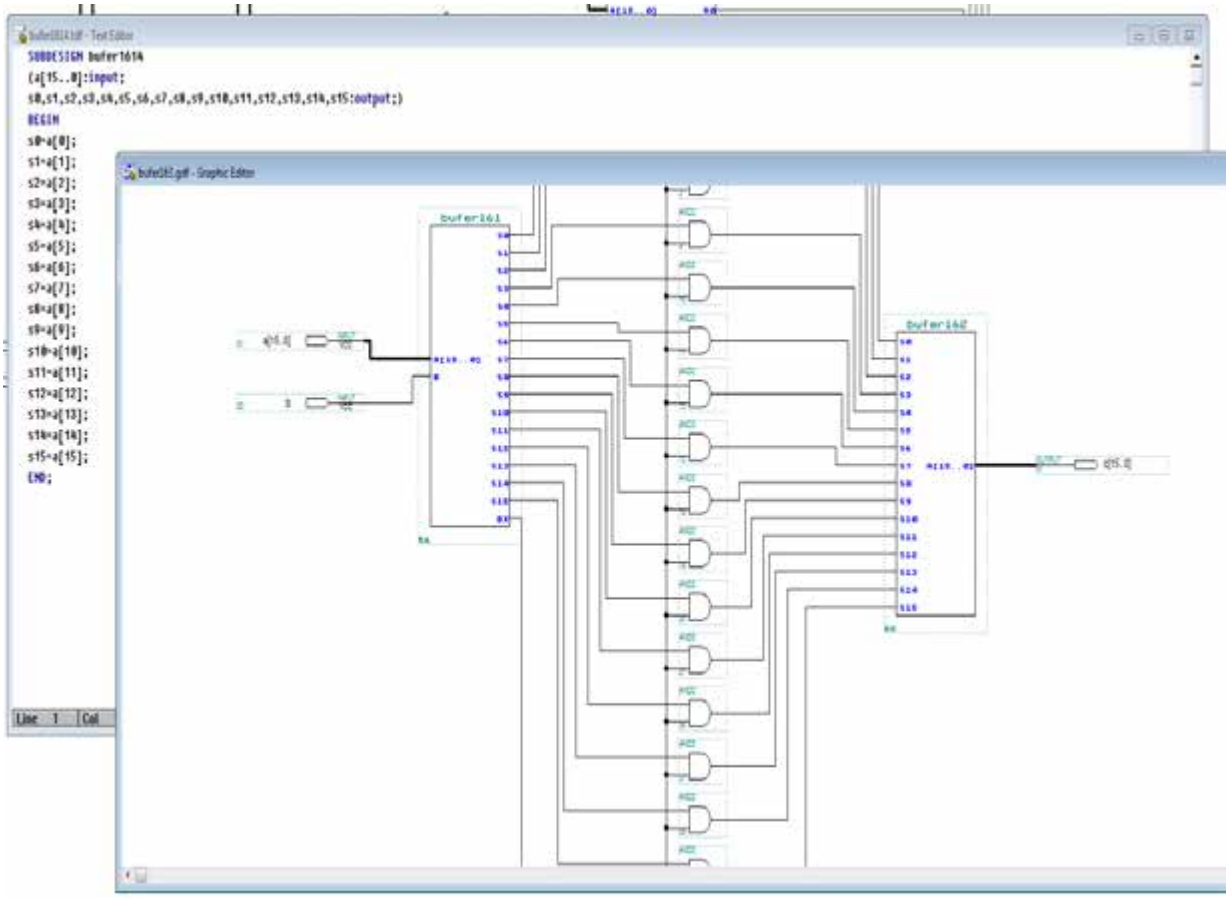


Fig. 6.18. Functional diagram of the buffer.

Buffer 161 and buffer 162 act as input and output buffers (fig. 6.19) (splitting and merging).

bufer161.tdf - Text Editor

```
SUBDESIGN bufer161
(a[15..0], b:input;
s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15,bx:output;)
BEGIN
s0=a[0];
s1=a[1];
s2=a[2];
s3=a[3];
s4=a[4];
s5=a[5];
s6=a[6];
s7=a[7];
s8=a[8];
s9=a[9];
s10=a[10];
s11=a[11];
s12=a[12];
s13=a[13];
s14=a[14];
s15=a[15];
bx=b;
END;
```

bufer162.tdf - Text Editor

```
SUBDESIGN bufer162
( s0,s1,s2,s3,s4,s5,s6,s7,s8,s9,s10,s11,s12,s13,s14,s15:input;
a[15..0]:output;)
BEGIN
a[0]=s0;
a[1]=s1;
a[2]=s2;
a[3]=s3;
a[4]=s4;
a[5]=s5;
a[6]=s6;
a[7]=s7;
a[8]=s8;
a[9]=s9;
a[10]=s10;
a[11]=s11;
a[12]=s12;
a[13]=s13;
a[14]=s14;
a[15]=s15;
END;
```

Fig. 6.19. Splitting and merging buffers.

mnsum 1616 and mnsum 16161 adds intermediate results. They are built using half-adders and adders (fig. 6.20).

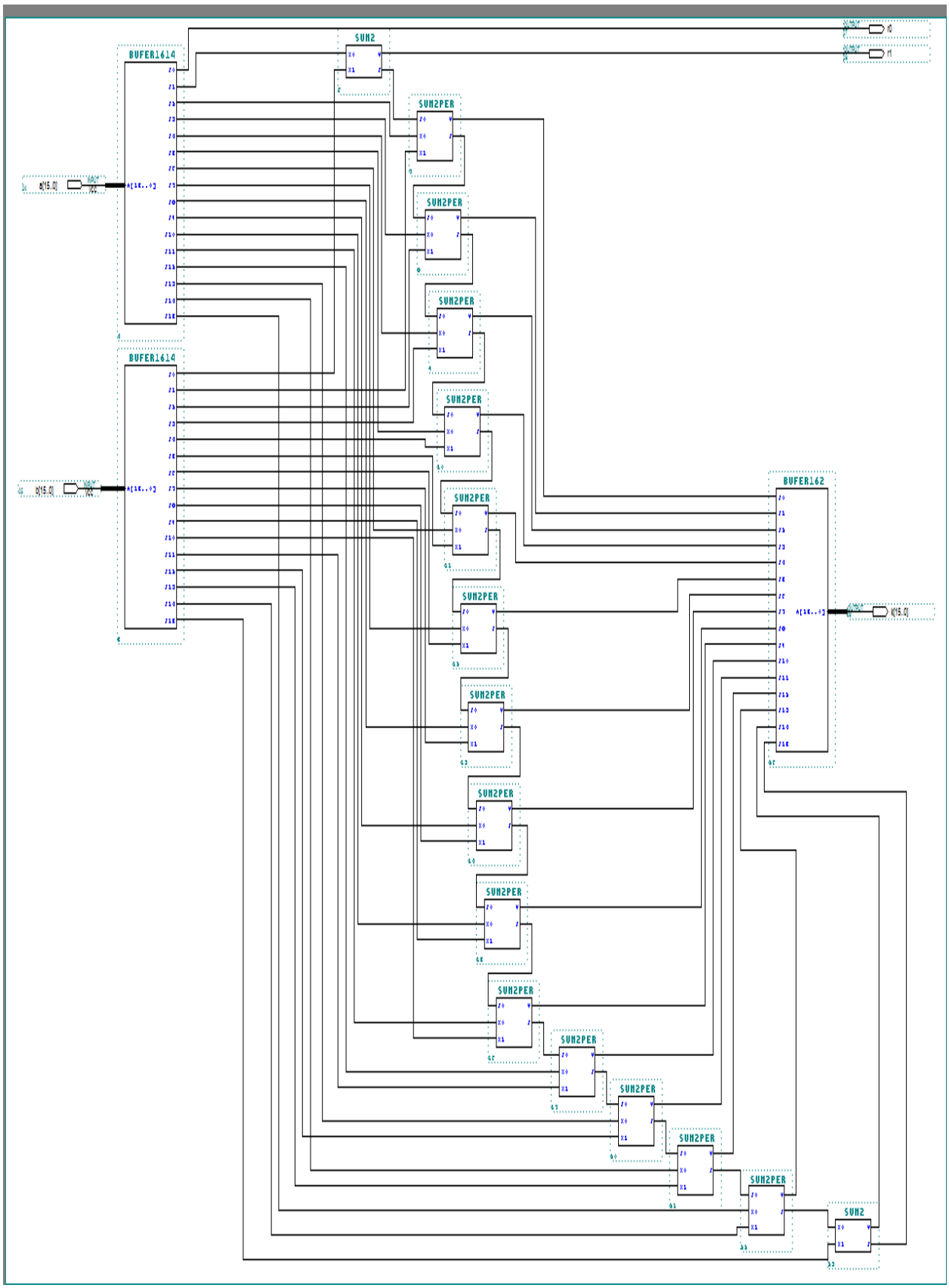


Fig. 6.20. Block for obtaining intermediate results.

Buffer 1627 structure (fig. 6.21) receives a 32- bit number as a result of multiplication.

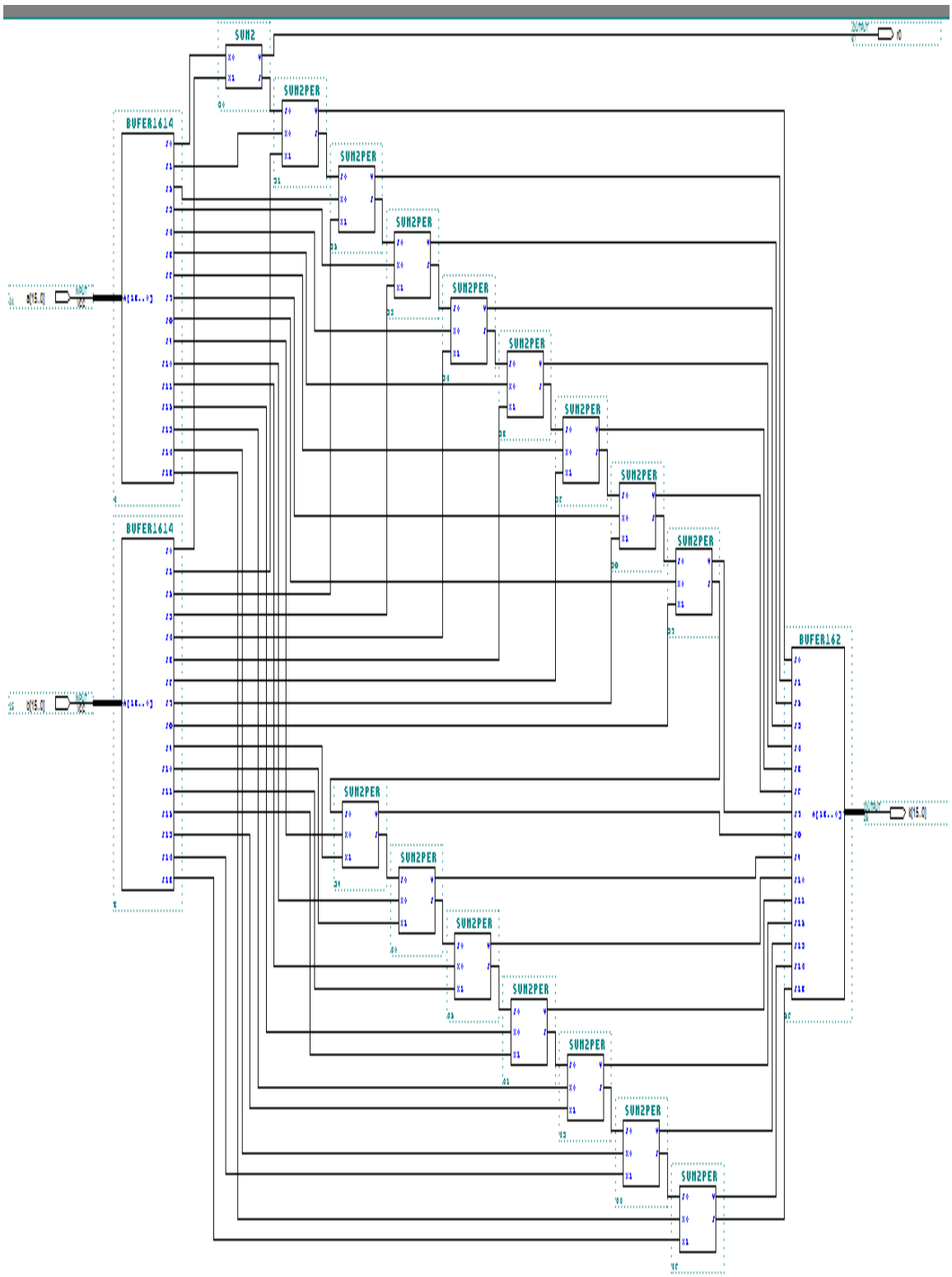


Fig. 6.21. Block diagram of the device.

6.3. Number divisors.

Multiplication and division devices play a major role in digital information processing devices. Most often there are tasks in which it is necessary to multiply the value of any quantity and divide it by various coefficients, which can also be variables. In general, the division device can be presented in the form of a functional diagram (fig. 6.22).

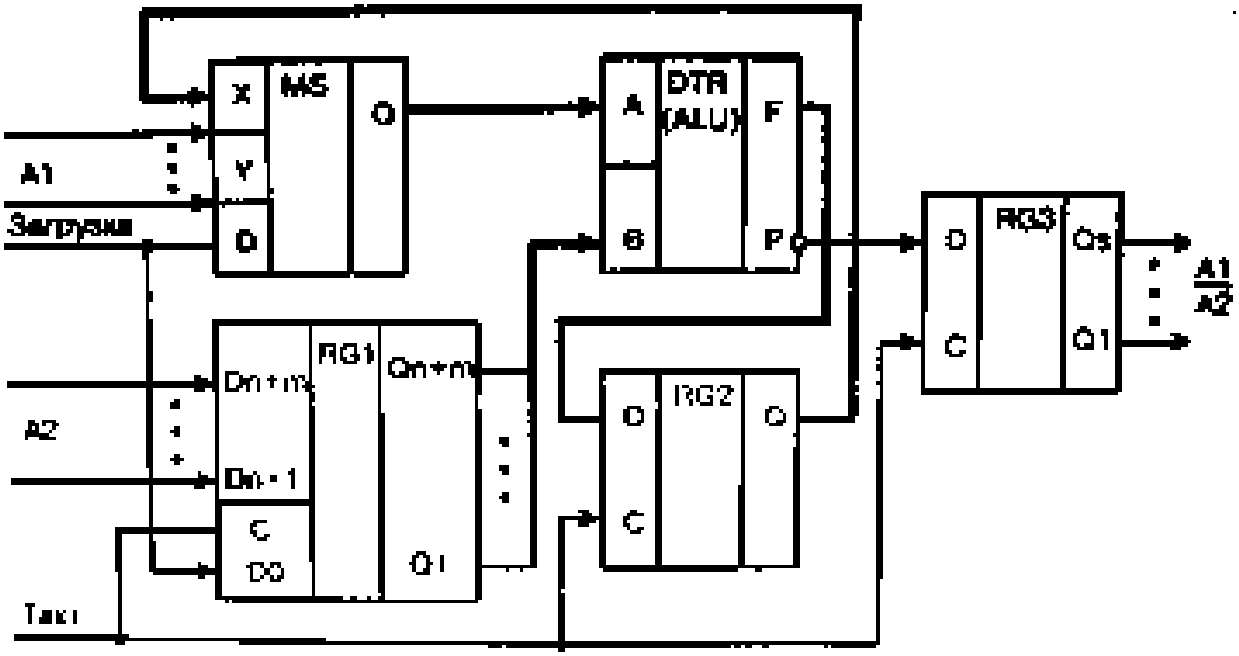


Fig. 6.22. Functional diagram of the division device

Construction of division devices is more difficult than multiplication devices. Let us consider an example of dividing numbers in binary codes: divide the number 56 (binary code 111000) into 8 (binary code 1000).

111000	1000
1000	111
1100	
1000	
1000	
1000	
0000	

As you can see, the result was the number 111 (that is, 7). The division algorithm looks like this: a divisor is subtracted from the higher digits of the dividend, if the difference is greater than zero, "0" is entered in the result, the divisor is shifted down by one digit (if the difference is less than zero, "0" is entered in the

result). In principle (unlike multiplication), this process can continue indefinitely (remember infinite fractions).

The functional diagram of the division device is shown in fig. 6.22. Its main element is the subtracting device DTR. The MS multiplexer is a switch from two inputs to one output. When the "Load" pulse is applied, the multiplexer MS is switched to the position when divided A1 is received at its output, therefore the divided is received at input A of the subtracting device DTR (in the rest of the time, the difference from the storage register RG2 is received at the output of the multiplexer MS). At the same time, a divisor A2 is recorded in the register RG1 according to the most significant bits, which enters the inputs B of the subtracting device DTR.

A difference is formed at the output F of the DTR subtractor. If this difference is greater than zero, then there is a "1" at the P output, which is entered into the RG3 register by a clock pulse. If there is an overflow (the difference is less than zero), then there is a "0" at the P output, which is entered into the RG3 register. With each operating cycle, the divider A2 is sequentially shifted down by one digit in the RG1 register, and the result is shifted up by one digit in the RG3 register. The number of operating cycles is determined by the bit rate of the result, but it should be remembered that the bit rate of the registers RG1, RG2 and the subtracting device DTR is determined as the sum of the number of bits of the divided and the number of bits of the result.

Implementation of an 8-bit divisor of binary numbers using CAD ALTERA MAX + plusII 10.2 BASELINE

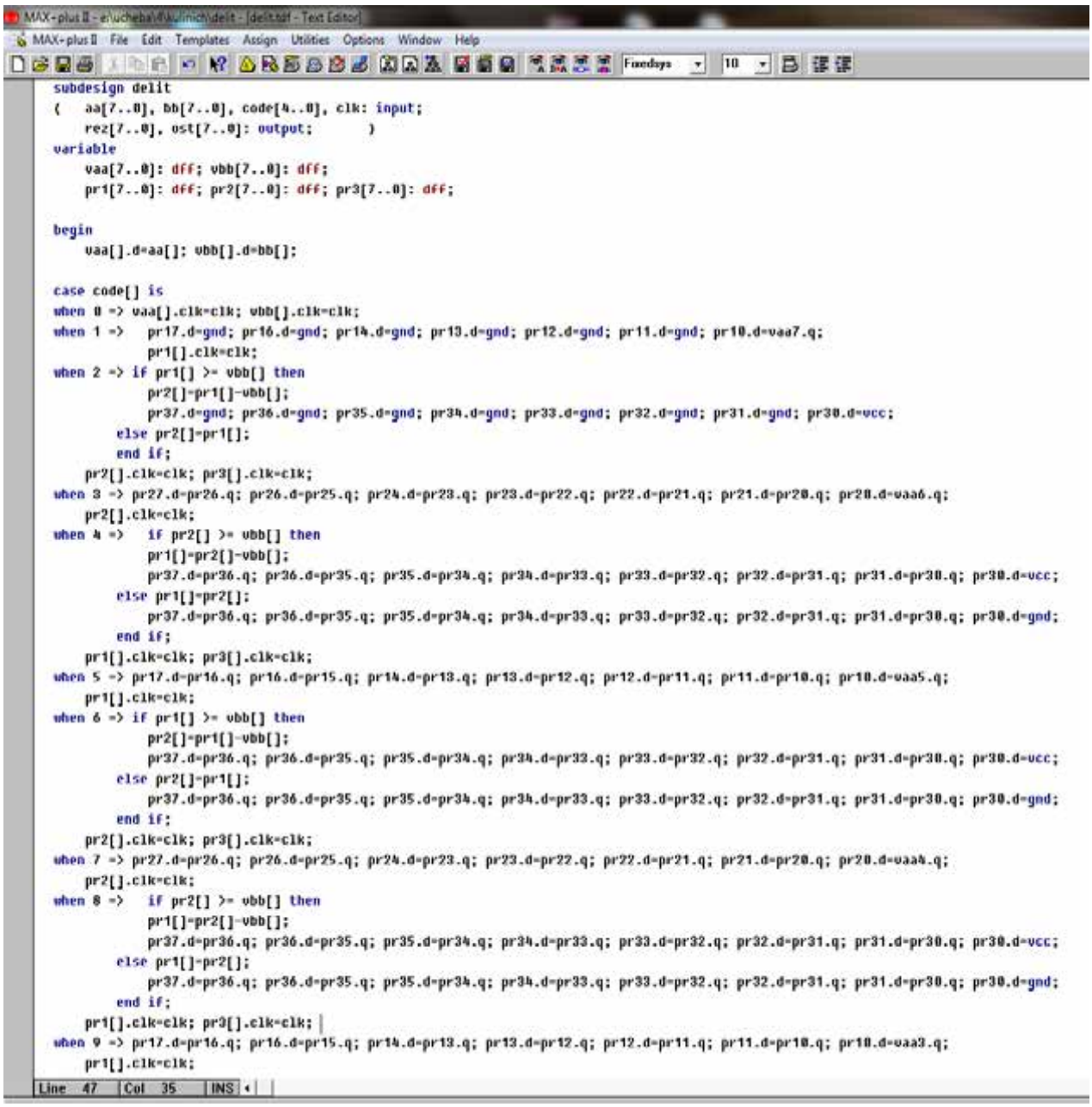
The implementation of the 8-bit number divider is implemented in the Text Editor File. To begin with, it is worth saying that we provide two 8-bit numbers aa [7..0] and bb[7..0] at the input, the clock frequency clk, and the bus code[4..0], with which we will facilitate the task of writing a software implementation.

We specify the module's internal variables intended for use in the logic description section in the variable section. We will use D- triggers (*DFF* is a synchronous D-trigger), on the basis of which registers are built: 2 D-triggers where we actually write our 8-bit numbers bit by bit, and 3 triggers to write intermediate results (intermediate multiplications). At the end of the program, we will receive 2 values: this is our result rez [7..0] and the remainder of the division ost[7..0].

The program uses such designations as *gnd and vcc*:

- gnd is ground - the point of zero potential of the microcircuit;
- vcc is the "+" power supply of the microcircuit relative to gnd.

Using the case operator, we sequentially describe all our actions. Three additional registers are enough not to get confused and overload the program. We can see the correct operation of the program on the functioning diagrams (fig. 6.23, 6.24, 64.25).




```
subdesign delit
(   aa[7..0], bb[7..0], code[4..0], clk: input;
    rez[7..0], ost[7..0]: output;       )
variable
    vaa[7..0]: dff; vbb[7..0]: dff;
    pr1[7..0]: dff; pr2[7..0]: dff; pr3[7..0]: dff;

begin
    vaa[].d=aa[]; vbb[].d=bb[];

case code[] is
when 0 => vaa[].clk=clk; vbb[].clk=clk;
when 1 =>   pr17.d=gnd; pr16.d=gnd; pr14.d=gnd; pr13.d=gnd; pr12.d=gnd; pr11.d=gnd; pr10.d=vaa7.q;
            pr1[].clk=clk;
when 2 => if pr1[] >= vbb[] then
            pr2[]=pr1[]-vbb[];
            pr37.d=gnd; pr36.d=gnd; pr35.d=gnd; pr34.d=gnd; pr33.d=gnd; pr32.d=gnd; pr31.d=gnd; pr30.d=vcc;
        else pr2[]=pr1[];
        end if;
    pr2[].clk=clk; pr3[].clk=clk;
when 3 => pr27.d=pr26.q; pr26.d=pr25.q; pr24.d=pr23.q; pr23.d=pr22.q; pr22.d=pr21.q; pr21.d=pr20.q; pr20.d=vaa6.q;
    pr2[].clk=clk;
when 4 =>   if pr2[] >= vbb[] then
            pr1[]=pr2[]-vbb[];
            pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q; pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=vcc;
        else pr1[]=pr2[];
            pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q; pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=gnd;
        end if;
    pr1[].clk=clk; pr3[].clk=clk;
when 5 => pr17.d=pr16.q; pr16.d=pr15.q; pr14.d=pr13.q; pr13.d=pr12.q; pr12.d=pr11.q; pr11.d=pr10.q; pr10.d=vaa5.q;
    pr1[].clk=clk;
when 6 => if pr1[] >= vbb[] then
            pr2[]=pr1[]-vbb[];
            pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q; pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=vcc;
        else pr2[]=pr1[];
            pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q; pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=gnd;
        end if;
    pr2[].clk=clk; pr3[].clk=clk;
when 7 => pr27.d=pr26.q; pr26.d=pr25.q; pr24.d=pr23.q; pr23.d=pr22.q; pr22.d=pr21.q; pr21.d=pr20.q; pr20.d=vaa4.q;
    pr2[].clk=clk;
when 8 =>   if pr2[] >= vbb[] then
            pr1[]=pr2[]-vbb[];
            pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q; pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=vcc;
        else pr1[]=pr2[];
            pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q; pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=gnd;
        end if;
    pr1[].clk=clk; pr3[].clk=clk;
when 9 => pr17.d=pr16.q; pr16.d=pr15.q; pr14.d=pr13.q; pr13.d=pr12.q; pr12.d=pr11.q; pr11.d=pr10.q; pr10.d=vaa3.q;
    pr1[].clk=clk;
```

Fig. 6. 23. Number divisor

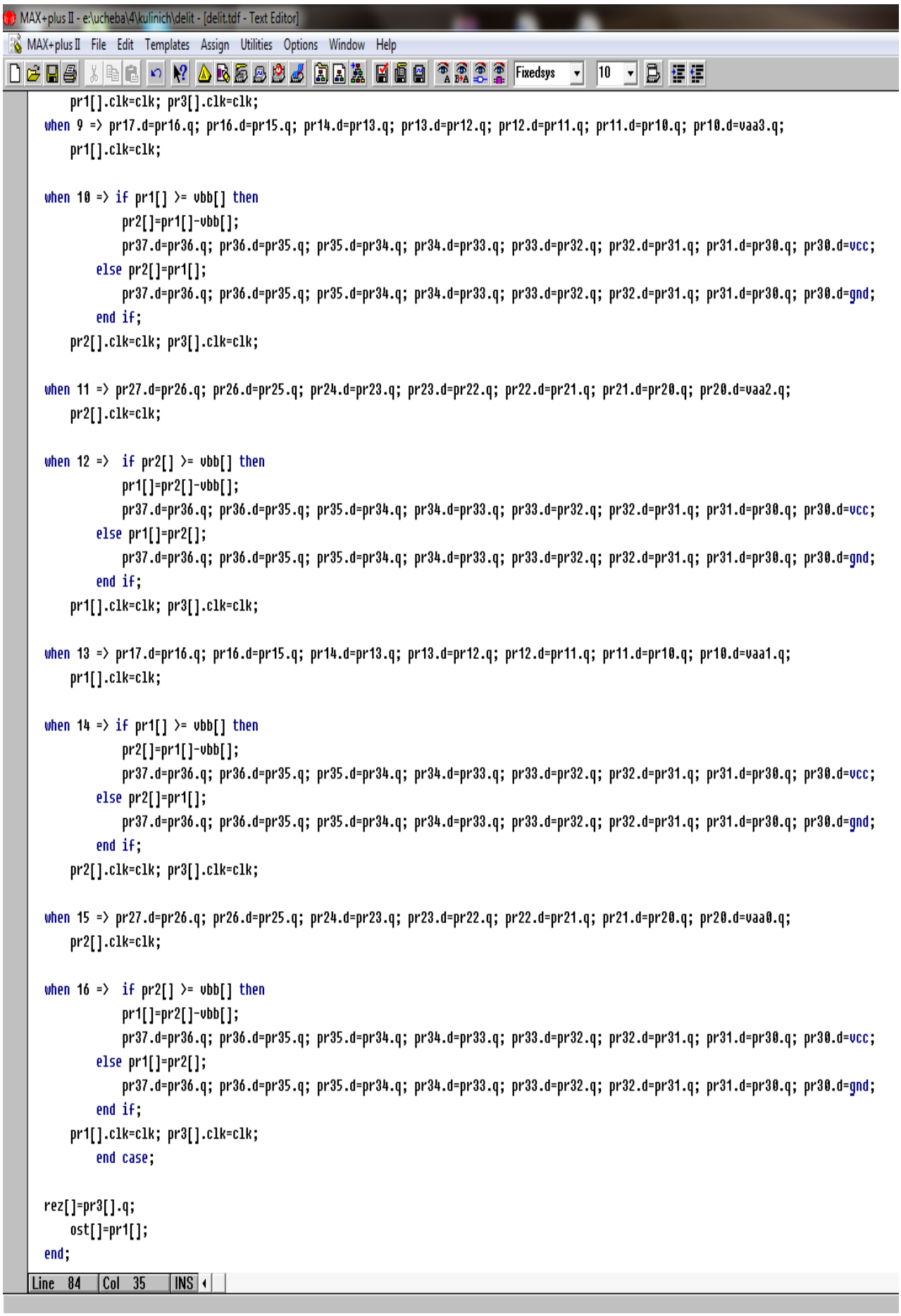


Fig. 6. 24. Number divisor (continued)

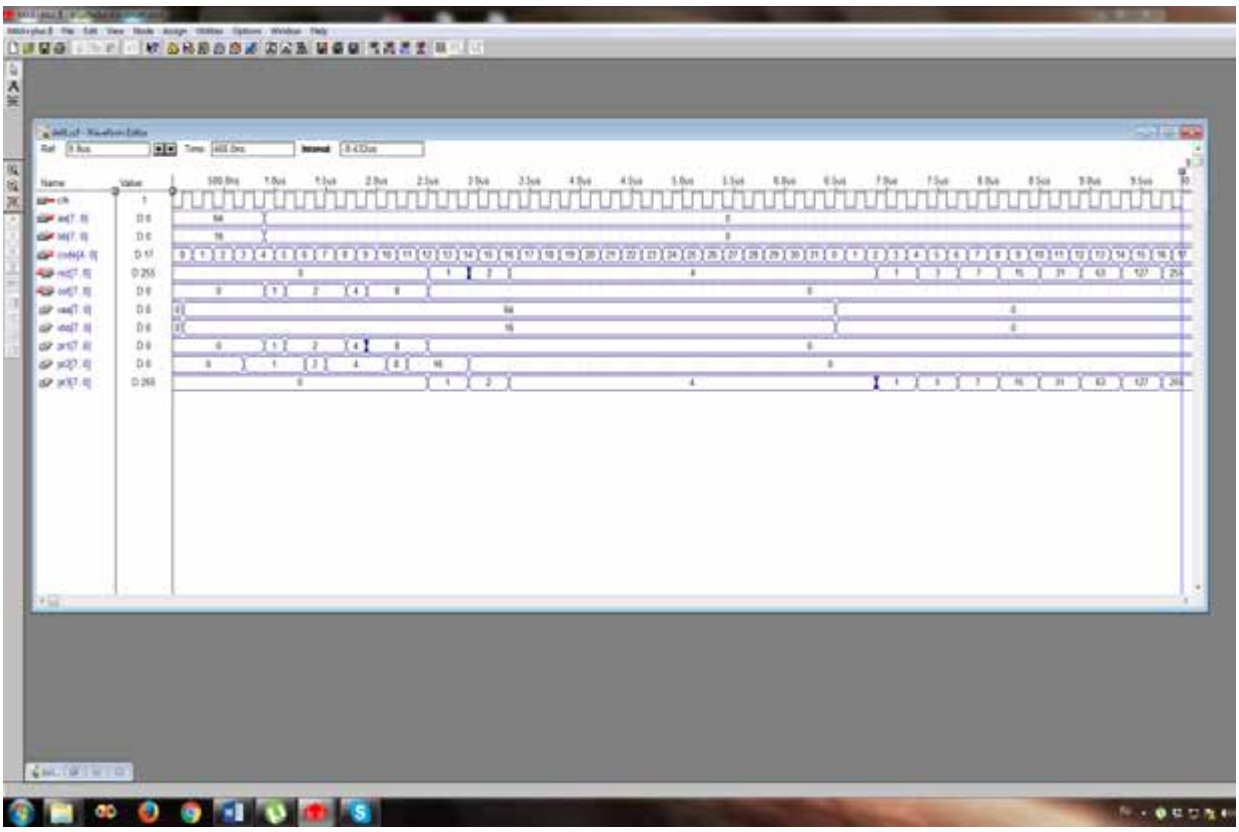


Fig. 6. 25. Diagrams of the operation of the number divider

The general listing of the program describing the operation of the device is given below.

```
subdesigndelit
(       aa [7..0], bb [7..0], code [4..0], clk : input ;
        rez [7..0] , ost[7..0]: output;                 )
variable
        vaa[7..0]: dff; vbb[7..0]: dff;
        pr1[7..0]: dff; pr2[7..0]: dff; pr3[7..0]: dff;
begin
        vaa[].d=aa[]; vbb[].d=bb[];
case code[] is
when 0 => vaa[].clk=clk; vbb[].clk=clk;
when 1 =>   pr17.d=gnd; pr16.d=gnd; pr14.d=gnd; pr13.d=gnd; pr12.d=gnd;
pr11.d=gnd; pr10.d=vaa7.q;
                        pr1[].clk=clk;
when 2 => if pr1[] >= vbb[] then
```

```
                    pr2[]=pr1[]-vbb[];
                    pr37.d=gnd; pr36.d=gnd; pr35.d=gnd; pr34.d=gnd; pr33.d=gnd;
pr32.d=gnd; pr31.d=gnd; pr30.d=vcc;
                else pr2[]=pr1[];
                end if;
        pr2[].clk=clk; pr3[].clk=clk;
when 3 => pr27.d=pr26.q; pr26.d=pr25.q; pr24.d=pr23.q; pr23.d=pr22.q;
pr22.d=pr21.q; pr21.d=pr20.q; pr20.d=vaa6.q;
        pr2[].clk=clk;
when 4 =>   if pr2[] >= vbb[] then
                    pr1[]=pr2[]-vbb[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=vcc;
                else pr1[]=pr2[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=gnd;
                end if;
        pr1[].clk=clk; pr3[].clk=clk;
when 5 => pr17.d=pr16.q; pr16.d=pr15.q; pr14.d=pr13.q; pr13.d=pr12.q;
pr12.d=pr11.q; pr11.d=pr10.q; pr10.d=vaa5.q;
        pr1[].clk=clk;
when 6 => if pr1[] >= vbb[] then
                    pr2[]=pr1[]-vbb[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=vcc;
                else pr2[]=pr1[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=gnd;
                end if;
        pr2[].clk=clk; pr3[].clk=clk;
when 7 => pr27.d=pr26.q; pr26.d=pr25.q; pr24.d=pr23.q; pr23.d=pr22.q;
pr22.d=pr21.q; pr21.d=pr20.q; pr20.d=vaa4.q;
```

```
        pr2[].clk=clk;
when 8 =>   if pr2[] >= vbb[] then
                    pr1[]=pr2[]-vbb[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=vcc;
            else pr1[]=pr2[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=gnd;
            end if;
        pr1[].clk=clk; pr3[].clk=clk;
when 9 => pr17.d=pr16.q; pr16.d=pr15.q; pr14.d=pr13.q; pr13.d=pr12.q;
pr12.d=pr11.q; pr11.d=pr10.q; pr10.d=vaa3.q;
        pr1[].clk=clk;

when 10 => if pr1[] >= vbb[] then
                    pr2[]=pr1[]-vbb[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=vcc;
            else pr2[]=pr1[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=gnd;
            end if;
        pr2[].clk=clk; pr3[].clk=clk;
when 11 => pr27.d=pr26.q; pr26.d=pr25.q; pr24.d=pr23.q; pr23.d=pr22.q;
pr22.d=pr21.q; pr21.d=pr20.q; pr20.d=vaa2.q;
        pr2[].clk=clk;

when 12 => if pr2[] >= vbb[] then
                    pr1[]=pr2[]-vbb[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=vcc;
            else pr1[]=pr2[];
```

```
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=gnd;
                end if;
        pr1[].clk=clk; pr3[].clk=clk;
when 13 => pr17.d=pr16.q; pr16.d=pr15.q; pr14.d=pr13.q; pr13.d=pr12.q;
pr12.d=pr11.q; pr11.d=pr10.q; pr10.d=vaa1.q;
        pr1[].clk=clk;
when 14 => if pr1[] >= vbb[] then
                    pr2[]=pr1[]-vbb[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=vcc;
                else pr2[]=pr1[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=gnd;
                end if;
        pr2[].clk=clk; pr3[].clk=clk;
when 15 => pr27.d=pr26.q; pr26.d=pr25.q; pr24.d=pr23.q; pr23.d=pr22.q;
pr22.d=pr21.q; pr21.d=pr20.q; pr20.d=vaa0.q;
        pr2[].clk=clk;
when 16 => if pr2[] >= vbb[] then
                    pr1[]=pr2[]-vbb[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=vcc;
                else pr1[]=pr2[];
                    pr37.d=pr36.q; pr36.d=pr35.q; pr35.d=pr34.q; pr34.d=pr33.q;
pr33.d=pr32.q; pr32.d=pr31.q; pr31.d=pr30.q; pr30.d=gnd;
                end if;
        pr1[].clk=clk; pr3[].clk=clk;
                end case;
rez[]=pr3[].q;
        ost[]=pr1[];
end;
```

6.4. Linear shift register.

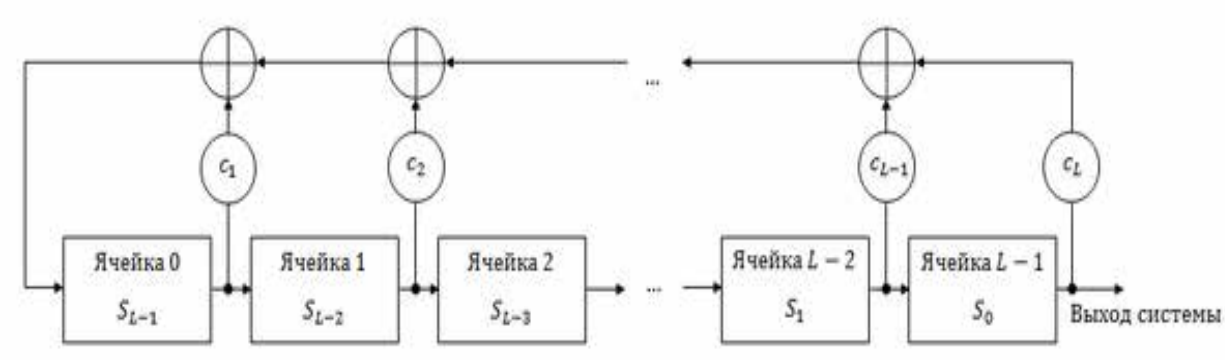
A linear feedback shift register (LFSR) is a bitword shift register in which the value of the input bit is equal to a linear Boolean function of the values of the other bits of the register before the shift. It can be organized by both software and hardware. It is used to generate pseudo-random sequences of bits, which is used, in particular, in cryptography.

Two parts (modules) are distinguished in the shift register with linear feedback:

- shift register itself;
- feedback scheme (or subroutine).

The register consists of functional memory cells, each of which stores the current state (value) of one bit. The number of cells L is called the length of the register. Bits (cells) are usually numbered with numbers i=0, 1, 2, ..., L -1. The feedback function for the LFSR is a linear Boolean function of the values of all or several bits of the register. The function multiplies register bits by coefficients c_i, where i=0, 1, 2, ..., L.

During each clock cycle a linear feedback shift register (fig. 6.26) performs



the following operations:

Figure 6.26. The structure of the linear recurrent register

- the bit located in cell L -1 is read; this bit is the next bit of the outgoing sequence;
- the feedback function calculates a new value for cell 0 using the current cell

values;

- the content of each i-th cell is moved to the next i+1 cell, where i=0, 1, 2, ..., L - 2;
- the bit previously calculated by the feedback function is written in cell 0.

Implementation of LRR (linear recurrent register) using CAD ALTERA MAX + plusII 10.2

The implementation of a linear recurrent register can be carried out both in a Text Editor File and in a graphic editor. The clock frequency clk, the initial filling kl and the control signal upr are supplied at the input. A recurrent sequence is formed using feedback os. The register itself is built on the basis of D -triggers. The principle of formation of the output sequence is depicted on the function diagrams (fig. 6.27).

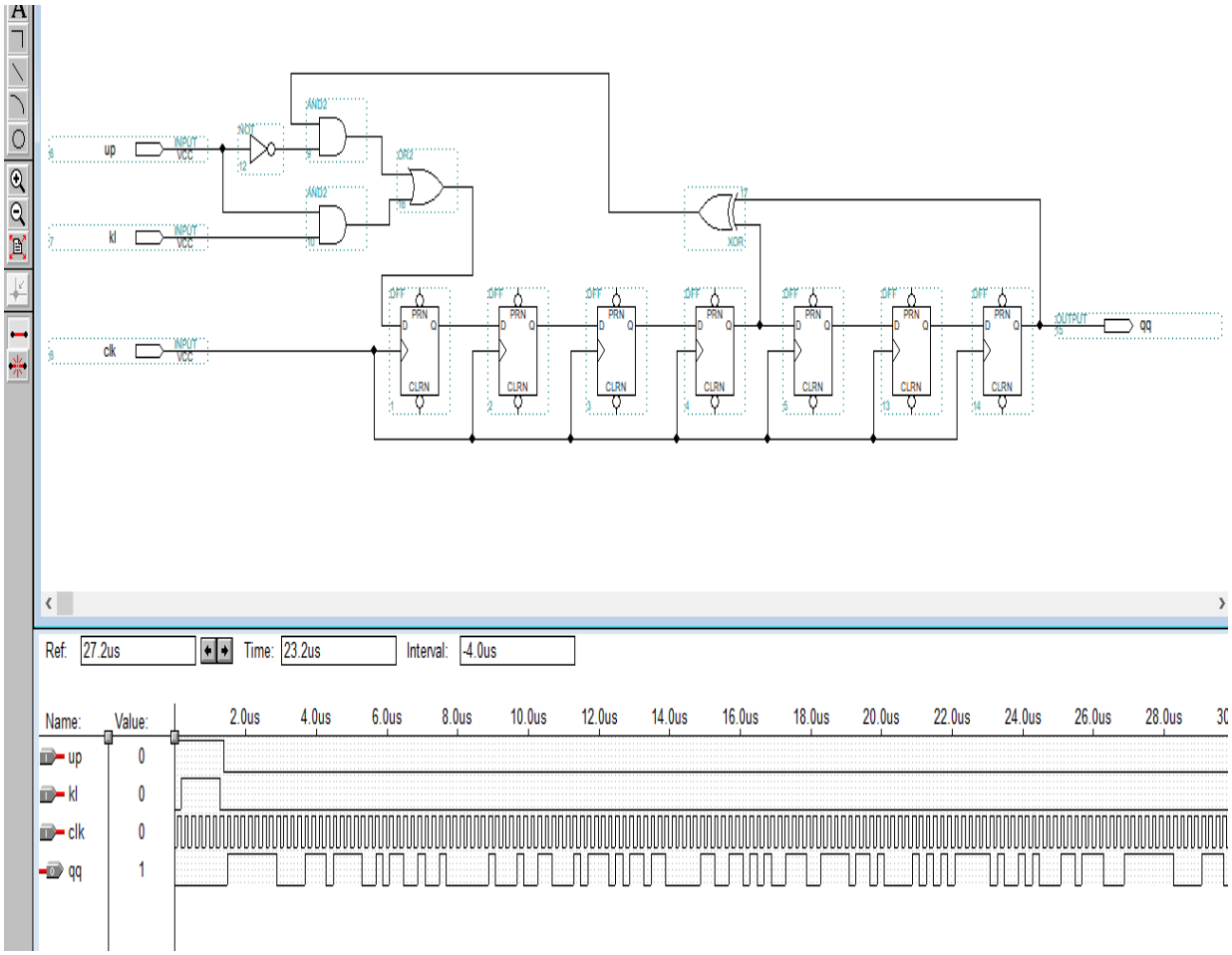


Fig. 6.27. Diagram of LRR and its functioning.

The LRR diagram is made using two modules that are combined through the INCLUDE operator in *the MAX + plusII* CAD text editor (fig. 6.28).

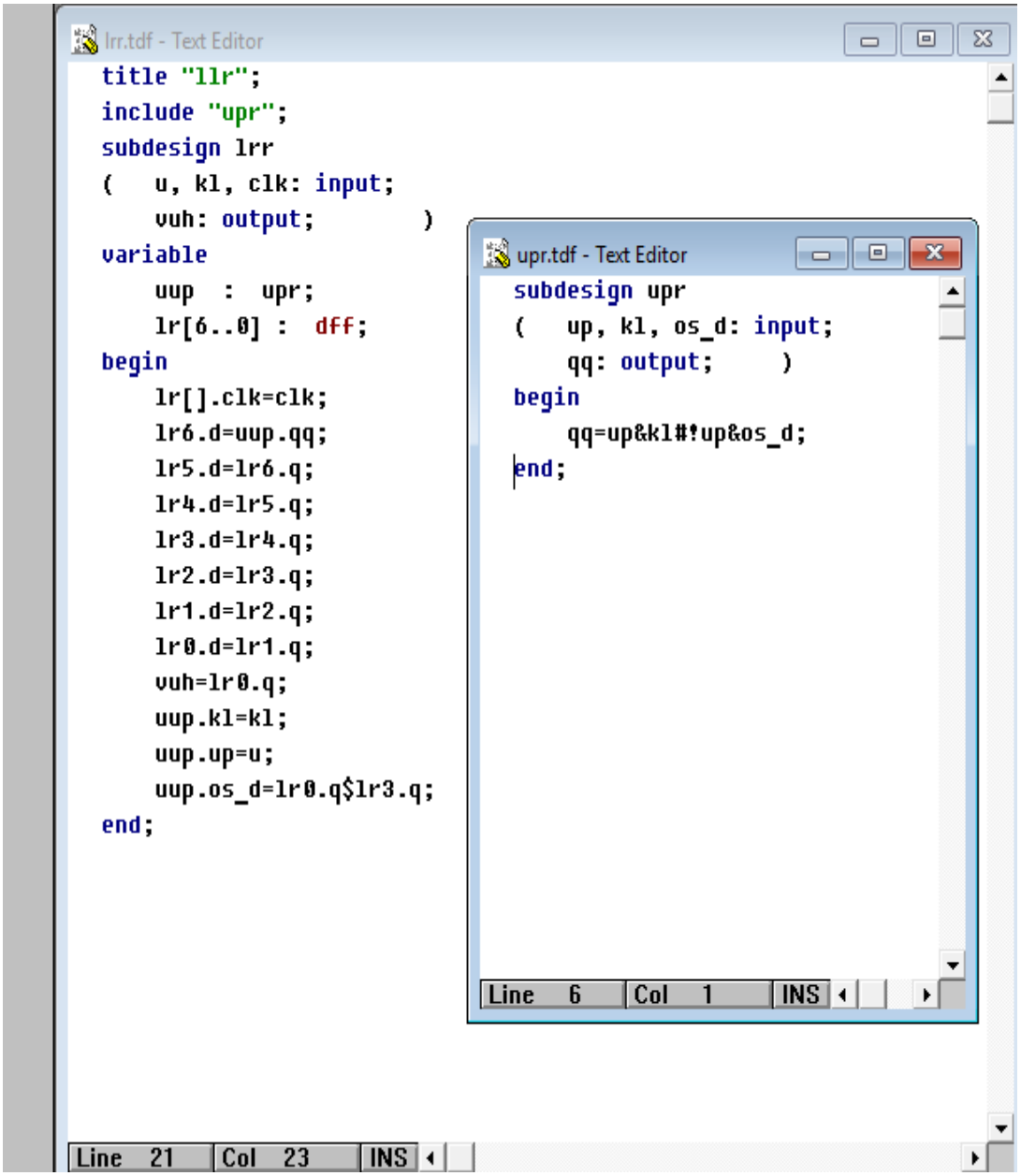


Fig. 6.28. The program implementing LRR.

Also, a linear recurrent register can be created in a graphic editor from individual modules, and then combine them using the generated symbols by the CREATE DEFAULT SYMBOL command (fig. 6.29, 6.30).

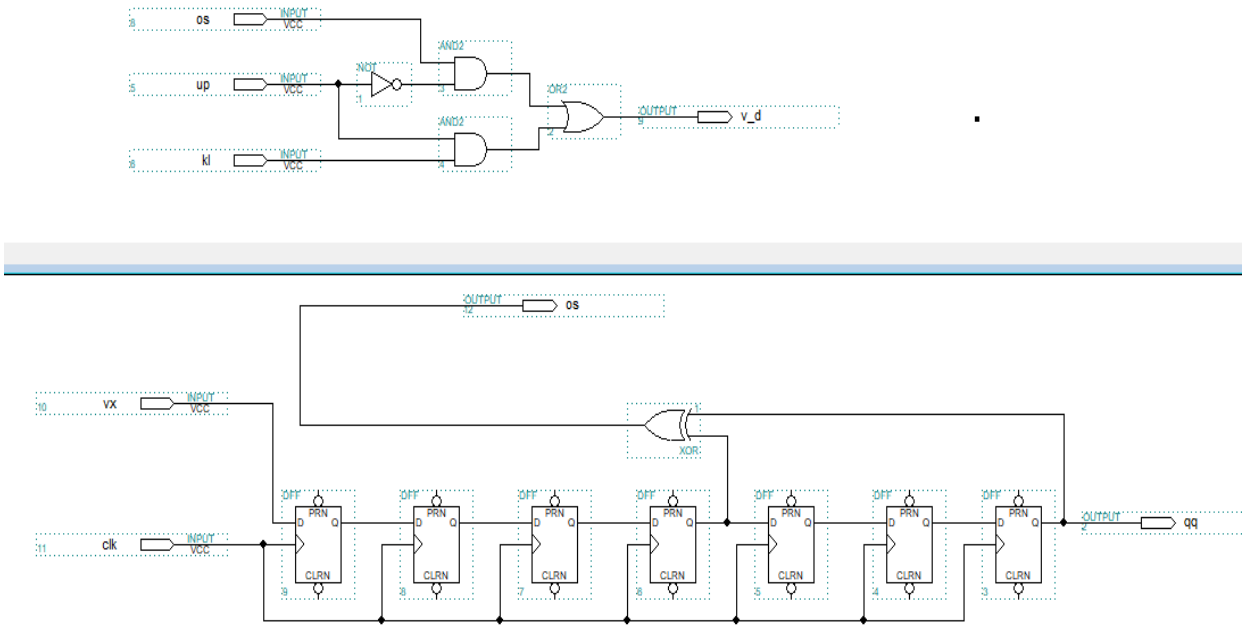


Fig. 6.29. Diagram of LRR.

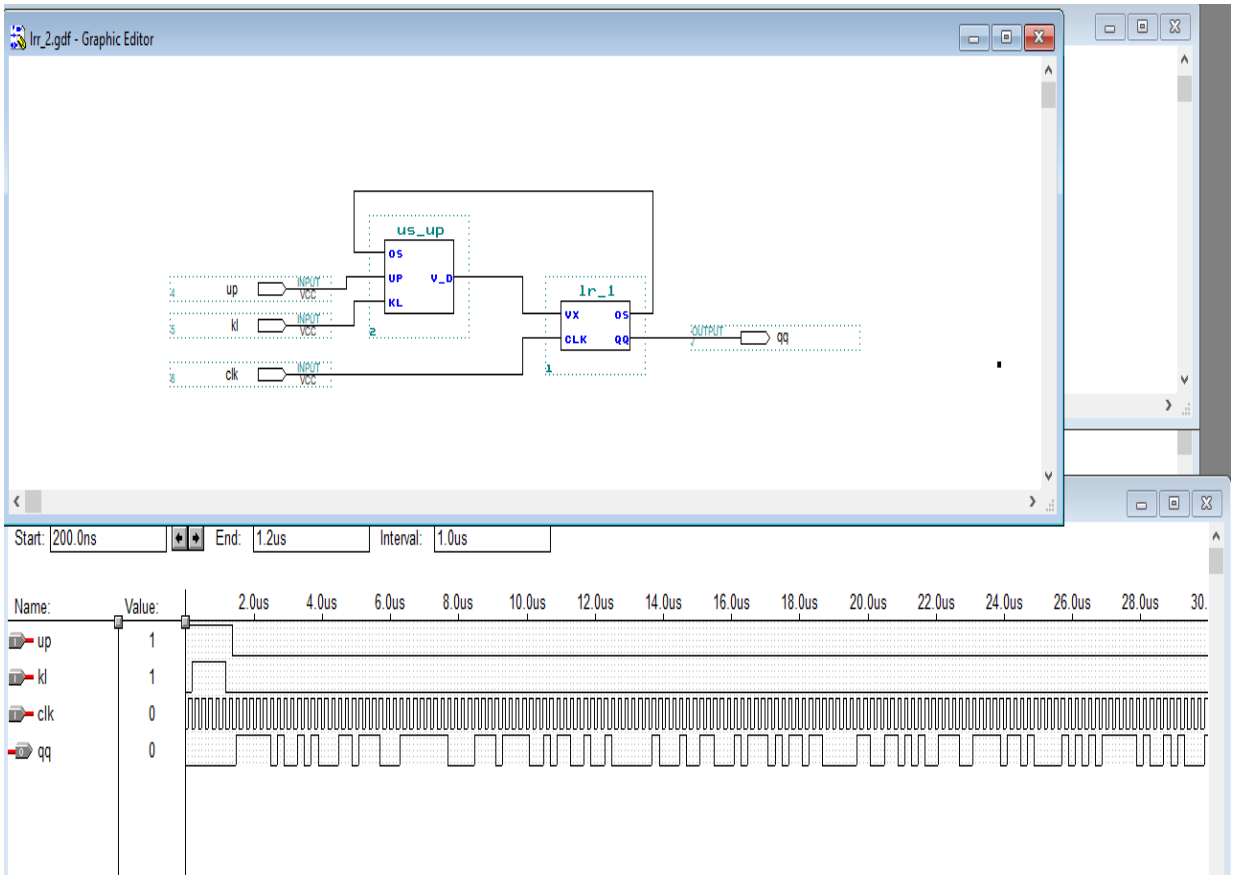


Fig. 6.30. Diagram of LRR and operation diagram.

6.5. DSTU GOST 28147:2009 in simple replacement mode

GOST 28147-89 is a block cipher with a 256- bit key and 32 conversion cycles, operating with 64-bit blocks. The basis of the cipher algorithm is the Feistel network . The basic mode of encryption according to GOST 28147-89 is the mode of simple substitution (more complex modes of gamming , gamming with feedback and the mode of imitation are also defined). For encryption in this mode, the plain text is first split into two halves (the least significant bits, the most significant bits). At each cycle of operation, the summation of the least significant bits is used with the 32 bits of the key sequence selected in a given order.

The result is split into eight 4-bit subsequences, each of which enters the input of its *substitution table node* (in ascending order of bit precedence), called *an S-block below* . The total number of GOST S-blocks is eight, that is, the same number of subsequences. Each *S-block* is a permutation of numbers from 0 to 15. The first 4-bit subsequence goes to the input of the first S-block, the second to the input of the second,etc.

The outputs of all eight S-blocks are combined into a 32-bit word, then the entire word is cyclically shifted to the left (to the most significant bits) by 11 bits. Then the shift result is summed modulo two with information from the second drive, and the result is written to the first, and so on for 32 cycles. The encrypted information is then supplied to the output.

An example of implementation of DSTU GOST 28147:2009 in the mode of simple replacement using CAD ALTERA MAX + plusII 10.2

Implementation of DSTU GOST 28147:2009 is carried out in the Text Editor File., We provide open information, key information, synchronization signals and service commands (key [31..0], inf , clk , code[7..0], kl , vvod) at the input.

We set internal variables in the form of registers for storing temporary information in the variable section, registers on D-triggers are used for this. At the end of the program we will receive 64 bits of encrypted text, which is supplied to the output (fig. 4.31, 4.32, 4.33, 4.34). All commands, symbols, operators and other things used in the implementation were discussed in detail in the 3rd chapter of the manual.

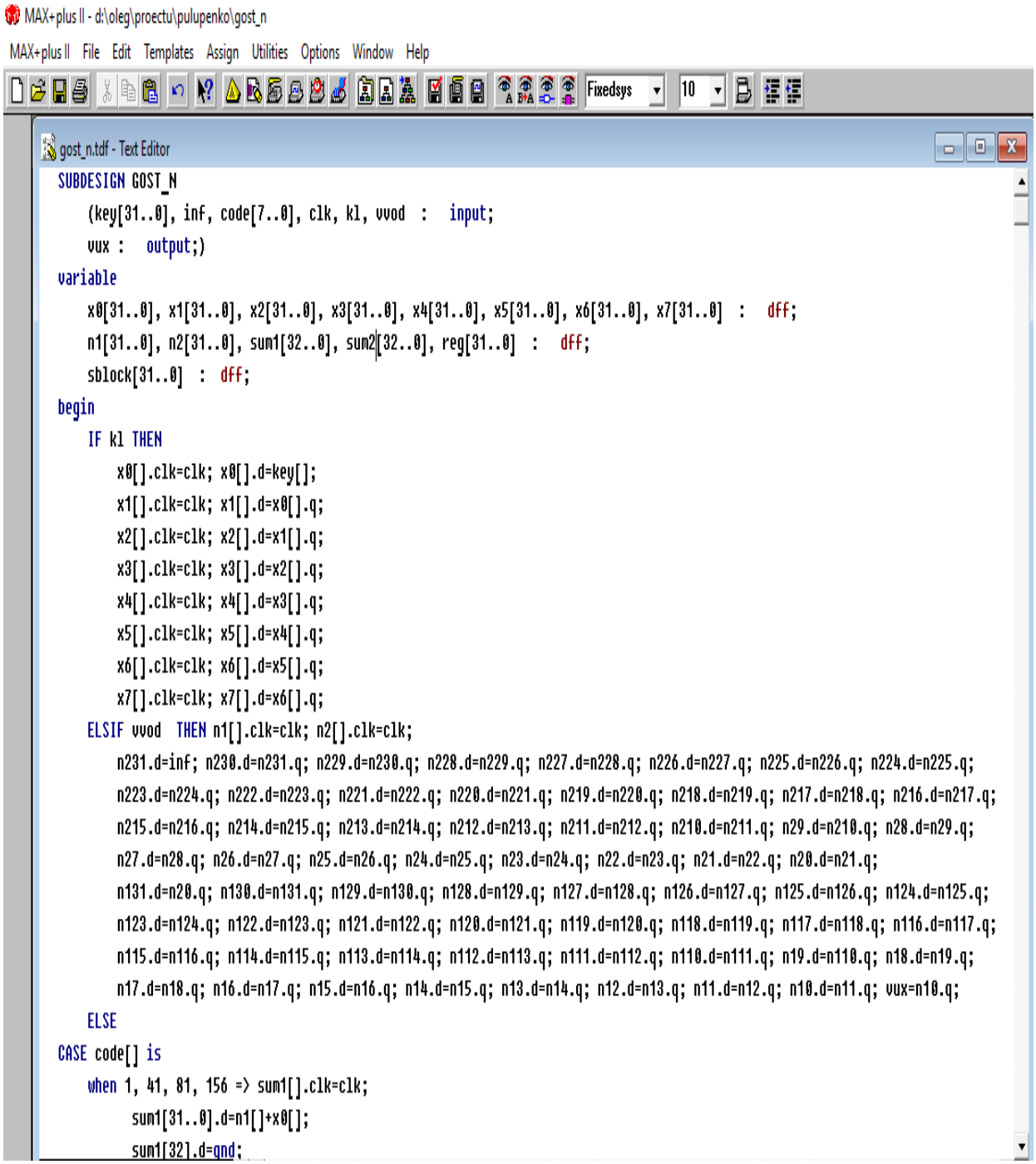


Fig. 6.31. Software implementation of GOST 28147-89 in AHDL language in simple replacement mode.

```
    when 2, 7, 12, 17, 22, 27, 32, 37, 42, 47, 52, 57, 62, 67, 72, 77, 82, 87, 92, 97, 102, 107,
        112, 117, 122, 127, 132, 137, 142, 147, 152, 157 => sblock[].clk=clk;
TABLE
    sum1[3..0].q => sblock[3..0].d;
    H"0"=>H"1"; H"1"=>H"F"; H"2"=>H"D"; H"3"=>H"0"; H"4"=>H"5"; H"5"=>H"7"; H"6"=>H"A"; H"7"=>H"4";
    H"8"=>H"9"; H"9"=>H"2"; H"A"=>H"3"; H"B"=>H"E"; H"C"=>H"6"; H"D"=>H"B"; H"E"=>H"8"; H"F"=>H"C";
    END TABLE;
TABLE
    sum1[7..4].q => sblock[7..4].d;
    H"0"=>H"4"; H"1"=>H"A"; H"2"=>H"9"; H"3"=>H"2"; H"4"=>H"D"; H"5"=>H"8"; H"6"=>H"0"; H"7"=>H"E";
    H"8"=>H"6"; H"9"=>H"B"; H"A"=>H"1"; H"B"=>H"C"; H"C"=>H"7"; H"D"=>H"F"; H"E"=>H"5"; H"F"=>H"3";
    END TABLE;
TABLE
    sum1[11..8].q => sblock[11..8].d;
    H"0"=>H"E"; H"1"=>H"B"; H"2"=>H"4"; H"3"=>H"C"; H"4"=>H"6"; H"5"=>H"D"; H"6"=>H"F"; H"7"=>H"A";
    H"8"=>H"2"; H"9"=>H"3"; H"A"=>H"8"; H"B"=>H"1"; H"C"=>H"0"; H"D"=>H"7"; H"E"=>H"5"; H"F"=>H"9";
    END TABLE;
TABLE
    sum1[15..12].q => sblock[15..12].d;
    H"0"=>H"5"; H"1"=>H"8"; H"2"=>H"1"; H"3"=>H"D"; H"4"=>H"A"; H"5"=>H"3"; H"6"=>H"4"; H"7"=>H"2";
    H"8"=>H"E"; H"9"=>H"F"; H"A"=>H"C"; H"B"=>H"7"; H"C"=>H"6"; H"D"=>H"0"; H"E"=>H"9"; H"F"=>H"B";
    END TABLE;
TABLE
    sum1[19..16].q => sblock[19..16].d;
    H"0"=>H"7"; H"1"=>H"D"; H"2"=>H"A"; H"3"=>H"1"; H"4"=>H"1"; H"5"=>H"8"; H"6"=>H"9"; H"7"=>H"F";
    H"8"=>H"E"; H"9"=>H"4"; H"A"=>H"6"; H"B"=>H"C"; H"C"=>H"B"; H"D"=>H"2"; H"E"=>H"5"; H"F"=>H"3";
    END TABLE;
```

```
TABLE
    sum1[23..20].q => sblock[23..20].d;
    H"0"=>H"6"; H"1"=>H"C"; H"2"=>H"7"; H"3"=>H"1"; H"4"=>H"5"; H"5"=>H"F"; H"6"=>H"D"; H"7"=>H"8";
    H"8"=>H"4"; H"9"=>H"A"; H"A"=>H"9"; H"B"=>H"E"; H"C"=>H"0"; H"D"=>H"3"; H"E"=>H"B"; H"F"=>H"2";
    END TABLE;
TABLE
    sum1[27..24].q => sblock[27..24].d;
    H"0"=>H"4"; H"1"=>H"B"; H"2"=>H"A"; H"3"=>H"0"; H"4"=>H"7"; H"5"=>H"2"; H"6"=>H"1"; H"7"=>H"D";
    H"8"=>H"3"; H"9"=>H"6"; H"A"=>H"8"; H"B"=>H"5"; H"C"=>H"9"; H"D"=>H"C"; H"E"=>H"F"; H"F"=>H"E";
    END TABLE;
TABLE
    sum1[31..28].q => sblock[31..28].d;
    H"0"=>H"D"; H"1"=>H"B"; H"2"=>H"4"; H"3"=>H"1"; H"4"=>H"3"; H"5"=>H"F"; H"6"=>H"5"; H"7"=>H"9";
    H"8"=>H"0"; H"9"=>H"A"; H"A"=>H"E"; H"B"=>H"7"; H"C"=>H"6"; H"D"=>H"8"; H"E"=>H"2"; H"F"=>H"C";
    END TABLE;

when 3, 8, 13, 18, 23, 28, 33, 38, 43, 48, 53, 58, 63, 68, 73, 78, 83, 88, 93, 98, 103, 108,
       113, 118, 123, 128, 133, 138, 143, 148, 153, 158 => reg[].clk=clk;
       reg[10..0].d=sblock[31..21].q;
       reg[31..11].d=sblock[20..0].q;

when 4, 9, 14, 19, 24, 29, 34, 39, 44, 49, 54, 59, 64, 69, 74, 79, 84, 89, 94, 99, 104, 109, 114,
       119, 124, 129, 134, 139, 144, 149, 154, 159 =>  sum2[].clk=clk;
       sum2[31..0]=n2[31..0].q$reg[31..0].q;
       sum2[32]=gnd;

when 5, 10, 15, 20, 25, 30, 35, 40, 45, 50, 55, 60, 65, 70, 75, 80, 85, 90, 95, 100, 105, 110,
       115, 120, 125, 130, 135, 140, 145, 150, 155 =>  n2[].clk=clk;
       n2[31..0].d=n1[31..0].q;
       n1[].clk=clk;
       n1[31..0].d=sum2[31..0].q;
```

Fig. 6.32. Software implementation (continued).

```
when 6, 46, 86, 151 => sum1[].clk=clk;   sum1[31..0].d=n1[]+x1[];   sum1[32].d=gnd;

when 11, 51, 91, 146 => sum1[].clk=clk;   sum1[31..0].d=n1[]+x2[];  sum1[32].d=gnd;

when 16, 56, 96, 141 => sum1[].clk=clk;   sum1[31..0].d=n1[]+x3[];  sum1[32].d=gnd;

when 21, 61, 101, 136 => sum1[].clk=clk;   sum1[31..0].d=n1[]+x4[];  sum1[32].d=gnd;

when 26, 66, 106, 131 => sum1[].clk=clk;    sum1[31..0].d=n1[]+x5[];   sum1[32].d=gnd;

when 31, 71, 111, 126 => sum1[].clk=clk;    sum1[31..0].d=n1[]+x6[];   sum1[32].d=gnd;

when 36, 76, 116, 121 => sum1[].clk=clk;    sum1[31..0].d=n1[]+x7[];   sum1[32].d=gnd;

when 160 =>     n2[].clk=clk;          n2[31..0].d=sum2[31..0].q;
       end case;
    end if;
end;
```

Fig. 6.33. Software implementation (continued).

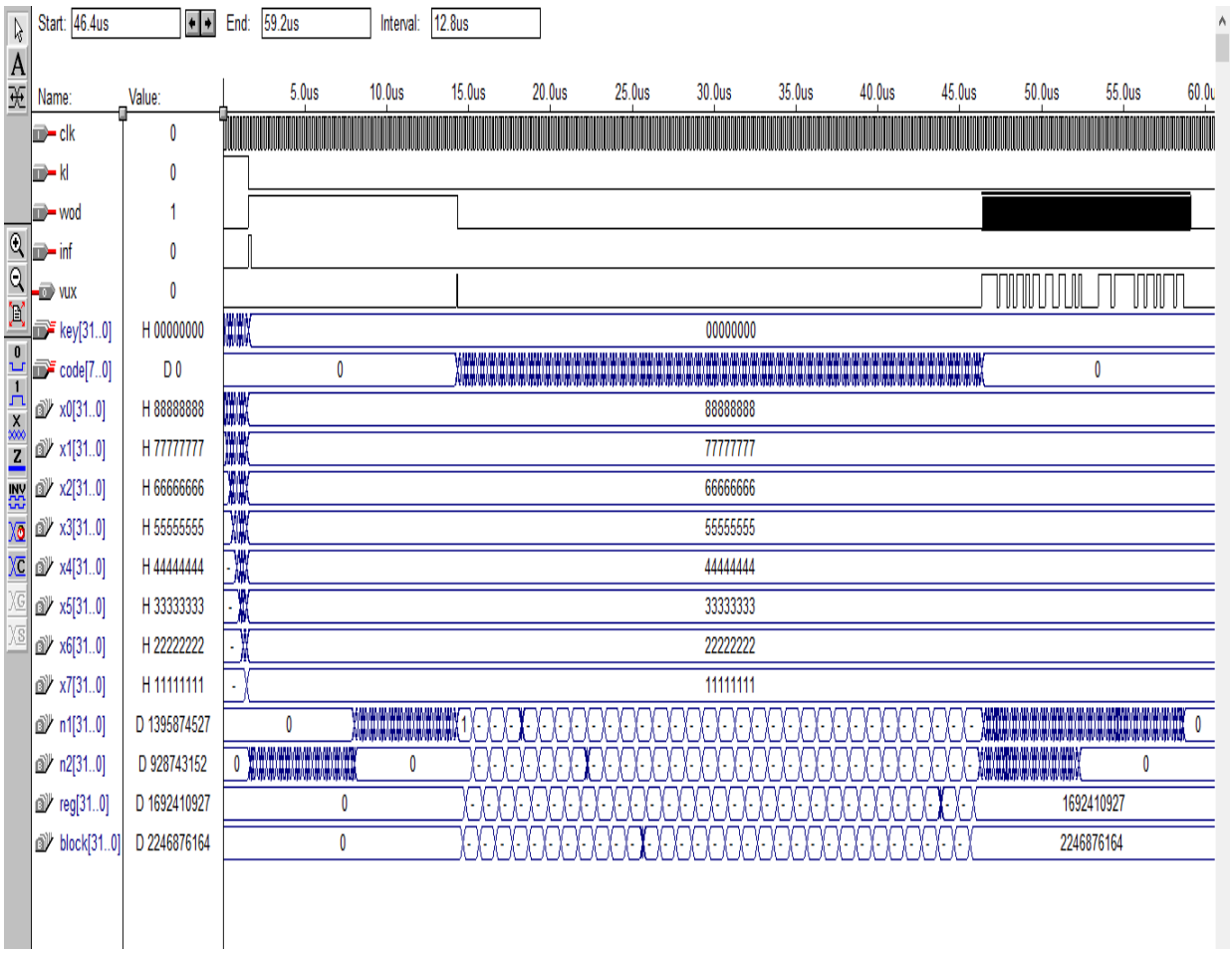


Fig. 6.34. Diagrams of the functioning of the program implementation.

Kulinich O.M., Kasatkin D.Yu., Lakhno V.A., Nikitenk0 Y.V.Kharchuk N.S.

TEXTBOOK

DESIGN OF DATA PROCESSING AND SECURITY SYSTEMS

Publisher of FOP Yamchynskyi O.V.
03150, Kyiv, str. Predslavynska, 28
Certificate of entry into the State Register
of the subject of the publishing case DK No. 6554 dated 12/26/2018

Manufacturer LLC "CPU "COMPRINT"
Format 60×8 /16. Circulation 50 Ave. Um. printing. sheet 23,8. Deputy No. 1 43
03150, Kyiv, str. Predslavynska, 28
Certificate of entry into the State Register
of the subject of the publishing case DK No. 4131 dated 04.08.2011