

# **Theory of Logic Circuits**

## **Laboratory manual**

### **Exercise 12**

**Computer aided design for circuit development**



### 1. Introduction. Algorithms for system synthesis and optimization.

There are many CAD methods helping Circuit developer do their job faster and better. During this labs you will test two methods:

1. A Table of Switching Sequence method for synthesis of sequential systems,
2. A Kazakow method, useful for I/O function search, generating non-redundant solution during prime implicants search process.

### 2. The Table of Switching Sequence method (SST)

(Also known as Table of Successive Connections)

The Table of Switching Sequence method, also known as Table of Successive Connections is applicable for asynchronous sequential systems, given by description, time diagram or flow chart. Several steps must be done to achieve final result which is SOP or POS of all outputs and additional elements (see later in this chapter for additional elements description). The algorithm is presented in the following subpoints.

#### 2.1. Obtain primary SST from the description of operation of the circuit

SST consists of several rows, each representing one input, output or additional elements (primary SST consists only of rows describing input and output signals) plus row header showing workflow steps of the system and footer containing numeric state of the system. Thus, each cell represents an event - a change in input, additional element or output signal. To achieve correct primary SST one must follow all changes in system's workflow and notify them, one by one in the SST table. Plus sign in the cell represents event when signal changes from logical 0 to 1, minus sign otherwise. Only one change per column can be notified. When two output signal changes immediately because of input signal change, one is always step before the other. Notification finishes, when all possibilities (given by task circumstances) are considered and put into the SST. Always the last drawn column in SST must be identical to the first one. All columns, but last one represents a work cycle of the system. As the cycle is restartable, you may think about it as it is drawn on a can, turning round: when cycle ends, the next one may start from the beginning (the first left column). As soon as the work cycle is known, it is necessary to draw switching-on and - off conditions for outputs (and additional elements, if any). Switching-on condition becomes true always one cell before a cell containing plus sign and finishes one cell before a cell containing minus sign. Each row (representing one signal) has its own value, a power of 2, starting from 0 ( $2^0=1$ ) for the top most row.

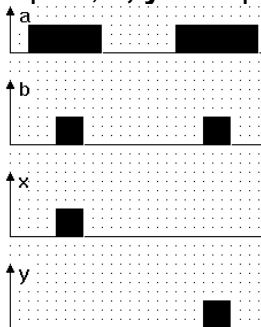
Numerical state of the system is calculated as follows:

- for the first column, its numerical state is equal to the sum of values representing those rows, where plus sign is indicating in the cell (given by selected row and first cell)
- for all the next columns, numerical state is obtained as a numerical state from the proceeding column and a change (event) in the considered column
  - when plus sign, one adds a value representing the signal changing,



- when minus sign, one subtracts this value.

Let's consider sample SST obtaining process, based on the time diagram description (**a**, **b** – inputs; **x**, **y** – outputs):



Analysis have to be done starting from the left of the diagram, notifying any change in input, output and additional elements. Notice, that an input signal causes output/additional element to change, never different. Only one signal changes at a single step. In the case above, first rising up of **b** signal causes change of **x**, (as **b** is input, **x** is output). Analyzing step by step each change in input and output signal, we obtain following SST:

STEP		Work cycle												
		0	1	2	3	4	5	6	7	8	9	10	11	0
SIGNAL	2 <sup>0</sup> a	■	+					■	+					■
	2 <sup>1</sup> b	■		+		■				+		■		
	2 <sup>2</sup> x	■			+	■								
	2 <sup>3</sup> y	■									+		■	
N. S. U.		0	1	3	7	5	1	0	1	3	11	9	1	0

Notify, that switching-on condition become true one step before plus sign and disappear one step before minus sign and has to be treated this way (as arrows drawn in the SST).

## 2.2. Check SST solvability.

When SST is ready, solvability analysis have to be done. We search for repeating in N.S.U. (numerical state of the system) and check, if for selected number repeating appears to have the same or different working conditions (output and additional elements switching). In the case above 0, 1 and 3 N.S.U. repeats:

- 0 repeats in steps 0 and 6, however both steps have the same switching condition which are x=off, y=off,
- 1 repeats in steps 1, 5, 7 and 11, and have the same switching condition which are x=off, y=off as well,
- 3 repeats in steps 2 and 8 (contradictory states), but has different switching condition – in step 2 x=on & y=off, in step 8 x=off & y=on, so SST is unsolvable right now and additional element(s) has(have) to be added.

It is useful to notify contradictory states with index of repeating, as on the figure below:



STEP		Work cycle												
		0	1	2	3	4	5	6	7	8	9	10	11	0
SIGNAL	2 <sup>0</sup> a	█	█					█	█					█
	2 <sup>1</sup> b	█		█		█				█		█		
	2 <sup>2</sup> x	█			█		█							
	2 <sup>3</sup> y	█									█		█	
N. S. U.		0	1	3	7	5	1	0	1	3	11	9	1	0

If there are no contradictory states, SST is solvable. If there are any, additional elements have to be added in correct places. In the case considered, SST is unsolvable.

### 2.3. Resolve unsolvable SST.

Resolving unsolvable SST is done by splitting SST into parts, that don't contain contradictory states. In the places of division, one or several additional elements have to be switched-on and off, to differ parts containing contradictory states. If partitioning is done well, in the SST obtained (containing added additional elements) should be no contradictory state.

Generally, as many additional elements have to be added as no contradictory states belong to the same part (one part can be split with another), however there are hints helping while splitting SST into the parts (see later in this chapter). In the case, only one additional element is necessary to differ steps 2 and 8 each other.

The most important to do is to correctly choose the places of splitting SST. There are 4 hints helping in this:

- one part (even split into many with others) should not contain contradictory states,
- a border (splitting place) between part A and B can't be selected next to the step that repeats not contradictory way and finishes part B or belongs to part A, unless part A is split into the parts and always next to the step (which belongs to part A) that repeats not contradictory way there is border between part A and B,
- a border between part A and B can't be selected next to the step that is contradictory to the step belonging to the part B,
- neighbour parts should be coded neighbour in terms of bi-stable logic, but it is always safe to put a border next to the none repeating step (however sometimes it is impossible).

Borders in the case considered are drawn bold on the figure below:

STEP		0	1	2	3	4	5	6	7	8	9	10	11	0
SIGNAL	2 <sup>0</sup> a	█	█					█	█					█
	2 <sup>1</sup> b	█		█		█				█		█		
	2 <sup>2</sup> x	█			█		█							
	2 <sup>3</sup> y	█										█		█
N. S. U.		0	1	3	7	5	1	0	1	3	11	9	1	0



STEP		0	1	2	3	4	5	6	7	8	9	10	11	0
SIGNAL	2 <sup>0</sup> a													
	2 <sup>1</sup> b													
	2 <sup>2</sup> x													
	2 <sup>3</sup> y													
N. S. U.		0	1	3	7	5	1	0	1	3	11	9	1	0

Change code of added elements:

q1

When having SST containing additional steps, as on the figure below, solvability checking have to be done once more.

STEP		0	1	2	3	4	5	6	7	8	9	10	11	12	13	0
SIGNAL	2 <sup>0</sup> a															
	2 <sup>1</sup> b															
	2 <sup>2</sup> x															
	2 <sup>3</sup> y															
	2 <sup>4</sup> q1															
N. S. U.		0	1	3	7	5	21	17	16	17	19	27	25	9	1	0

When SST is still unsolvable it means that incorrect border places were selected, and this step of solution have to be done again for correct border places. In the case SST is solvable.

### 2.4. Obtain canonical SOP and POS forms for output signals and additional signals.

The result comes in canonical form, following the switching-on and off conditions, as shown on the previous figure:

$$x = \sum (3, 7, 5)_{q1 y x b a}$$

$$x = \prod (1, 21, 17, 16, 19, 27, 25, 9, 0)_{q1 y x b a}$$

$$y = \sum (19, 27, 25)_{q1 y x b a}$$

$$y = \prod (1, 3, 7, 5, 21, 17, 16, 9, 0)_{q1 y x b a}$$

$$q1 = \sum (5, 21, 17, 16, 19, 27)_{q1 y x b a}$$

$$q1 = \prod (1, 3, 7, 25, 9, 0)_{q1 y x b a}$$

### 3. The Table of Switching Sequence method software.

(Known also as Table of Successive Connections, as named in software.)



### 3.1. System requirements:

System requirements:

Windows based or compatible OS, able to run both 16-bit and 32-bit Windows applications.

### 3.2. Basics:

Basic idea when using Table of Switching Sequence method is to follow four main steps:

- preparing description of working system, means to formulate conditions of working,
- preparing and description presentation of working system by the method Table of Successive Connection,
- hacking solution of the table,
- if table is solved - presentation solution, if not - inserting to the system added elements and then solution presentation again.

### 3.3. Preparing model:

When preparing description of working system, there are 3 possible ways describing working and non-working conditions:

- gather all data from file (file format included in help system), MENU: [File/Open table]
- input formula of connection, MENU: [File/Formula of connection]:
  - enter number of input and output signals
  - enter input and output signal names (single letters only, used as signal IDs)
  - enter begin state for Table of Switching Sequence (typically 0)
  - describe run of signals (remember to enclose complete conditions both for working and non-working),  
ex. (input signals: **a**, **b**, output signal **c**) : a+b+a-c+a+b-a-c-
- input data using timing diagram, MENU: [File/Time diagram]:
  - enter number of input and output signals
  - enter input and output signal names (single letters only, used as signal IDs)
  - draw sample timing diagram, example below:

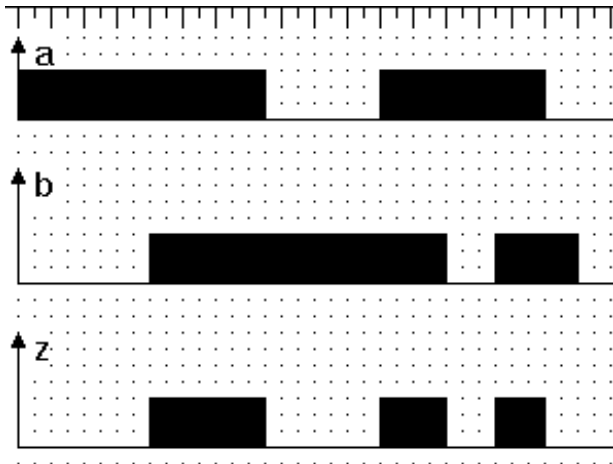


Fig. 1 Sample timing diagram

Notice, that this timing chart and formula of connection assumes combinational circuits (non-sequential circuits).

Timing diagram editor:



- drawing tool



- clear selected block of diagram



- run algorithm



- cancel edition

### 3.4. Options and menu items:

Allows user to change working behavior and result presentation of software:

- Diagram options, MENU: [Options/Diagram]
- Working (partially / only end solution) , MENU: [Options/Working], when partially is selected, system prompts user after each step during solution (in case there is necessary to insert “added elements”, depending on working and non-working conditions occurrences),
- Presentation (general / particular), MENU: [Options/Presentation]
- Solution settings, MENU: [Options/Presentation], decides on solution format.
- Reporting, MENU: [Report] - writes text report.
- Printing, MENU: [Printing] - prints report and/or timing diagram on selected system printer.
- Save solution on disk, MENU: [Write\_solution] – writes solution to the selected file
- End program, MENU: [End] – ends program



### 4. Kazakow method (algorithm).

#### 4.1. Algorithm description:

Kazakow algorithm is efficient, small and fast way, providing irredundant solution for output function, given by many input signals (i.e. 20 and more), when output is weakly specified function. Weakly specified function is the one that is given by a small amount of one's components / zero's factors

The idea for this method bases upon searching for prime implicants covering all canonical expressions from one's component set.

Algorithm starts from any canonical 1's component.

We search for a prime implicant covering this component. Checking if any product expression (single or multi-variable) that comes from this canonical component is a prime implicant does searching.

First we check single-variable (single-literal) product expressions, than increasing product expression by one variable, and so on. A product expression is prime implicant, when is not covered by any 0 factor.

When prime implicant is found, we mark all 1's components covered by this prime implicant. We continue searching for prime implicants for all not covered 1's components.

Finally we reduce number of implicants by removing those implicants, which deletion do not leaves components not covered.





### 4.2. Learn by example (see Fig. 2):

1's components and 0's factors, as shown on Fig. 2, are giving output function F based on 20 input signals X1...X20 .

First, let's analyze first 1's component.

Each single-variable literal is covered by some element of 0's factor set, so we analyze all possible arrangements for two selected literals (multi-variable). Arrangements  $\sim X1\sim X2 \dots \sim X1X12$  are also covered by some element of 0's factor set, but no element from this set contains  $\sim X1\sim X13$  expression, so this is our first prime implicant I1.

Now we mark with number 1 all rows (see Fig. 1) containing  $\sim X1\sim X13$  (in other words, where X1 and X13 are equal to zero) in 1's component set. Those rows are numbered 1,3,9,11 and are covered by prime implicate I1.

Now we continue with first, not covered 1's component, in row 2. We found single-variable literal  $\sim X11$ , not covered by 0's factor set. This is second, prime implicant, I2. Now we mark with number 2 all rows containing  $\sim X11$  in 1's component set. Those rows are numbered 2,3,7 and 13.

We continue with element in row 4, having prime implicant I3 as  $\sim X1X4$ , marking with number 3 rows 2,4,9,10 and 13.

We continue with element in row 5, having prime implicant I4 as  $\sim X2X5$ , marking with number 4 rows 5 and 10.

We continue with element in row 6, having prime implicant I5 as  $X2\sim X16$ , marking with number 5 rows 3,6,7,8,11 and 12.

As all 1's component are covered by at least one prime implicant, searching procedure is done.

Now let's check if estimated solution is irredundant. We do so, by removing unnecessary implicants (an implicant is unnecessary, when if is removed, at least one of 1's components becomes uncovered).

We can remove I5, redundant implicant.

Finally, the solution is:

$$F = \sim X1\sim X13 + \sim X1X4 + \sim X2X5 + X1\sim X16, \text{ (as sum of implicants I1,I3,I4 and I5).}$$

	Input variables																				Found implicants
	X1	X2	X3	X4	X5	X6	X7	X8	X9	X10	X11	X12	X13	X14	X15	X16	X17	X18	X19	X20	
	1's component set:																				:
1	0	0	0	0	0	0	0	0	0	1	1	0	0	0	0	0	0	0	0	0	1
2	0	0	0	1	0	0	0	0	0	0	0	1	1	1	0	0	0	0	0	0	2,3
3	0	1	0	0	0	1	0	0	0	0	0	1	0	1	0	0	0	1	1	1	1,2,5
4	0	0	1	1	0	0	1	0	0	0	1	1	1	0	1	1	1	0	0	0	3
5	1	0	1	0	1	0	1	1	1	1	1	0	0	0	0	0	0	0	0	0	4
6	1	1	0	1	1	1	0	0	0	0	1	0	0	1	0	0	0	0	0	0	5



7	1	1	0	1	1	0	1	1	1	1	0	0	0	0	0	0	0	0	0	2,5
8	1	1	0	0	0	0	0	1	1	1	1	0	0	1	1	0	1	1	1	5
9	0	0	1	1	0	1	0	0	0	0	1	0	0	0	0	0	0	0	0	1,3
10	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	3,4
11	0	1	0	0	0	0	0	0	0	0	1	0	0	0	0	0	0	0	0	1,5
12	0	1	0	0	0	0	0	0	0	0	1	1	1	0	0	0	1	0	0	5
13	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	1	1	1	1	2,3
0's factor set:																				
	1	1	0	0	0	1	0	0	1	1	1	1	0	0	1	1	0	1	0	
	1	1	1	0	0	1	0	0	1	1	1	0	0	0	1	1	0	0	1	
	1	1	1	0	1	0	1	1	0	0	1	1	1	0	1	1	0	0	1	
	1	0	1	0	0	0	1	1	1	1	1	1	1	1	1	1	1	1	1	
	0	0	1	0	0	0	0	0	0	0	1	1	1	0	1	0	0	1	1	
	0	0	0	0	0	1	1	0	1	1	1	0	1	1	1	0	0	0	0	
	1	0	0	1	0	0	0	1	1	1	1	1	1	1	0	1	1	1	1	
	1	1	1	1	1	1	1	1	1	1	1	1	1	0	1	1	1	1	1	

Fig. 2 Output function given by canonical form

## 5. Implementation of functions with MUX and DMUX

Optimal realization with DMUX and MUX elements is not a trivial task. Available software allows for such implementation as long as a function to be connected meets the constraints as to the number of variables and type of a chosen structure.

### 5.1. Short overview

Program MuxDmux enables computer assisted design for realizations of circuits with multiplexers and demultiplexers.

The operation of the program consists of entering a function definition and then selecting a structure in which this function will be implemented. Function may be entered in two forms: canonical numerical decimal or literal. The function definition may be also loaded from previously saved file. The entered function is then minimized using Quine-McCluskey algorithm. If user chooses literal form of function to enter then they have possibility to turn off minimization. Next a user can define a structure that is required. Afterwards the program generates a diagram of the chosen design. The result is displayed in the program window. When a structure is set up it can be stored in a file as a project definition, exported to a chosen graphical format or printed.

Two different file formats of data inputs are implemented in the program, mainly:

- Function definition file – it stores a definition of a function entered,
- Project definition file – it contains a definition of a function and a generated structure definition

Therefore, everything in this manual referred to a function is connected with a function definition itself, and referred to a project is connected with a function and a design structure altogether.

### 5.2. Main window

Executing the program causes the window (Fig. 3) to be displayed. As it can be seen, some options from the toolbar and menu are disabled. They become active when appropriate data is entered, or structure generated.

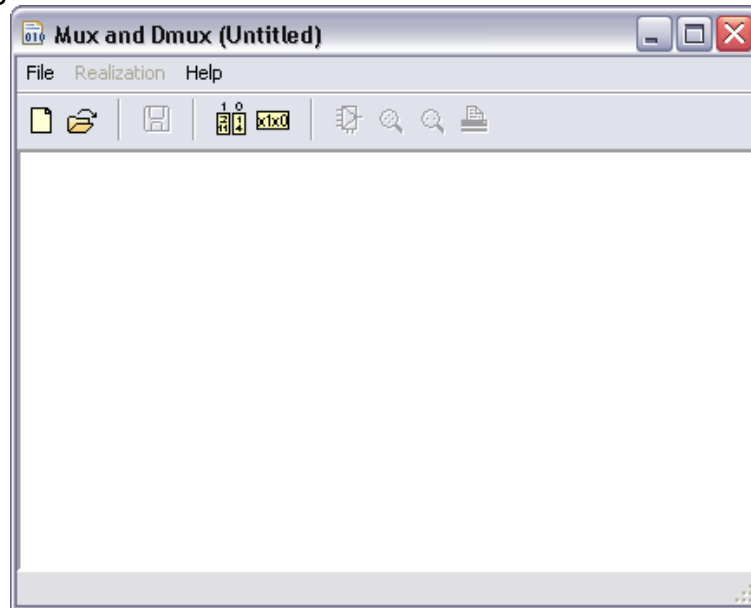


Fig. 3. Main program window

The toolbar with all available functions is shown in Fig. 4.

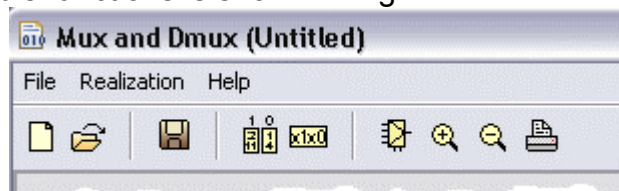











Fig. 4. Toolbar options

Toolbar options:

	– Start a new project
	– Open a project definition
	– Save the project in a project definition file
	– Canonical numerical representation of a function
	– Literal representation of a function
	– Realization of a function
	– Zoom into a diagram
	– Zoom out a diagram
	– Print generated project

The program menu hierarchy is presented in Fig. 5 and Fig. 6.

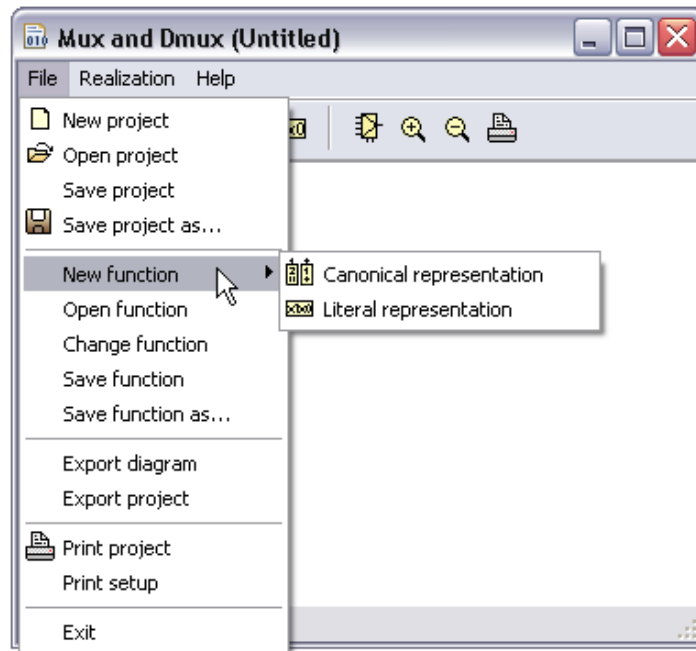


Fig. 5. File menu item hierarchy

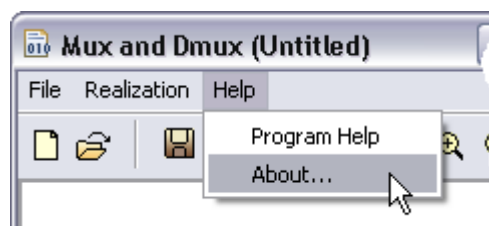


Fig. 6. Help menu item hierarchy

Description of menu items:

**File** – operations connected with entering function definition and files:

- New project – creates new project data, closing current one,
- Open project – opens existing project from a file,
- Save project – saves changes in the current project,
- Save project as... – saves the current project with a new name,
- New function – creates new function definition:
  - Canonical representation – allows to enter the function in canonical numerical representation,
  - Literal representation – allows to enter the function in literal representation,
- Open function – opens existing function definition from a file,
- Change function – changes currently entered function,
- Save function – saves changes in the current function definition file,
- Save function as... – saves the current function definition with a new name,
- Export diagram – exports diagram to graphical format,
- Export project – exports diagram and function definition to graphical format,
- Print project – prints the current project,
- Print setup – printer setup,
- Exit – closes the program,

**Realization** – realization of a function entered,



**Help** – help with the program:

- Program Help – displays program help,
- About... – short information about the program.

### 5.3. Entering function definition

Function may be entered in two forms:

- Canonical numerical decimal representation,
- Literal representation.

In *canonical numerical representation* function definition can be entered by proceeding menu steps **File** ⇒ **New function** ⇒ **Canonical representation** or by clicking "New project"<sup>1</sup> icon  and then "Canonical representation" icon  from the toolbar. Afterwards the window that enables entering data (Fig. 7) shows up.

The window consists of three lists of values:

- **Ones**
- **Zeros**
- **Don't cares**

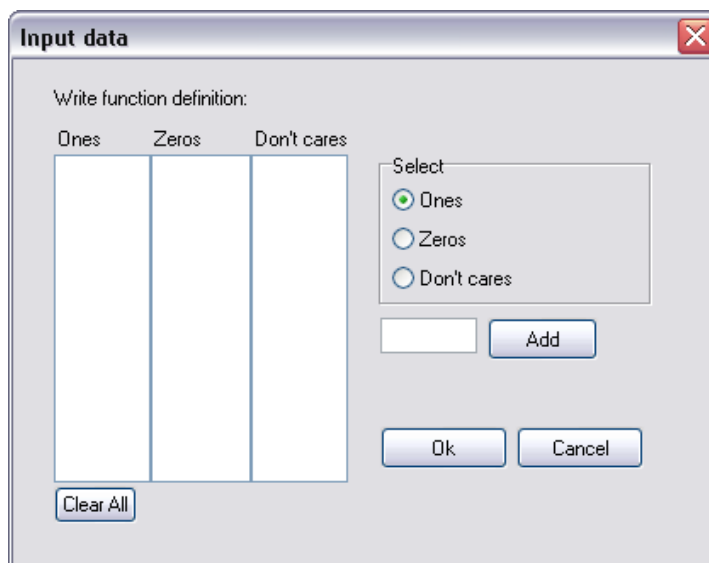


Fig. 7. Window that enables entering function definition in canonical form

Following steps should be performed to enter data to each of these lists:

1. Select to which list the value should be added, by clicking on one of radio buttons.

<sup>1</sup> If currently open project or function definition is not saved user will be prompted to do it.

Select

Ones

Zeros

Don't cares

2. Write value in edit field.

3. Push "Add" button.

According to these steps all function components (ones, zeros and don't cares) have to be entered to these lists. The maximal implemented number of function variables is 10 so maximal possible value that can be entered in the edit field is **1023**, minimum is **0**.

To apply the changes "OK" button should be clicked. After these steps the middle column describing "Zeros" is filled up with values supplementary to the function entered. Thus, all function terms that are not present in "Ones" or "Don't cares" are put into "Zeros" list. In order to delete a term from one of the lists user should select this value, click right mouse button, then from popup menu (Fig. 8) select "Delete". The same operation can be performed by selecting a term and pressing "Del" keyboard key.

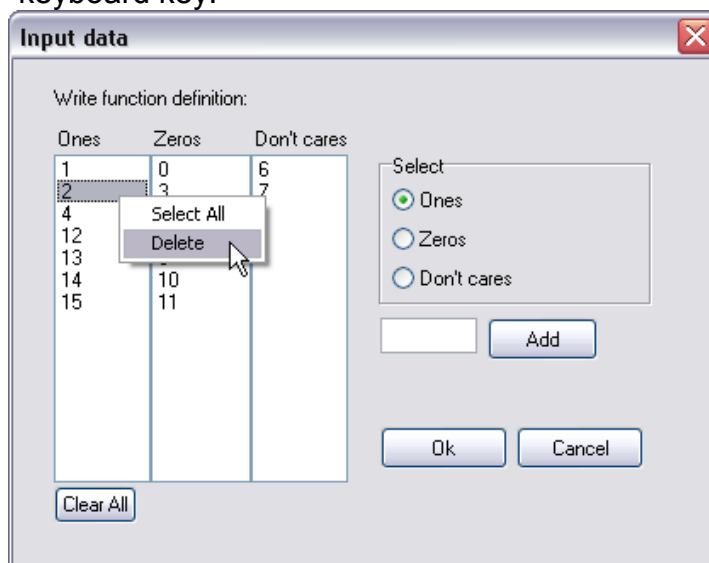



Fig. 8. Deleting term from the list

A list can be erased by selecting this list (by clicking in its area) and choosing in the popup menu "Select All" option and then clicking "Del" keyboard key or choosing "Delete" from the popup menu. All terms from all lists may also be cleared by selecting "Clear all" button. If there is a need to delete more than one value from one list the user should push and hold CTRL keyboard key and select as many values as they need. Then they should release the CTRL key, click right mouse button and select "Delete". All selected lists will be erased.

In order to change previously entered function definition it is advised to select the option from menu **File** ⇒ **Change function** or click on "Canonical representation" icon . To apply or to cancel changes made to the function definition button "OK" or "Cancel" should be pressed appropriately.





In *literal representation* function definition can be entered by proceeding menu steps **File** ⇒ **New function** ⇒ **Literal representation** or by clicking "New project" icon  and then "Literal representation" icon . Afterwards the window that enables entering function definition (Fig. 9) shows up.




Fig. 9. Window that enables entering function definition in literal form

Function definition is written using characters: "~", "x", "+" and numbers 0, 1...9. The function should be rewritten in the form, where complemented value (i.e.  $\overline{x_1}$ ) is represented by *tilde* sign and a variable ( $\sim x_1$ ). It is necessary to emphasize that function variables must be entered in descending order. Exemplary function that is required to be entered in literal form:

$$F = (0,1,3)_{x_1x_0} = x_1x_0 + x_1x_0 + x_1x_0 \equiv \sim x_1 \sim x_0 + \sim x_1 x_0 + x_1 x_0$$

Additionally, if user enters data in reversed order, the program will change the function according to decreasing order of variable indexes, i.e.  $x_0x_1 + x_0x_2 + x_1x_0 \equiv \sim x_1 \sim x_0 + x_2 \sim x_0 + x_1 x_0$ .

In order to change previously entered function definition it is advised to select the option from menu **File** ⇒ **Change function** or click on "Literal representation" icon . Editing a function definition is realized as in usual text fields, i.e. in word processor. A user can copy and paste text into text field in the window, as shown in Fig. 10. To apply or to cancel changes button "OK" or "Cancel" should be pressed appropriately.

As it was stated in overview, the function entered in canonical decimal form is minimized without reaction of the user, whereas function in literal form can be minimized on user request. As well as in entering a function in canonical decimal or in literal form<sup>2</sup>, the function is minimized after clicking "OK". After minimization process function definition may change, thus when user needs to change a function it may look different than the one previously entered.

<sup>2</sup> In the window enabling entering function in literal form user has to check the checkbox in order to minimize the function.

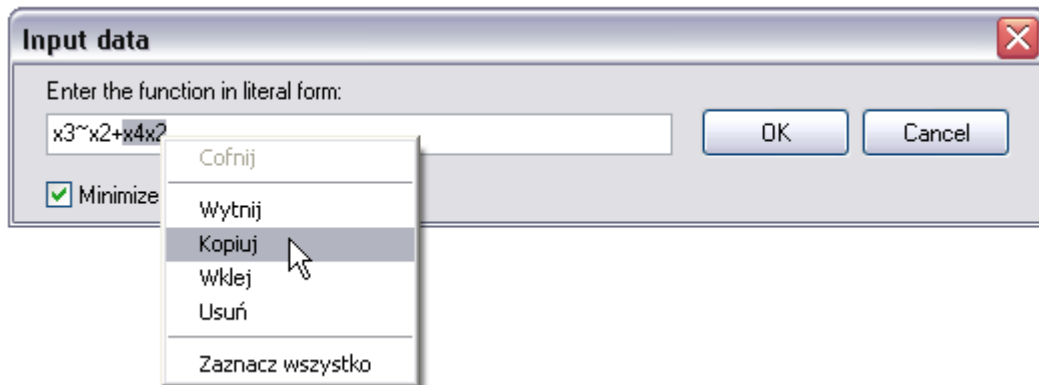


Fig. 10. Editing function definition entered in literal form

The progress bar window (Fig. 11) is displayed when there are eight or more independent function variables in window where function in literal form is entered or more than two hundred of function components in window where function in canonical decimal form is entered. For large number of data minimization process may last some few minutes. The progress bar window is displayed in order to visualize that the application is working properly.

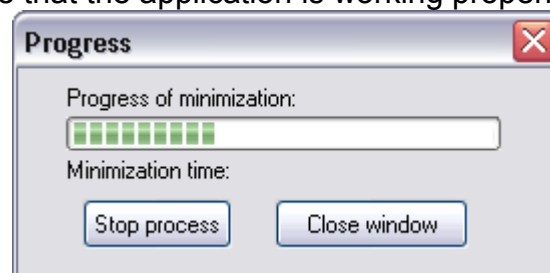


Fig. 11. Progress bar of minimization

When minimization is finished, window (Fig. 12) is displayed, where user can see minimization time. Then user can close the window.

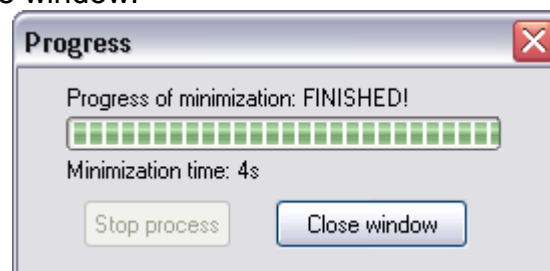


Fig. 12. Window displayed when minimization process is finished

If minimization time takes too much time, user can stop minimization process by clicking "**Stop process**" which causes the window (Fig. 13) to be displayed. If the process is stopped the function is left not minimized.




Fig. 13. Window displayed when minimization process is stopped



## 5.4. Structure realization

Program generates following structures:

- Multiplexer,
- Multiplexer with gates:
  - NAND gates only,
  - NOR gates only,
  - AND and OR and NOT gates,
- Demultiplexer,
- Tree of Multiplexers,
- Multiplexer – demultiplexer.

The structure is chosen by a user. The menu item which allows performing the structure realization is activated only when the function entered can be implemented<sup>3</sup>. Beginning of an implementation process is achieved by clicking **Realization** from the program menu or *realization* icon  from the toolbar. It causes the window (Fig. 14) to be displayed.

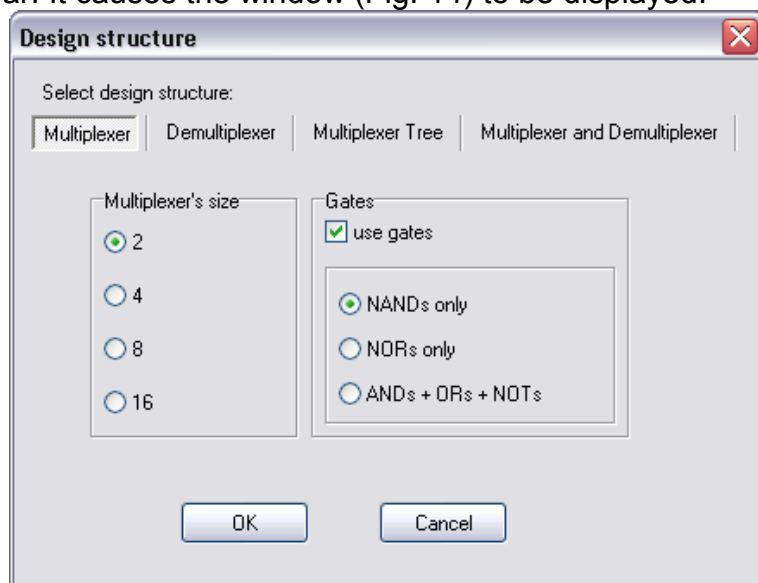


Fig. 14. Window that enables the structure selection

The window contains 4 buttons indicating the possible structures selections. These are: **Multiplexer**, **Demultiplexer**, **Tree of Multiplexers** and **Multiplexer – Demultiplexer**. Switching buttons causes that appropriate tab is displayed. Each tab represents parameters that should be set in order to generate a proper structure. For *Multiplexer* tab there exist a few of possible structures. Choosing different multiplexer's size user can obtain different results. Also marking "use gates" checkbox enables generating structures with different gates: NANDs only, NORs only or ANDs+ORs+NOTs. In order to obtain demultiplexer (or tree of multiplexers) structure from *Demultiplexer* (*Multiplexer Tree*) tab user should select demultiplexer's size. The size of multiplexer and demultiplexer ranges from 2 bits to 16 bits, as these are sizes of manufactured commutators nowadays. *Multiplexer and Demultiplexer* tab has two parameters – sizes of

<sup>3</sup> A function can be implemented in the program if it is not constant 0 or 1 after minimization.

multiplexer and demultiplexer. As it is not always possible to generate a structure of a function of some number of variables (1 variable or more than 8) radio buttons that correspond to these impossible structures are disabled (Fig. 15). Also while clicking on the size whether of multiplexer or of demultiplexer, the second will change to enable proper generation of a Mux–Dmux structure.

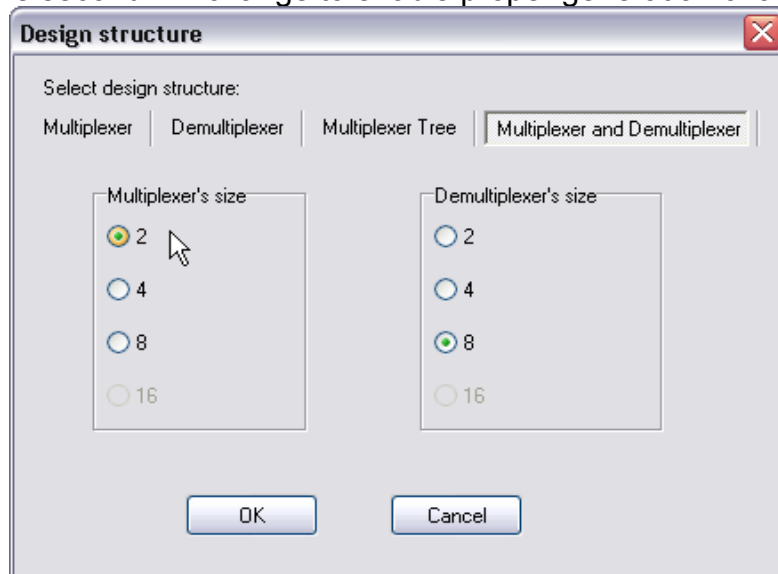


Fig. 15. Structure selection window with some options disabled

After applying the options by hitting "OK" button program generates a given structure and displays a diagram on the screen (exemplary result shown in Fig. 16).

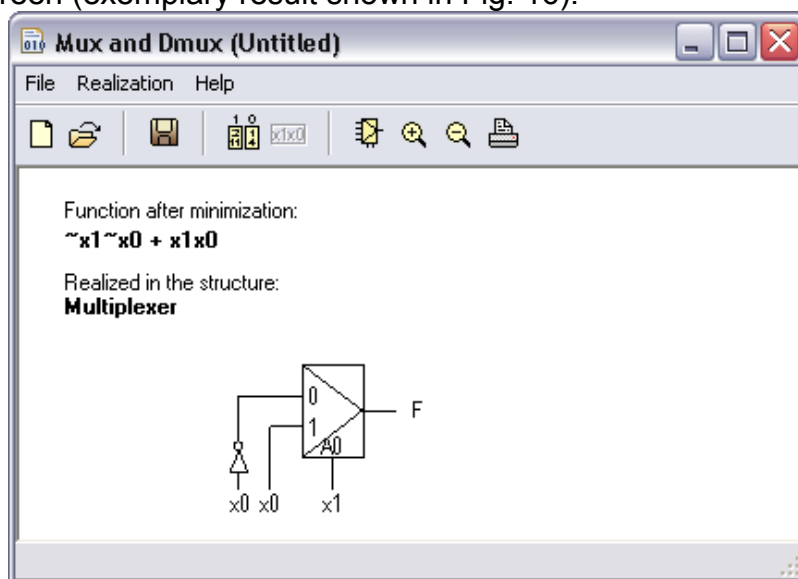



Fig. 16. Exemplary resulting structure

As it can be seen some toolbar options that were previously disabled are activated. The diagram can be zoomed in or out to facilitate viewing. The structure can be also changed by clicking **Realization** from the program menu or *realization* icon  from the toolbar.

### Saving and opening function definitions from a file

In order to save a function definition to a file a user should select **File** ⇒ **Save function as...** item. Choosing "**Save function**" option saves the current function to previously opened function definition file. If there was no previously opened file, while pressing "**Save function**", "**Save**

**function as...**" dialog window appears. In this dialog user can enter a file name to be saved to (Fig. 17).

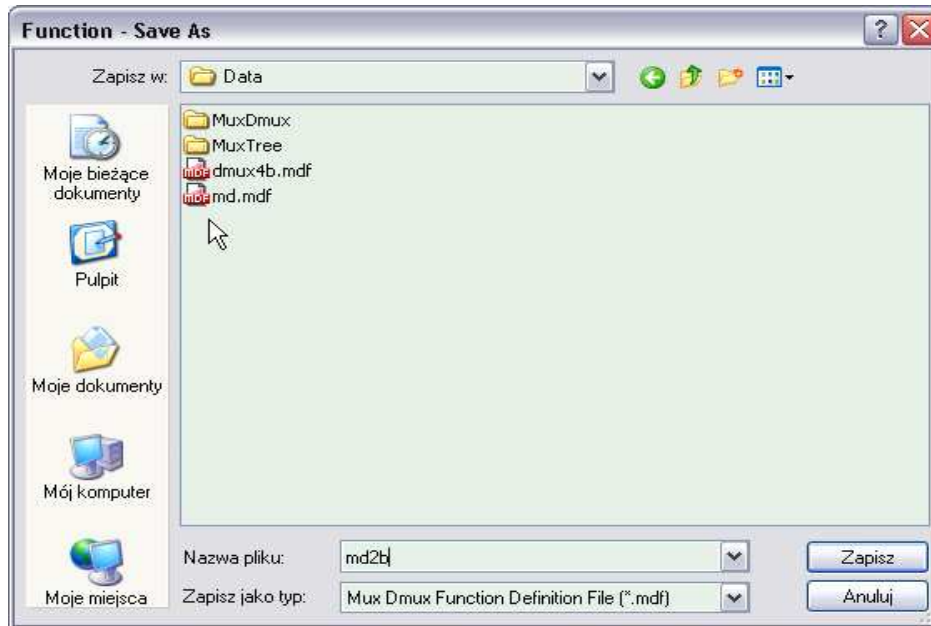


Fig. 17. Dialog window for saving a file

The file extension for function definition file is ".mdf". After the file is saved, at the bottom of the program window, on the status bar there appears the text "*File saved!*".

In order to open function definition from a file user should select **File** ⇒ **Open function**. Similarly to this in Fig. 17 the dialog window shows up where user can choose a file to be opened. If the current function definition in the program has been modified user is prompted to save it or discard these changes. Loading a function definition file enables the user to change the function or immediately implement this function.

In order to save project definition to a file user should select **File** ⇒ **Save project as...** item. Choosing "**Save project**" option saves the current function to previously opened project definition file. If there was no previously opened file the dialog window "**Save project as...**" appears where user can choose a file name to save to. The file extension for a project definition file is ".mdp". After the file is saved, at the bottom of the program window, on status bar "*File saved!*" text appears for three seconds.


In order to open project definition from a file user should select **File** ⇒ **Open project** item. Open dialog window (similar to Fig. ) appears where user can choose a file. If the current function or project definition in the program has been modified user is prompted to save it or discard these changes. After loading a project definition file the program immediately generates the structure saved in the file.

## 5.5. Export and print of a project

A user has two options while exporting the design. The realized structure may be saved in a graphical format as a diagram file or as a project file. The difference is that the diagram file

includes only diagram of a generated structure, while the project file contains a diagram as well as a minimized function definition. To export the generated structure user should select from program menu **File** ⇒ **Export diagram** or **File** ⇒ **Export project**.

The program enables exporting diagram/project to a vector (WMF) or raster (BMP, JPEG) graphic file format. For later use it is preferred to use vector format as it can be scaled without losing quality.

The project can also be printed on a printer set up in **File** ⇒ **Print setup**. In order to print a project user should select **File** ⇒ **Print project** from the program menu or click toolbar icon .

It has to be underlined that printing big structures (having more than five elements in one level) due to scaling may lead to illegible printouts. If user needs to obtain better results he is advised to use other graphical software and print the project on multiple pages.

## 5.6. Learn by example

This tutorial shows a step-by-step instruction of entering exemplary function and obtaining structure realizing this function.

### Goal:

*Implement the function  $F = (1, 2, 4, 12, 13, 14, 15, 18(6, 7))_{x_3, x_2, x_1, x_0}$  in a tree of multiplexers structure, where size of each multiplexer is 8 bits.*

Executing the program causes the window (Fig. 18) to be displayed.

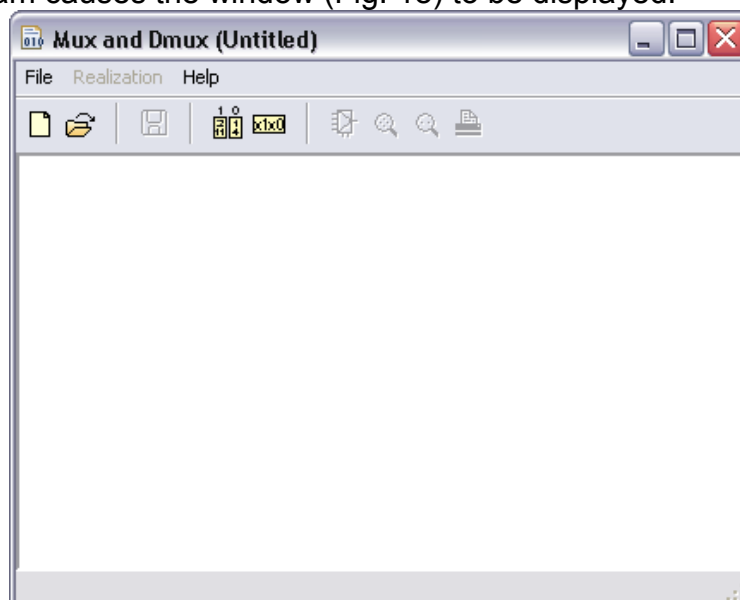

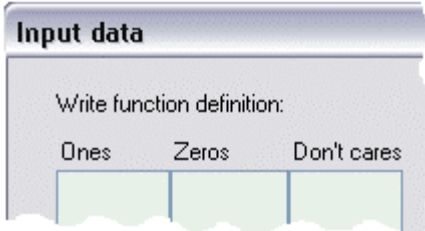


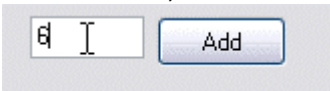
Fig. 18. Main program window (tutorial)



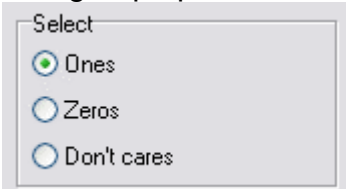
The function definition should be entered to the program in i.e. canonical numerical form, thus a user should click  button or select appropriate menu options – **File** ⇒ **New function** ⇒ **Canonical representation**. Window shows up, as in Fig. 19. The window consists of: Three lists of values: *Ones, Zeros, Don't cares*,



Text edit field,



Radio group options selecting to which list the value entered in the text field should be put,



Buttons: **"Add"**, **"Clear All"**, **"OK"**, **"Cancel"**.

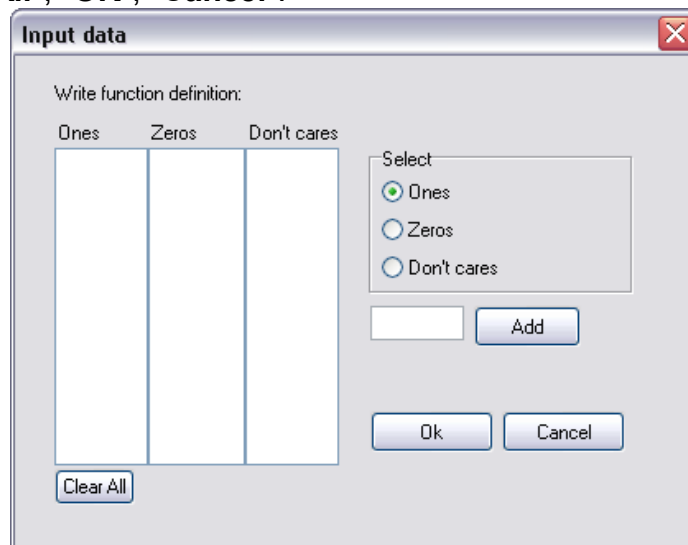


Fig. 19. Window that enables entering function definition in canonical form (tutorial)

To enter function ones components user is required to do:

Select to which list the value should be added, by clicking on one of the radio buttons, thus click **"Ones"**,

In text field individual one component should be entered,

Button **"Add"** should be clicked.

Steps 2 and 3 should be repeated until all ones components are entered. In case of mistake (i.e. misspelling, wrong list selection) user can delete the value entered by selecting this value and clicking right mouse button, and choosing **"Delete"** from popup menu, as shown in Fig. 20.

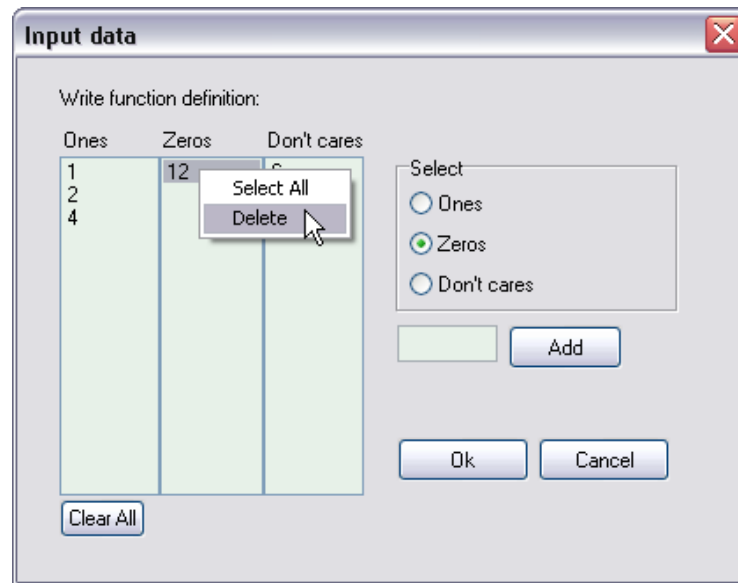



Fig. 20. Deleting term from the list (tutorial)

In order to enter don't care states the same algorithm is used apart from the first step. Here, user should select other radio group button by clicking "*Don't cares*" button. When all function components are entered the window should look similar to this shown in Fig. 21.

To apply the changes user is required to click "**OK**" button.

Beginning of function implementation process is achieved by clicking **Realization** from the program menu or *realization* icon  from the toolbar. The window shows up, where user should select required structure (Fig. 22).

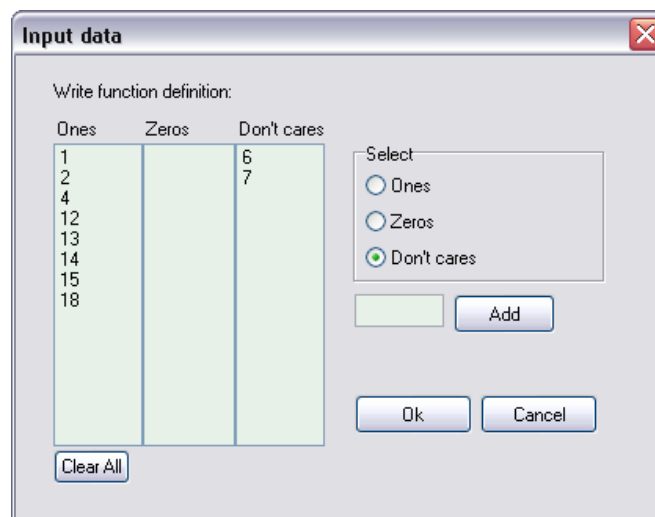


Fig. 21. Window showing all values properly entered (tutorial)

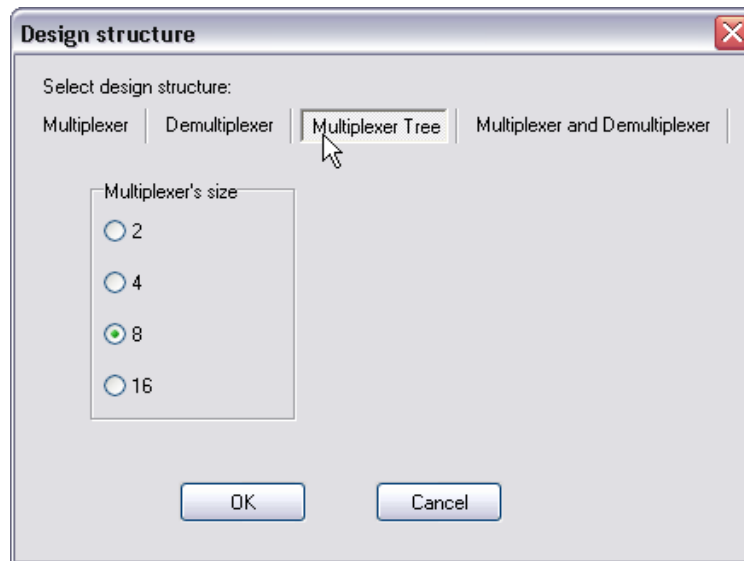


Fig. 22. Structure selection window (tutorial)

As it is shown the only one parameter to set, in order to generate multiplexer tree structure, is the size of a multiplexer. Therefore user should click **8**, as it is stated in the goal of this example. Then "OK" button should be clicked to generate the required structure. The window shows up (Fig. 23).

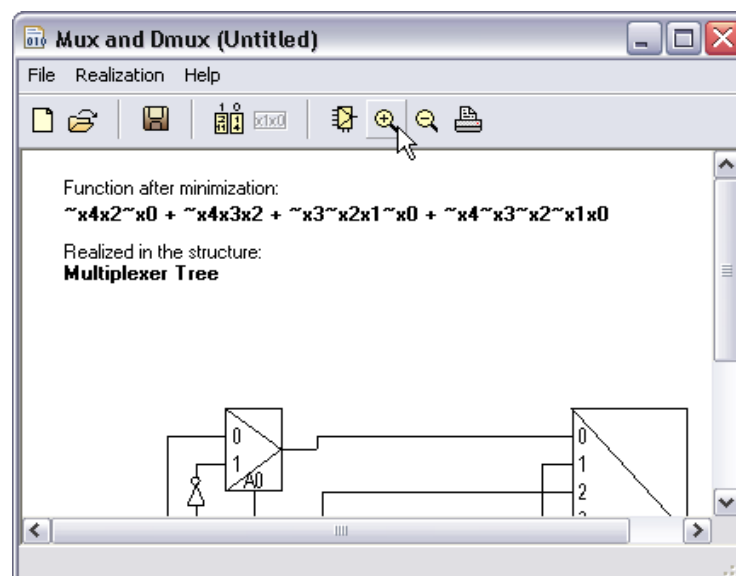




Fig. 23. Window showing the structure generated (tutorial)

Since not full structure is visible in the window it is necessary to either change size of this window or zoom in  or zoom out . Fully visible structure is presented in Fig. 24. As it can be seen only one 8-bit multiplexer is needed, remaining multiplexers are 2-bit.

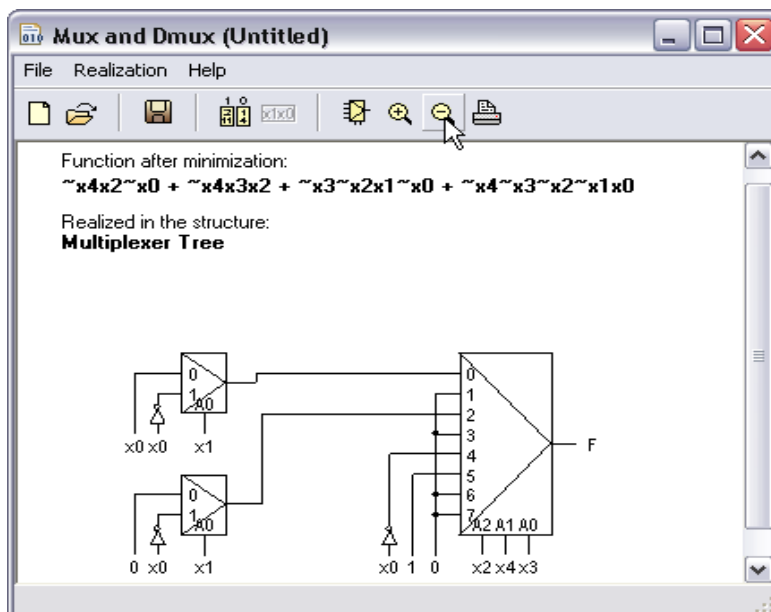




Fig. 24. Window presenting fully visible structure (tutorial)

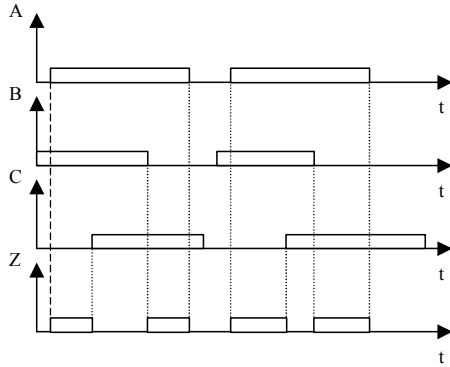
The goal of the tutorial is completed so it is advised to save the project for later use, print it or export it to graphical format. Saving may be performed in two ways, as a project or as a function. When user needs to save the function definition only, he discards the diagram generated. In the case when the structure is obtained, a better solution is to save data in project definition file, by selecting **File** ⇒ **Save project as...** or toolbar icon . In the dialog box appeared the user needs to supply the file name.

The project can also be printed or a printer can be set up in **File** ⇒ **Print setup**. In order to print a project user should select from program menu **File** ⇒ **Print project** or click toolbar icon . It is also useful to export the generated structure and use it in any graphical program or word processor. The structure may be saved in a graphical format (WMF, BMP, JPEG) as a diagram file (only structure) or as a project file (structure and function definition). To export the generated structure user should select from program menu **File** ⇒ **Export diagram** or **File** ⇒ **Export project**.

## 6. Tasks to be performed during laboratory

1. Using Table of Switching Sequence software, find function describing output signal Z behavior depending on input signals A, B, C, given by following time diagram.  
Draw Table of Switching Sequence for this example.  
How many added elements are necessary for this example? Why?





- Using Table of Switching Sequence software, find function describing output signals Y and Z behavior depending on input signals A and B, given following formula of connections:

$$A+B+Y+B-Z+A-Z-A+Y-A-$$

How many added elements are necessary for this example? Why? Define briefly, what “added elements” means.

- Using Table of Switching Sequence software, try to invent and solve your own example, using formula of connections or time diagram, for minimum 6 variables and 10 states. An extra bonus is for reports with working circuit enclosed.
- Describe solution steps for function F, given by canonical form, during the lab.  
Use Kazakow algorithm. Enclose detailed algorithm steps in report.
- Prepare implementation diagram for function F, given by the exercise supervisor during the lab, using MUX and DMUX elements in some of available structures. Use MuxDmux software.

## 7. Instructions to follow

- Get know with the methods utilized by the software (SST, Kazakov, solving using MUXs & DMUXs).
- Solve tasks given by supervisor.
- Present solutions to your supervisor for acceptance.