Method for simulation of digital automata

By T. VELITCHKOV and K. BOYANOV

Simulation of digital automata is more and more widely applied in designing of digital devices, as it permits full functional testing before their building. The methods for simulation known so far may be classified into two groups. The first one includes the methods for simulation based on the algorithm describing the behaviour of the tested devices [1], [2]. The second group includes the methods based on the logical equations, describing the components the digital device consists of [3], [4]. Some of these methods give an idea of the time sequence of the signals of the real device, others do not. The methods of the first group can be easily applied when the simulation of the automaton as a whole is required, while those of the second group are convenient when the simulation of some components of the automaton is necessary. The existing methods for simulation have some inconveniences. In the detailed examination of the behaviour of a digital automaton the volume of the data processed by the simulating programme and the machine time required, grow much faster than the increase of complexity of the automaton. Due to that it is accepted that the whole automaton consists of few, but complex components. This approach does not allow the detailed study, which is often required.

This paper proposes a method permitting the detailed examination of the behaviour of each of the structurally described components when the simulation of an algorithmically described complex digital automaton is carried out. The method makes possible the determination of the output signals of the automaton for every time interval for which the input signals are determined. Moreover, the simulating programmes remain unchangeable for different structures of the simulated digital automaton and for the various input signals.

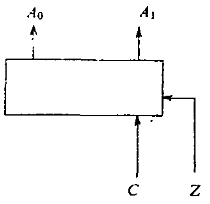

Fig. 1

Basic definitions

We define a finite automaton according to [5]. Logical network for us is a multitude of finite automata with connected inputs and outputs. We accept that time is divided into equal elementary time intervals during which the input and the output symbols of the automaton remain unchangeable and

$$T = \frac{t}{\tau}$$

where t is a real time interval, τ is the elementary time interval and T is a number without dimensions. We call T *delay factor*. The multitude of all input and output symbols assigned for an elementary time interval to the automata the logical network consists of is called *momentary state*. *Time diagram* is the multitude of momentary states for the time interval $t=n\tau$ where $n>1$. We classify the internal states of the

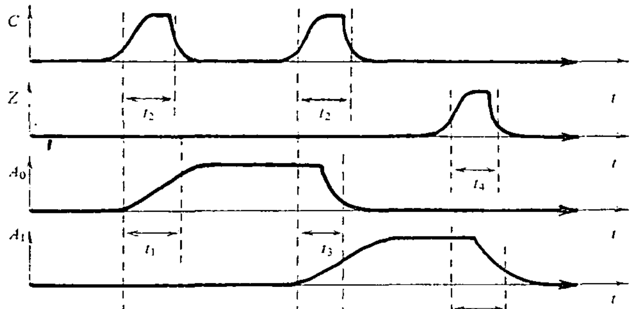

Fig. 2

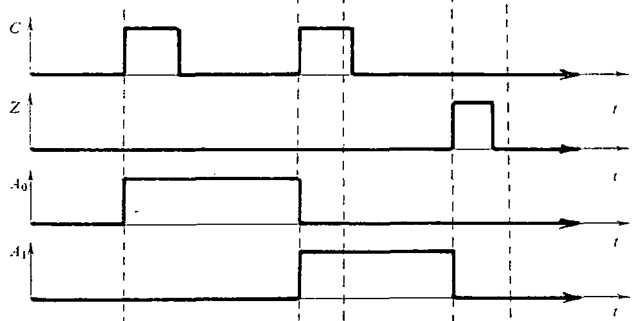

Fig. 3

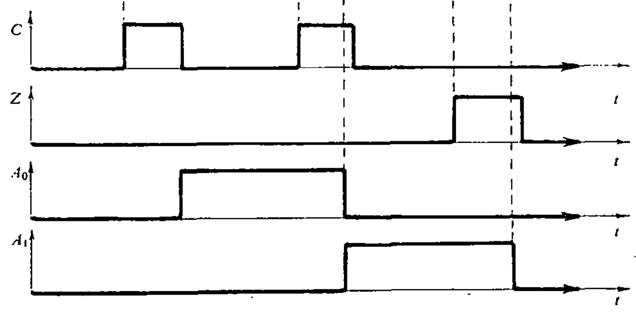

Fig. 4

automaton into two groups: *stable states,* in which the automaton can stay during more than one successive time intervals and *quasistable states,* in which the automaton cannot stay during more than one elementary time interval. We call *signals* the physical phenomena representing the input or the output symbols of the finite automata.

Essential ideas

We will try to represent the real automata so as to obtain the time-diagram of a logical network and its components.

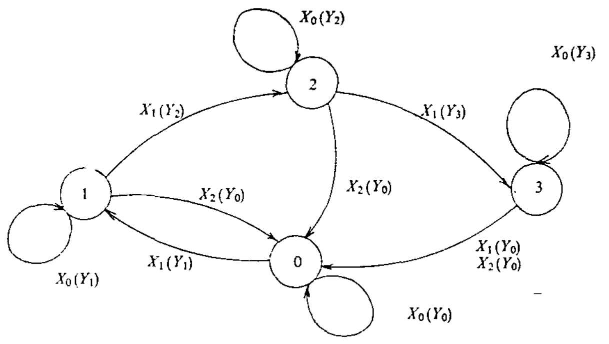

Fig. 5

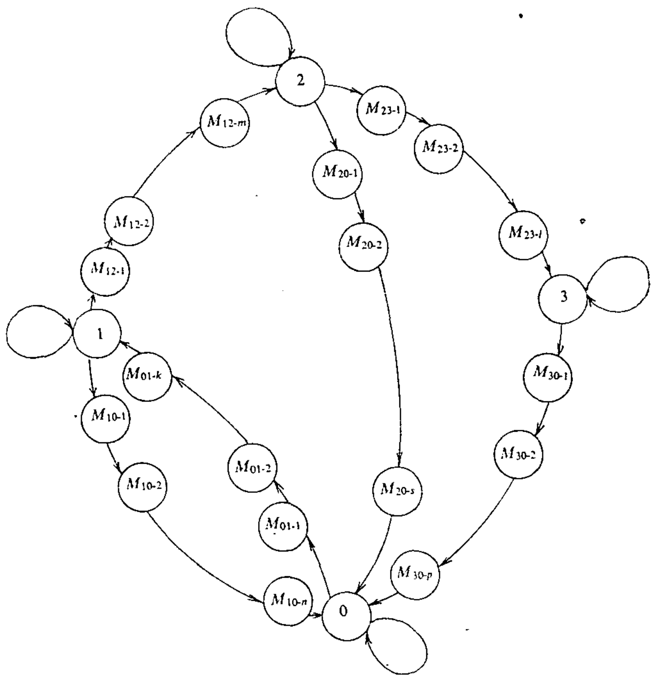

Fig. 6

Let us consider a real automaton having two inputs C and Z and two outputs A_0 and A_1 (fig. 1), whose input and output symbols are represented with real signals. The time-diagram of the automaton is shown on fig. 2. It can be seen that during the time intervals t_1, t_2, t_4 and t_5 the output state of the automaton is undetermined.

We will approximate the behaviour of the real automaton to this of a finite automaton, whose graph is shown on fig. 5 and whose time-diagram is given on fig.3.

Both the input and the output alphabets of the automaton are shown on fig. 14. (The symbol Δ will be discussed later.)

On fig. 3 it can be seen that there is no undetermined in the time-diagram. However, it does not show the time delay between the input affectations and the output reactions of real automaton (time intervals $t_1 \div t_5$.)

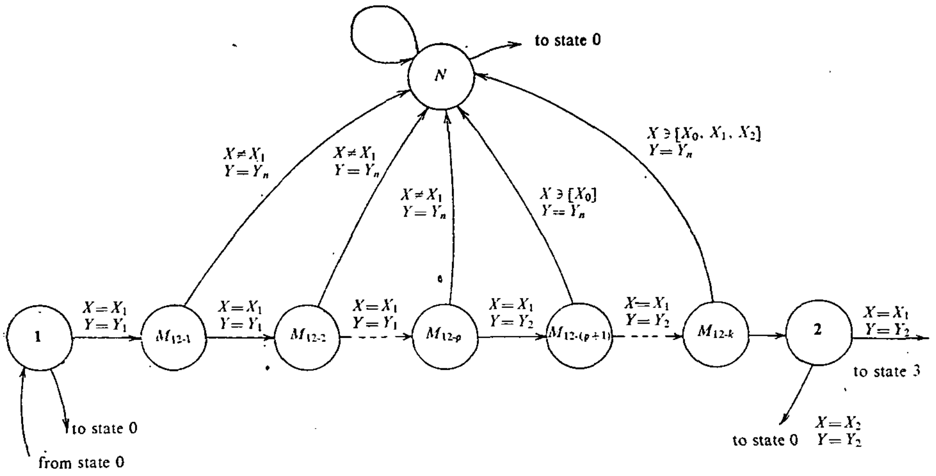

Fig. 7

The time-diagram on fig.4 approximates permissibly precisely the time-diagram given on fig.2. Fig.6, fig.7 and fig.8 show the graphs of automata meeting the requirements of the time-diagram on fig.4. The automaton defined by the graph on fig.6 analyses its own input symbol for every elementary time interval and according to its internal state provides the corresponding output symbol. The stable states of the automaton (0, 1, 2 and 3) are separated by sequences of quasistable states in which the automaton can be during the time intervals $t_1 \div t_5$. The number of the quasistable states is obviously dependent on the duration of the corresponding time interval. The

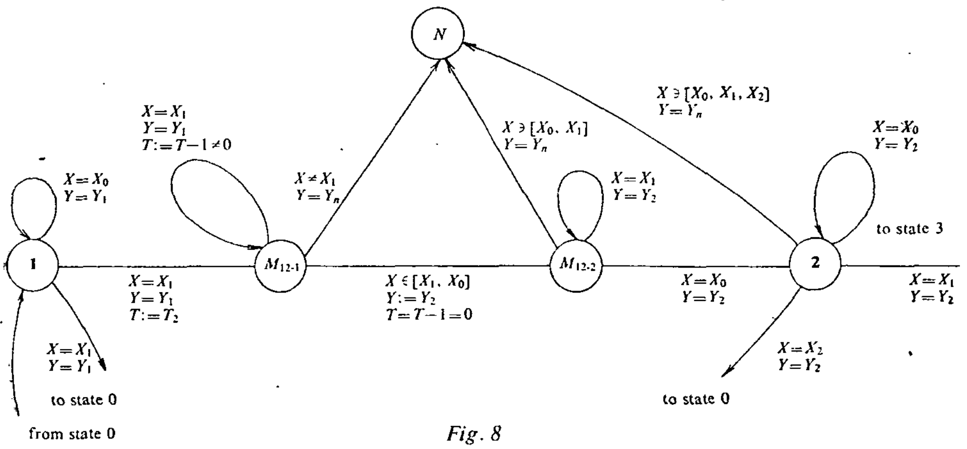

Fig. 8

output symbol of the automaton being in a quasistable state is identical with its output symbol of the latest stable internal state it has been into. The described automaton carries out the time-diagram from fig.4, but in this representation the number of the internal states is too great and this makes difficult the representation and the simulation of complex automata. Besides that, two functionally equivalent real automata with different delay factors will have a different number of internal states and this is quite inconvenient. Moreover, it don't makes possible the checking of the correctness of the input symbols. This inconvenience can be avoided by the introduction of an additional stable state N (undeterminated state, fig. 7). Before getting into any successive state (regardless of its being stable or quasistable) the automaton analyses its input symbol and in case it proves to be unacceptable for the given internal state, the automaton gets into the undeterminated state N and supplies the undetermined output symbol Y_n. The automaton remains in the state N till some preliminarily defined input affectation does not make it change its state. For the automaton on fig.1 this is the input affectation X_0. Nevertheless, the number of the internal states of the automaton is still great in this representation. In order to avoid this inconvenience the automaton is represented on fig.8 by a graph where the successive quasistable states are joined into new stable states M_{12-1}, M_{12-2}... The automaton will remain in these new states for as many elementary time intervals as is the duration of the intervals $t_1 \div t_5$. We consider this representation of the real automata to be enough convenient and will use it in simulation of digital devices.

Simulation

To simulate the work of logical network means in fact to simulate the work of the automata it consists of. Not the real automata are considered, but their equivalent finite automata, which are represented in the manner described above. A multitude of variables $\mathscr{L}$ is introduced for the simulation of the work of a finite automaton.

arguments a_1	a_2	conjunction	disjunction	nor operation	nand operation	exclusive or	exlusion of a_1
0	0	0	0	1	1	0	1
0	1	0	1	0	1	1	1
1	0	0	1	0	1	1	0
1	1	1	1	0	0	1	0
0	Δ	0	Δ	Δ	1	0	1
Δ	0	0	Δ	Δ	1	Δ	Δ
1	Δ	Δ	1	0	Δ	Δ	0
Δ	1	Δ	1	0	Δ	Δ	Δ
Δ	Δ	Δ	Δ	Δ	Δ	Δ	Δ

Fig. 9

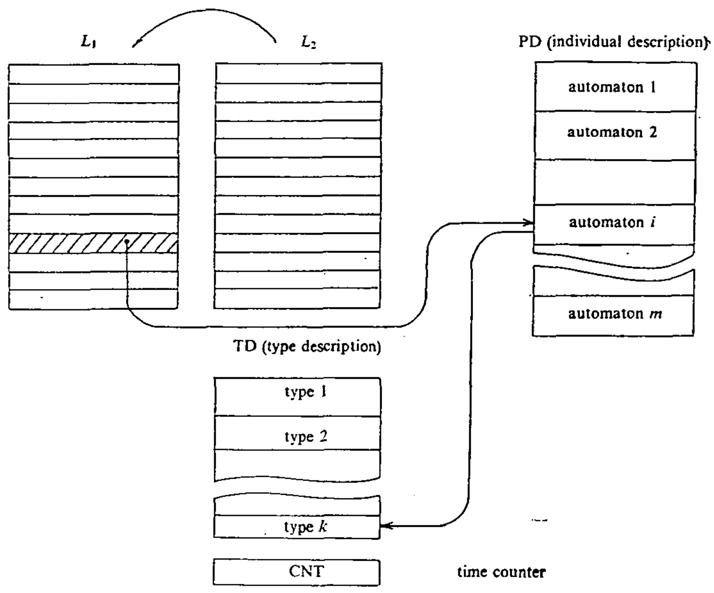

Fig. 10

These variables correspond to the input, output and internal states of the automaton and will be called *internal variables* of the automaton. The variable representing the input state can have many values as is the number of the symbols of the input alphabet of the automaton and its value in any given moment corresponds to the actual input symbol. The internal and the output states are represented in the same way. The multitude of the internal variables includes also variables corresponding to the delay factors, as well as service variables required by the simulating system. As the functioning of every type of automata is individually simulated, it is necessary to define all its stable and quasistable states, the incorrect input affectations, which will be looked for, when the simulation will be carried out and also to define the internal variables of the automata and to work out the algorithm of their changes. For the simulation of a logical network all the automata it consists of should be described and all their connections should be pointed out. It is also necessary to determine the initial state of the network. This presents a difficult problem when complex logical networks are considered. That is why this method accepts that the input (and output) symbols are three : 0, 1 and Δ. The Δ state is undetermined (0 or 1) and it is assign-

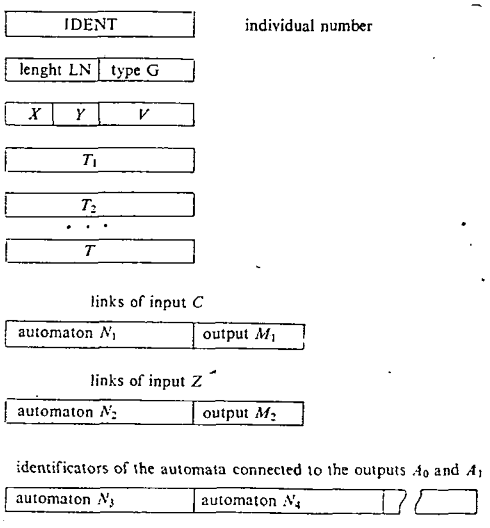

Fig. 11

ed to any input (or output) whose state cannot be precisely specified, as well as to the outputs of the automata, which have received an incorrect input affectation (on the graph on fig.5 and fig.6 this state is marked by Y_n). The introduction of this third state demands that the truth table of the logical functions should be changed (we accept that all logical functions are represented in a classical *and, or, not* basis). The truth table of the elementary functions of two arguments shown on fig.9 illustrates this.

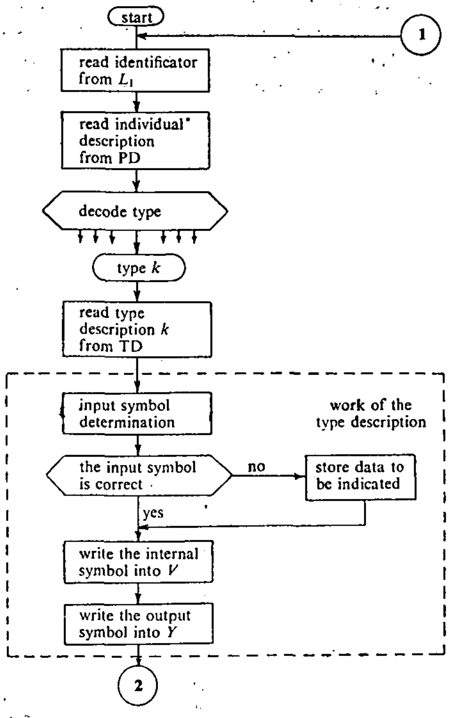
start
1
read identificator from L_1
read individual description from PD
decode type
type k
read type description k from TD
input symbol determination
work of the type description
the input symbol is correct
no
store data to be indicated
yes
write the internal symbol into V
write the output symbol into Y
2

Fig. 12a

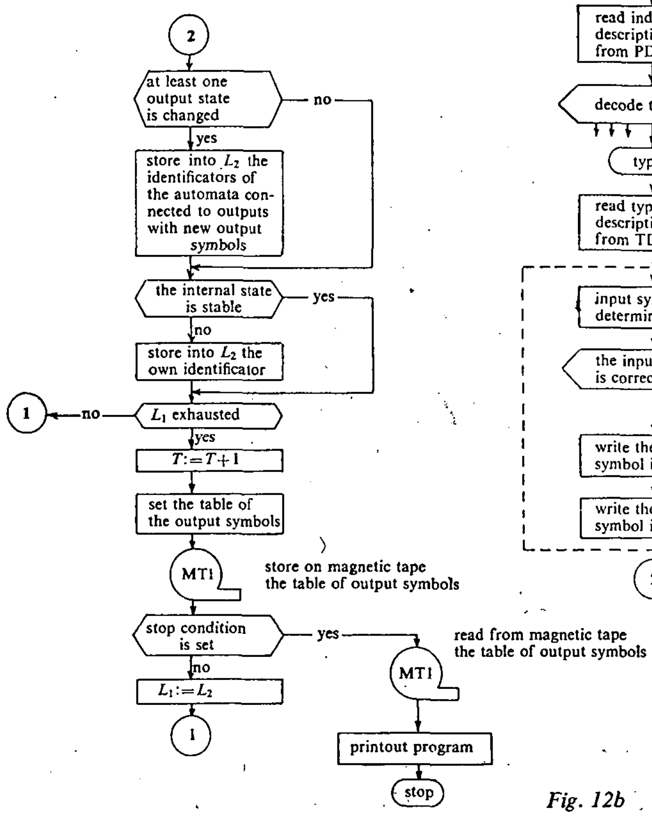
2
at least one output state is changed
no
yes
store into L_2 the identificators of the automata connected to outputs with new output symbols
the internal state is stable
yes
no
store into L_2 the own identificator
1
no
L_1 exhausted
yes
$T:=T+1$
set the table of the output symbols
MT1
store on magnetic tape the table of output symbols
stop condition is set
yes
no
$L_1:=L_2$
1
read from magnetic tape the table of output symbols
MT1
printout program
stop

Fig. 12b

Structure of the simulating system

This paper considers the simulating system carrying out the described ideas. The system includes the files TD and PD (fig.10), the lists L_1, L_2 and the counter CNT. It makes possible the simulation of the logical network, built of a finite number of types of automata, which have preliminarily assigned internal variables and algorithms. The programmes simulating the functioning of each type of automata are worked out according to their algorithms and are grouped into the file *type descrip-*

tions (TD). The internal variables of all automata, taking part in the network, form the *individual descriptions* of the automata and are grouped into the file PD. Besides that, every individual description includes the length factor (LN), indicating its own length, the identificator IDENT, showing the individual number of the automaton and the type factor G, denoting the type of the automaton (fig. 11). The connections between the components of the simulated network are shown by the identificators of the automata and by the numbers of the outputs connected to each input. (See fig. 11.) In order to examine a network it is necessary to define precisely the input affectations for every elementary time interval. This method achieves that by describ-

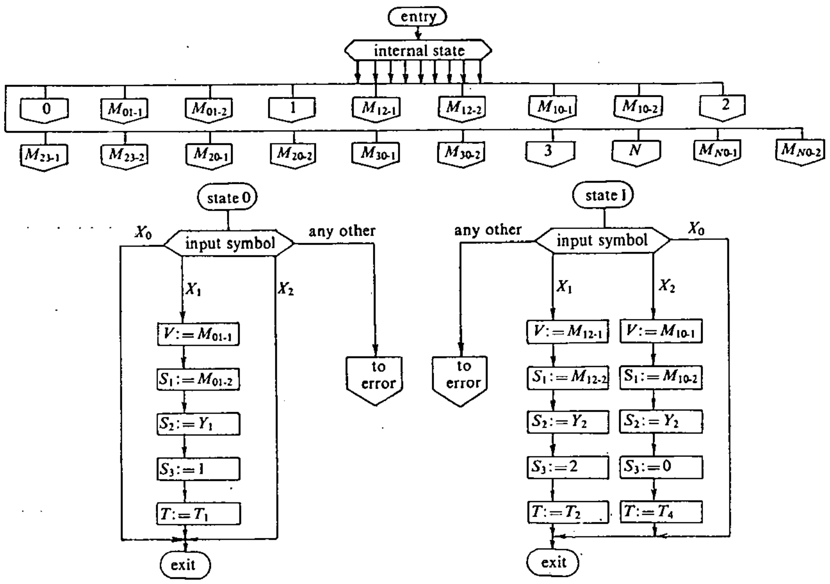

Fig. 13a

ing all input variables as autonomous automata, according to the definition given in [6]. The time counter CNT contains the number of the actual time interval. At the initial moment its value is 0, while all inputs and outputs are in the Δ state. The list L_1 includes the identificators of the autonomous automata, representing the input variables. The programme reads an identificator of an automaton from L_1, takes out its individual description from the file PD, decodes its type and reads its type description from the file TD. During the first elementary time interval this automaton will be an autonomous one whose output will be set in 0 or 1. After that the identificators of all automata connected to the output which has changed its state are put in L_2. In case the automaton has got into a quasistable state, its own identificator will be put in L_2. This process continues till the list L_1 is exhausted. Then the contents of L_2 is put in the place of L_1, "1" is added to the time counter and the study of the behaviour of the network for the next time interval begins. The list L_1 contains now the identificators of all the automata to whose inputs new symbols have been assigned, as well as the identificators of the automata being in a quasistable state. The rest automata are not studied because their states (input and internal) remain unchangeable. The general algorithm of the simulating programme is given on fig.12. After every elementary time interval "the complete table of output states" is filled with the new data, composed of the identificators and of the states of the outputs of all the automata the network consists of. This table is the result of the simulation and is

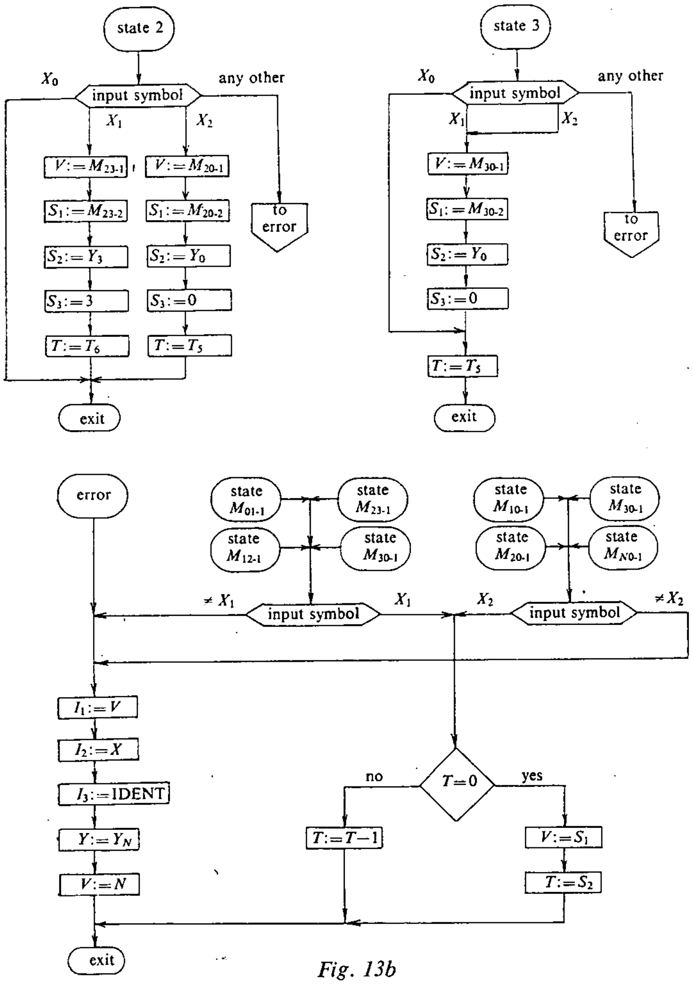

Fig. 13b

stored on a magnetic tape. The simulation will be carried out till some preliminarily specified conditions are detected. (Certain states of some automata or a given value of the time counter CNT.) The structure of the simulated network and the algorithm of the automata for each type are described in a proper language, which is not discussed in this paper.

Let us consider a counter, for example. Its time-diagrams (real and approximated) are shown on fig.3 and fig.4. The graph of the finite automaton, approximating the counter, is given on fig.8. (Only the stable states 1 and 2 and the quasistable ones linking them are shown; the other branches are similar to them.) The multitude of the internal variables becomes of the kind

$$\mathcal{L} \equiv \{V, X, Y, T_1, T_2, \ldots, T_i, T\}$$

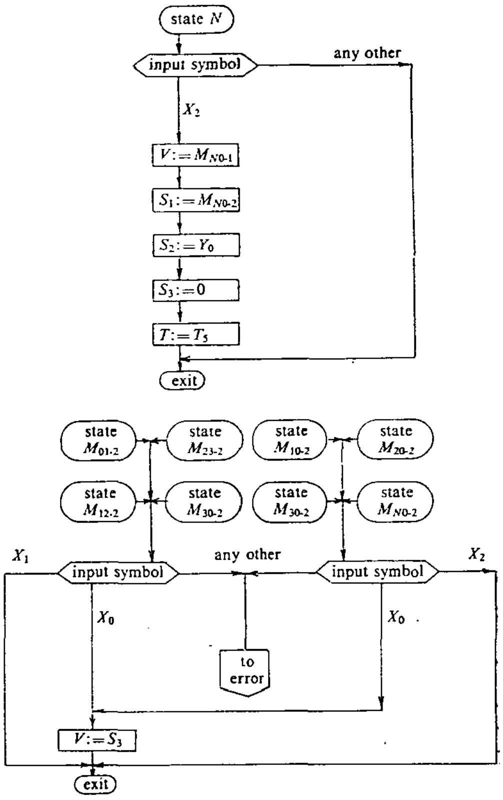

Fig. 13c

where:

V — is the variable of the internal state, it contains the number of the actual internal state;

X — is the variable of the input state, it points out the actual input symbol;
Y — is the variable of the output state, it points out the actual output symbol;
T_i, $i=1, 2, \ldots$ are delay factors;
T — is the variable of the actual delay.

input symbols	binary values of the input signals	
	C	Z
X_0	0	0
X_1	0	1
X_2	1	0
X_3	1	1
X_4	0	$\varDelta$
X_5	$\varDelta$	0
X_6	$\varDelta$	1
X_7	1	$\varDelta$
X_8	$\varDelta$	$\varDelta$

output symbols	binary values of the output signals	
	A_0	A_1
Y_0	0	1
Y_1	0	1
Y_2	1	0
Y_3	1	1
Y_4	0	$\varDelta$
Y_5	$\varDelta$	0
Y_6	$\varDelta$	1
Y_7	1	$\varDelta$
Y_N	$\varDelta$	$\varDelta$

Fig. 14

note: the symbols Y_4, Y_5, Y_6, Y_7 are not allowed

Examining the behaviour of the counter we accept that the input affectations cannot be shorter than t_1, t_2, t_4 (fig.4). Any input affectations which do not meet this requirement will be considered to be incorrect. Let us suppose that the counter is in the stable state 1 (fig.8). It will stay in it as long as the input symbol remains X_0. When the input symbol becomes X_1, the internal state will get into the quasistable state M_{12-1} (see fig.8), the variable of the actual delay (T) receives the value T_2 and the output symbol remains unchangeable. Being in the state M_{12-1} the counter analyses its input symbol, whose value should remain invariably X_1. Meanwhile "1" is subtracted from the current delay during every elementary time interval. When the value of T becomes 0 the counter gets from the state M_{12-1} into the state M_{12-2} and the output symbol changes from Y_1 to Y_2. The change of the input symbol while the internal state is still M_{12-1} means that the input affectation is shorter than T_2. In such a case the next internal state of the automaton will be N and the variable of the output state will receive the value Y_N. The automaton will remain in the state M_{12-2} as long as the input symbol is X_1. It will get into the state 2, when the input symbol is X_0. Any other input symbol will lead into the state N. The maximum length of the input affectation is not checked during the time interval t_3, because it is not limited in time. So far we have considered the branch of the graph connecting the stable states 1 and 2. All other branches are similar.

The algorithm of the counter described above is shown on fig.13. The variables S_1, S_2, S_3, I_1, I_2 and I_3 are service variables. They are not required by the graph but their use makes the algorithm simpler.

Conclusions

The method considered in this paper makes possible the simulation of logical networks, composed of large and expansible set of components. The simulating programme does not depend on the structure of the network and on the kind of the components it consists of. On one hand, the examined network could consist of only one algorithmically described automaton (except the automata representing the input variables). On the other hand, the structure of the network could be described in full details (it can be accepted that the network is composed of elements carrying out the elementary logical functions). This method permits the combination of these two extreme cases. Complex digital automata can be simulated in this way and the detailed examination of some of their parts is possible. Descriptions of new types of components can be easily added to the simulating programme system.

A full information about the functioning of the simulated automata is obtained as a result of the work of the programme. Moreover, race simulation or oscilation of the tested network can be registrated. Finally, by the lists L_1 and L_2 we can judge how often the various components of the tested nerwork are used.

The authors are indebted to ass. prof. D. Dobrev from the Institute of Mathematics and Mechanics of the Bulgarian Academy of Sciences for his valuable recommendations.

BULGARIAN ACADEMY OF SCIENCES
INSTITUTE OF MATHEMATICS AND
MECHANICS WITH COMPUTER CENTER
SOFIA, BULGARIA

References

[1] Боянов, К. & Т. Величков, Моделирование логических схем на цифровой вычислительной машине, *Доклады БАН*, т. 22, № 5, 1969.
[2] Breuer, M. A., Functional partitioning and simulation of digital circuits, *IEEE Trans. Computers*, v. C19, N11, 1970.
[3] Breuer, M. A., Techniques for the simulation of computer logic, *Comm. ACM*, v. 7, 1964, pp. 443—446.
[4] Shalla, L., Automatic analysis of electronic digital circuits using list processing, *Comm. ACM*, v. 9, 1966, pp. 372—380.
[5] Глушков, В. М., *Синтез цифровых автоматов*, Физматгиз, 1969.
[6] Трахтенброт, Б., *Конечные автоматы (поведение и синтез)*, Наука, Москва, 1969.

(Received April 13, 1974)