

*Cybernetics and System Analysis*, Vol. 33, No. 2, 1997, pp. 177-185

© 1997 Plenum Publishing Corporation

Translated from *Kibernetika i Sistemnyi Analiz*, No. 2, pp. 32-44, March-April, 1997,  
published by the V.M. Glushkov Cybernetics Institute of the Ukrainian Academy of Sciences.  
Original article submitted December 16, 1996.

## **FORMALIZATION OF LOGIC DESIGN OF THE REGISTER COMPONENTS IN DISCRETE SYSTEMS**

**A. A. Mishchenko and A. T. Mishchenko**

UDC 681.325.5; 658.512.011.56

Discrete hardware design involves at a certain stage the logic design of three types of components: functional components (combinational logic networks), operational components (registers), and control components (automata) [1]. Control automata design has been developed in considerable detail, while formal (and thus computer-aided) register design still involves difficulties, despite its apparent simplicity. In practice, register design uses various flipflops as storage elements, and each type of flipflops can be implemented by dozens of different logic circuits with different reliability characteristics and optimization options. Moreover, writing data to a register may involve more than ten different register operations. The designer thus has to choose the hardware solution from a large number of options, allowing for the time characteristics of the input signals, as well as requirements for speed, cost-efficiency, reliability, etc.

The present article is an attempt to propose a solution of this problem through the development of a formal register design procedure originally described in [2]. The designer, using his experience and intuition, allows for a variety of circumstances and constraints, selects some register design, and describes its operating algorithm in an accessible language. The register circuit is then constructed on the basis of this algorithm. The simplicity of the procedure enables the designer to foresee the outcome of the computer-aided design procedure. By formulating the register description, the designer is thus capable of making supplementary decisions and allowing for constraints in such a way that the end result is a reliable functioning unit.

We now proceed to present the algorithmic description language and the procedure for formal register design.

### **REGISTER STRUCTURE**

A register is a device that receives, stores, and delivers information at specified time instants. An  $n$ -digit binary number  $a_1a_2\dots a_n$  requires an  $n$ -bit register: each register bit contains

the corresponding digit of the binary number and is implemented using a single flipflop. A simplified register structure is shown in Fig. 1. The input and output lines link the register with other network components. Let us describe in general terms each part of the register, namely the flipflops and three combinational logic circuits: the term former, the input former, and the operation control former.

A flipflop is a two-state automaton with a complete system of transitions. A flipflop may have a different number of inputs and in general two outputs for signals that are one the inverted form of the other. This device can maintain one of two possible stable states, 0 and 1. The transition from state 0 to state 1 and back in a flipflop is triggered by input signals, which have the values 0 or 1. Different types of flipflops are used in design practice. Each flipflop type is characterized by its own state-change technique depending on the input signals. The different flipflop types are characterized using the standard notation for flipflop inputs, and the flipflop behavior is described by transition tables. Transition tables for common flipflops are shown in Tables 1a-h. The letter  $a$  in the Tables 1a-h denotes the flipflop state [3].

Table 1a shows flipflops of group RS (RS, R, S, E), and Tables 1b-h show flipflops of types D, T, DV, TV, JK, RST, DVT respectively. The dash in Table 1a indicates that the input combination  $R=1, S=1$  is forbidden in RS flipflops; all the input patterns not shown explicitly in Tables 1g and 1h are also forbidden. The last two tables describe combined RST and DVT flipflops, which function as a T flipflop for  $T=1$ , and as RS and DV flipflops, respectively, when 1 appears on the other inputs.

The information inputs R, S, D, T, J, K receive information that has to be stored; the control inputs V open ( $V=1$ ) or close ( $V=0$ ) the access to the flipflop through the information inputs. Clocked flipflops, in addition to information and control inputs, have special inputs that receive clock signals  $\tau$ , as well as setting inputs  $R'$  and  $S'$  that are used to reset the flipflop to state 0 and 1, respectively. Reset is performed, in particular, when the flipflop is cleared by a special signal  $\bar{N}$  before starting to operate; in the course of normal operation, the flipflop may be cleared by writing zeros through the information inputs.

A flipflop typically has two outputs. On one of the inputs (the 1-input) the signal  $a$  corresponds to the flipflop state, whereas on the other input (the 0-input) the signal  $\bar{a}$  is the inverted value of the first signal.

The term former (see Fig. 1) is an input stage that receives information from other registers and devices. The incoming information is written into the flipflops by one of the write operations (many such operations may be supported in flipflops). Each operation  $i$  is executed

when the register receives the signal  $U_i=1$ , which is the condition for the execution of the corresponding operation. This condition is formed in one of the control devices. When  $U_i=0$ , the operation  $i$  is not executed.

We denote by  $f_i^j$  ( $\bar{f}_i^j$ ) the information written by operation  $i$  in register bit  $j$ . This operation is received from 1-output (respectively, 0-output) of other devices. Then the direct term  $B_i^j$  and the inverted term  $C_i^j$  of operation  $i$  are defined by the following expressions:

$$B_i^j = U_i \& f_i^j \quad (1)$$

$$C_i^j = U_i \& \bar{f}_i^j \quad (2)$$

The register term former consists of two-input AND elements that implement the direct and inverted terms of all the write operations in the register. The term former filters the information reaching the register inputs and passes to the flipflops only the information that has to be written at the current time instant. This occurs when some condition  $U_i$  takes the value 1.

The input former (see Fig. 1) forms in each register bit  $j$  for each information input (R, S, D, T, J, K) input signals according to the expression:

$$\bigvee_{i'} B_{i'}^j [C_{i'}^j], \quad (3)$$

which includes either the direct term  $B_{i'}^j$  or the inverted term  $C_{i'}^j$ , depending on whether the bit  $j$  requires a direct or an indirect signal during operation  $i'$  (this is reflected by square brackets in formula (3));  $i'$  in (3) takes only the value of the indices  $i$  of the operation that involves the bit  $j$ . The input former thus combines (by means of an OR element) on each information input all the signals that should be delivered to the given flipflop input.

The write control former (see Fig. 1) performs two functions.

1. If some signal  $U_i$  is generated in several automata (and not in one automaton), the write control former produces the control signal  $U_i$  for each write operation  $i$  in the form of the disjunction

$$U_i = \bigvee_k U_i^k, \quad (4)$$

where  $U_i^k$  is the operation- $i$  control signal received from each automaton  $k$  where the given signal  $U_i$  is generated.

2. If the register employs DV or TV flipflops, then for each bit  $j$  the control signal  $V_j$  is determined by the formula

$$V_j = \bigvee_{i'} U_{i'}, \quad (5)$$

where  $i'$  takes the value of the indices of all the operations that write information into bit  $j$ .

## **TYPES OF OPERATIONS EXECUTED IN THE REGISTER**

The outcome of some signals reaching the register inputs depends on the pattern of input signals, the register contents (which in particular may be zero), and on the type of flipflops used in the register. When an  $n$ -digit binary number  $F=f_1f_2\dots f_n$  reaches the inputs of register R, its impact on the register contents  $A=a_1a_2\dots a_n$  may be viewed as an operation on two  $n$ -digit binary number. The operations have a different number of arguments and require a different number of terms for their execution. Table 2 lists the possible operations and the flipflop excitation modes realizing these operations for each type of flipflops used the register.

The first column in Table 2 gives the operation name; the second column gives the operation result  $a$ . The subsequent column give the phase structure of the operations for different flipflops. The phase structure of an operation is the pattern of direct and inverted terms on the flipflop inputs, constructed using (1) and (2). A dash indicates that the corresponding term  $B$  or  $C$  is not realized on the flipflop inputs. An empty entry indicates that the corresponding operation is not directly realizable automatically because of the properties of the D flipflop. When this operation is realized on DV or DVT flipflops, the term  $C=U_i$  must be included in expression (5) for the control signal V. The symbol T in bitwise mod-2 addition operation for JK flipflops indicates that during the execution of these operations the JK flipflop functions as a T flipflop, and thus the term  $B$  or  $C$  corresponding to the symbol T should be included in expression (3) for both input J and input K. Forced writing of zeros or ones is generally performed by zero signals (described by inverted conditions) on the special inputs  $R'$  and  $S'$ . We assume that these operations may be necessary with flipflops of any type, which requires corresponding  $R'$  and  $S'$  inputs in all flipflops.

Here we consider only the write operations into a register, and do not consider the read operations from a register. The latter may be interpreted as write operations in blocks whose input signals are the output signals from the register described above.

Let us note some features of the operations in Table 2. These features may be taken into consideration by the designer in formulating the algorithmic descriptions of the devices for design optimization by speed and hardware costs.

1. The one-phase direct and inverted code write operations  $f_i$  and  $\bar{f}_i$  in the same bits of register R can be executed simultaneously for several numbers, which is equivalent to bitwise disjunction of these numbers.

2. The bitwise disjunction operations  $a \vee f$  and  $a \vee \bar{f}$  in the same bits of register R can be executed simultaneously for several numbers, which is equivalent to bitwise disjunction of all these numbers with the contents of register R.

3. The bitwise conjunction operations  $a \& f$  and  $a \& \bar{f}$  in the same bits of register R can be executed simultaneously for several numbers, which is equivalent to bitwise conjunction of all these numbers with the contents of register R.

4. For T and TV flipflops, despite the specific structure of their transition tables, the one-phase write operations  $f_i$  and  $\bar{f}_i$  may be executed in the same way as for RS and other flipflops, because the register is cleared before these operations. As a result, information can be passed to counters consisting of T and TV flipflops as into registers with RS or DV flipflops.

### **ALGORITHMIC DESCRIPTION OF THE REGISTER (REGISTER CHART)**

The register chart (Table 3) presents in a concise format the information that characterizes the register design in general (the top row of the table), and also describes all the information reception operations in the register (the following lines).

Let us describe the contents of different cells in the register chart. We start with the top row in Table 3, which contains the general characteristics of the register:  $R_n$  is the register name;  $j_1 \wedge j_2$  is the number of register bits (any ascending or descending interval of numbers from  $j_1$  to  $j_2$  is allowed, with numbering starting from 0 to 1, or from any other value); the type of flipflop circuit used in the register (we assume that flipflops of the same type are used in all bits);  $V_{ff}$  is the serial number of the flipflop circuit selected from the recommended list (if the same flipflop circuit is used in all register bits; otherwise, the correspondence between register bits and flipflop circuits should be specified outside the register chart);  $T_{sw}$  is the flipflop switching time (the switching time for the flipflop second stage);  $Q_{op}$  is the number of operations executed in the register;  $Q_{ing}$  is the number of operations with signals that require more than one time step;  $Q_{in}$  is the number of register inputs;  $Q_{out}$  is the number of register outputs. The contents of some of the cells becomes known when the register design is completed.

The second row of the register chart contains the following column headers (see the example in Table 4);  $i$  is the serial number of the operation (0 is conveniently associated with

forced register clearing);  $U_i$  is the execution condition for operation  $I$  (the operation is executed when  $U_i=1$ ; it is not executed when  $U_i=0$ );  $J_i$  are the indices of register bits that take part in operation  $i$ ;  $F_i^j$  is the list of names of input signals arriving at the register during operation  $i$  and written in the same sequence as the corresponding bit indices in column  $J_i$ ;  $a_i$  is the type of operation  $i$  described using the same notation as in column  $a$  of Table 2.

The indices of the bits taking part in operation  $i$  are listed as a sequence of natural numbers depending on the numbering of the register bits — whether from lower to higher order, or from higher to lower order. To avoid complicated notation, we use  $j_1 \wedge j_2$  to represent the intervals of ascending or descending sequences of consecutive natural numbers, starting with  $j_1$  and ending with  $j_2$ . Thus, for instance, instead of writing 1, 2, ..., 19, 20, Table 4 contains  $1 \wedge 20$ .

When describing a sequence of input signals  $f_i^j$  that differ only by their indices  $j$ , the common part  $f_i$  is written outside parentheses, and the parenthetical part contains only the series of indices  $j$ . If possible, the same abbreviation is used for these indices as for the description of  $J_i$ .

The register chart is completed when we finish describing all the register operations. Filling the columns  $B_i$  and  $C_i$  is the first stage of register design.

Let us consider a register chart (see Table 4) for a 20-bit register  $R_1$  consisting of RS flipflops. The zeroth row ( $i=0$ ) in the table describes forced clearing of the register by a special zero signal  $\bar{N}$ , which is delivered to the flipflop setting inputs  $R$ .

The first row ( $i=1$ ) also describes a register clear operation; this is the operation executed in run time when the condition  $U_1=1$  is received. To realize this operation in the register, the inputs  $R$  of all flipflops should realize the inverted term  $C$ , which equals  $U_1$  when writing the constant 0 by (2) (see Table 2).

The next row ( $i=2$ ) describes the operation of one-phase direct-code write on the condition  $U_2=1$  in all 20 bits of the register  $R_1$  from some register  $D$ . The information  $d_j$  from each bit  $j$ ,  $j=1, 2, \dots, 20$ , is received in bit  $j$  of register  $R_1$ . The realization of this operation, according to (1), requires the presence of the direct term  $B_2^j = U_2 \& d_j$  on the input  $S_j$  (see Table 2).

In row  $i=3$ , paraphase direct-code write is used to represent a one-bit cyclic shift of the contents of register  $R_1$  toward bits with lower  $j$ . The information from bit 1 is rewritten into bit 20, and information from bits 2 through 20 is shifted to the neighboring bits.

Finally, the last operation ( $i=4$ ) is executed differently in different parts of the register. The same condition  $U_4=1$  triggers in bits 5<sup>^</sup>10 paraphase inverted-code write of the contents of bits 5 to 10 of some register B, whereas in bits 11<sup>^</sup>20 it triggers bitwise disjunction of the contents of register  $R_1$  with the contents of the first ten bits of some register C. Bits 1 to 4 in register  $R_1$  do not take part in this operation.

## FORMAL REGISTER SYNTHESIS FROM REGISTER CHART

The objective of the synthesis procedure is to construct logical expressions for term formers, input formers, and write control formers which, jointly with the flipflops listed in the register chart, produce the device specified by the register chart. The synthesis process consists of the following stages.

1. First the phase structure of the operations is entered in columns  $B_i$  and  $C_i$ . It is obtained from Table 2, given the operations and the flipflop type in the register.

2. Then for each register bit  $j$  and for each information input ( $R_j, S_j, D_j, T_j, J_j, K_j$ ) and setting input ( $R_j^-, S_j^-$ ), we find the disjunction (3) of the direct (1) and inverted (2) terms, which are included in (3) only in application to each operation  $i^-$  that contains the given bit  $j$  in column  $J_i$ . A nonzero direct term  $B_i$  [or inverted term  $C_i$ ] is thus included in (3), or no term is included, depending on the phase structure of the operation.

3. For the control inputs  $V_j$  of DV, TV, or DVT flipflops of each bit  $j$ , we find expressions of the form (5) and include the conditions  $U_i$  in these expressions. This should take into account the properties of writing the constant 0 into the DV flipflop reflected in Table 2.

4. For each operation- $i$  controlling condition  $U_k$ , which is generated in more than one control automaton, we find an expression of the form (4). To automate this procedure, the symbols of the components  $U_i^k$  combined in each disjunction (4) should differ only by the index  $k$  of the control automaton generating the components  $U_i^k$ .

5. In general, there is no need to design the flipflop circuits, because they are normally selected from the available range of circuits in a given component base. When specifying the register chart, the designer should select the flipflop type to be used in the register and choose a specific flipflop circuit, if several are available. Given the range of register operations, the designer selects the flipflop circuit that meets the requirements of reliability, speed, hardware costs, etc.

## REGISTER SYNTHESIS EXAMPLE

We will demonstrate the proposed procedure in application to the synthesis of a register specified by the register chart in Table 4. The flipflop type is determined by the fact that the sought register  $R_1$  should include the bitwise disjunction operation ( $i=4$ ), which is directly executable (without an additional former realizing  $a_j \vee f_j$  in each bit) on RS flipflops (see Table 2). When selecting the RS flipflop circuit, we should note that the register shift operation ( $i=3$ ) requires two-stage flipflops (see [2]) or similar circuits, which are capable of reliably reading and writing information at the same time. We also assume that the register speed requirements permit using a flipflop circuit that contains one information input R, one information input S, and one setting input  $R'$ . The forced clear operation ( $i=0$ ) delivers the reset signal  $\bar{N}$  to the setting inputs, i.e.,  $R'_j = \bar{N}$  for all  $j=1, 2, \dots, 20$ .

We fill the columns  $B_i$  and  $C_i$  of the register chart, and then write the left-hand sides of the expressions  $R_j$  and  $S_j$  for each  $j=1, 2, \dots, 20$ . Successively examining all the operations with  $i=1, 2, 3, 4$ , from Table 4, we fill their right-hand sides according to formulas (3), (1) and (2) and columns  $B_i$  and  $C_i$  in Table 4. As a result of this examination, from the clear operation ( $i=1$ ), which writes the constant 0, we include according to the column C in all the expressions  $R_j$  the inverted terms  $C_1^j = U_1$  (because for this operation by (2)  $C_1^j = U_1 \& I = U_1$ ), while the direct terms  $B_1^j \equiv 0$  remain unused (therefore the column  $B_i$  contains a dash).

Similarly, from the operation with  $i=2$  we include in all the expressions  $S_j$  the term  $B_2^j = U_2 \& d_j$ , while the term  $C_2^j = U_2 \& \bar{d}_j$  remains unused.

From the operation with  $i=3$  we include in the expressions  $S_j$  the term  $B_3^j = U_3 \& a_{j+1}$  for  $j=1, 2, \dots, 19$  and the term  $B_3^{20} = U_3 \& a_1$ ; in the expressions  $R_j$  we include the terms  $C_3^j = U_3 \& \bar{a}_{j+1}$  for  $j=1, 2, \dots, 19$  and the term  $C_3^{20} = U_3 \& \bar{a}_1$ .

Finally, from the last operation with  $i=4$ , for  $j=5, 6, \dots, 10$  we add to  $R_j$  the terms  $B_4^j = U_4 \& b_j$ , and to  $S_j$  the terms  $C_4^j = U_4 \& \bar{b}_j$ , and for  $j=11, 12, \dots, 20$  we add to  $S_j$  the terms  $B_4^j = U_4 \& c_{j-10}$ , while the inverted term  $C_4^j = U_4 \& \bar{c}_{j-10}$  remains unused.

Expressions (1) and (2) for the terms former and expression (3) for the input former are thus constructed for the register  $R_1$ . If each signal  $U_i$  is generated in one control automaton, the write control former is not used in this example (because the RS flipflops do not have the control signal  $V_j$ ). Otherwise, expressions (4) and (5) also will have to be constructed.

Thus, for the first bit we obtain

$$R'_1 = \bar{N};$$

$$R_1 = C_1^1 \vee C_3^1, \text{ where } C_1^1 = U_1; C_3^1 = U_3 \& \bar{a}_2;$$

$$S_1 = B_2^1 \vee B_3^1, \text{ where } B_2^1 = U_2 \& d_1; B_3^1 = U_3 \& a_2.$$

## FORMULA DESCRIPTION OF REGISTER OPERATIONS

Once we agree on some conventions, we obtain a tool for concise description of register operations, which is consistent with the autogram [2] and is useful in other cases.

We represent operations in the form

$$i. U_i \& F_i^j \rightarrow J_i - a_i \quad (6)$$

Here letters have the same notational meaning as in Table 3. The conjunction  $U_i \& F_i^j$  stands for a set of terms. The set  $F_i^j$  is enclosed in parentheses if it consists of two or more elements (separated by a comma).

In addition to this fairly abstract description of operations, we can use a description that takes into account the specific implementation of the operation in the register. This description has the form

$$i. U_i \& F_i^j \rightarrow WJ_i - a_i \quad (7)$$

where  $W$  is the phase structure of the operation, described as in Table 2. For the two write operations, and also for the two one-phase write operations and the four operations with constants (see Table 2) it is more convenient to use (7) where  $W$  is replaced with the notation of the flipflop inputs realizing the terms for the particular operation  $a_i$ . Depending on whether the direct or the inverted term is required on the input  $W$ , the set  $F_i^j$  should include respectively the names of the direct or the inverted signals; for the operations with constants the set  $F_i^j$  is omitted altogether, because with these operations certain inputs  $W$  realize only terms equal to the condition  $U_i$ . The elements  $i$  and  $a_i$  in (7) are not always used either.

As a demonstration, we give some operation descriptions in the form (7).

Clear a 20-bit register through the setting inputs  $R^0$ :

$$0. \bar{N} \rightarrow R^0 \wedge 20 - 0.$$

Clear this register through the information inputs  $R$ :

$$i. U_i \rightarrow R \wedge 20 - 0.$$

One-phase direct-code write:

$$i. U_i \& bI^{20} \rightarrow SI^{20} \text{ — } f_i.$$

One-phase inverted-code write:

$$i. U_i \& \bar{b} I^{20} \rightarrow SI^{20} \text{ — } \bar{f}_i.$$

Paraphase direct-code write:

$$i. U_i \& bI^{20} \rightarrow (SR)I^{20} \text{ — } f_2.$$

Paraphase inverted-code write:

$$i. U_i \& bI^{20} \rightarrow (RS)I^{20} \text{ — } \bar{f}_2.$$

## GENERAL REMARKS

The formal register-chart-based synthesis procedure described in this article produces only logical equations that describe the register functioning and circuit. To implement the register, the equations should be transformed taking into account the real constraints on the component elements forming the register. These are primarily constraints on the number of inputs in AND and OR elements, constraints on admissible loads on element outputs, and constraints on the functions realized by the elements. To this end, we can use formal transformations of expressions by the rules of Boolean algebra, which have been considered by many authors (see, in particular, [2, 4]).

If these formal transformations produce unacceptably long signal paths, we need to apply informal design techniques, which may involve backtracking to the initial stages of development to re-examine the register and other components of the unit being considered.

When specifying the register chart, we can allow for various optimizing considerations, which in particular impact on speed and hardware costs. In this context, we should mention the previously described features 1-4 of the operations from Table 2. Moreover, all the operations in Table 2 are dual, i.e., they are executed both with direct signals  $f_i^j$  and with inverted signals  $\bar{f}_i^j$ . Taking a particular version of the operation, we can change the loads on the flipflop outputs forming the signals  $f_i^j$  and  $\bar{f}_i^j$ , which may also change the length of the signal paths.

If RS flipflops are used in the register, the paraphase write is executed in one time step, but it doubles the number of realized terms and produces more complicated output expressions compared with the one-phase write. The one-phase write, on the other hand, requires preliminary clearing of the register, which involves an additional time step, an auxiliary register

clear circuit, and a clear control signal. These shortcomings can be avoided by replacing RS flipflops with DV flipflops, but this requires generation of control signals  $V_j$ , thus involving longer circuits and, if different signals  $V_j$  are used for different digits, essentially higher hardware costs. All this reduces the advantages of one-phase write into DV flipflops, so that paraphase write into RS flipflops may prove preferable.

Hardware costs can be minimized by specifying several versions of the register chart and comparing the results of computer-aided register synthesis to select the best option.

To ensure reliable operation of the register produced by this design procedure, we have to consider carefully the time characteristics of all input signals reaching the register, including the clocking signals. This is discussed in detail in [2]. Here we only consider the basic principle.

The simplest way is to build a register using two-step flipflops clocked by two series separated in time, or else a three-flipflop design with one series of clocking signals, i.e., flipflops triggered by the signal front [3]. At the instant when the flipflops are switched (this is determined by the clock-signal phase), all the signals written into the register during the given step should have had enough time to reach a stable value. If the formation of some input signals requires more than one time step, these signals are written into the register on the condition  $U_i$ , which takes the value 1 only after these signals have finally formed.

If the register uses one-stage flipflops, the algorithmic description of the units should ensure that the write operation is not used simultaneously with output signals from register digits that are affected by writing.

In conclusion, let us note some features of the register chart, which are useful for formal (computer-aided) logic design of registers.

First, the register chart is a concise algorithmic description of the register. Horizontal conciseness is achieved by using collapsed or abbreviated description of the contents of the columns  $J_i$  and  $F_i^j$ , while in the interest of vertical conciseness we should reduce the number of rows in the register chart, covering the largest possible number of register bits in each row. A limiting factor in this approach is that one row may contain only those register bits in which the relevant operation is executed with the same phase structure.

Second, the register chart provides a highly visual tool. The most non-obvious aspects of the register chart is that it requires comparing the list of digits represented in the column  $J_i$  with the list of signals reaching these digits and represented in the column  $F_i^j$ . Each row may contain many different descriptions of  $J_i$  and  $F_i^j$ . In practice, we should primarily strive to order

or simplify the contents of the column  $J_i$ . In most operations, the above-mentioned lists corresponds to some contiguous part of the register, and not the entire register. In relatively rare and complex cases, computer-aided design may use auxiliary software tools, which unfold the abbreviated descriptions of  $J_i$  and  $F_j$  into detailed series with explicit correspondence between their elements and, conversely, collapse detailed series into abbreviated descriptions.

Finally, the register chart can be applied to calculate the contents of the register after any operation using known input information. This provides opportunities for efficient register simulation at the algorithmic level before embarking on logic design. The simple representation of information in the register chart reduces the simulation time, while at the same time accelerating the design of discrete units.

Because of its qualities and complete formalization, the proposed procedure may be used in computer-aided design systems for discrete units.

## **REFERENCES**

1. V. M. Glushkov, Yu. V. Kapitonova, and A. A. Letichevskii, *Computer-Aided Computer Design* [in Russian], Naukova Dumka, Kiev (1975).
2. V. M. Glushkov, Yu. V. Kapitonova, and A. A. Mishchenko, *Logic Design of Discrete Devices* [in Russian], Naukova Dumka, Kiev (1987).
3. N. N. Bukreev, B. M. Mansurov, and V. I. Goryachev, *Digital Microelectronic Circuits* [in Russian], Sovetskoe Radio, Moscow (1973).
4. V. M. Glushkov, *Digital Automata Synthesis* [in Russian], Fizmatgiz, Moscow (1962).

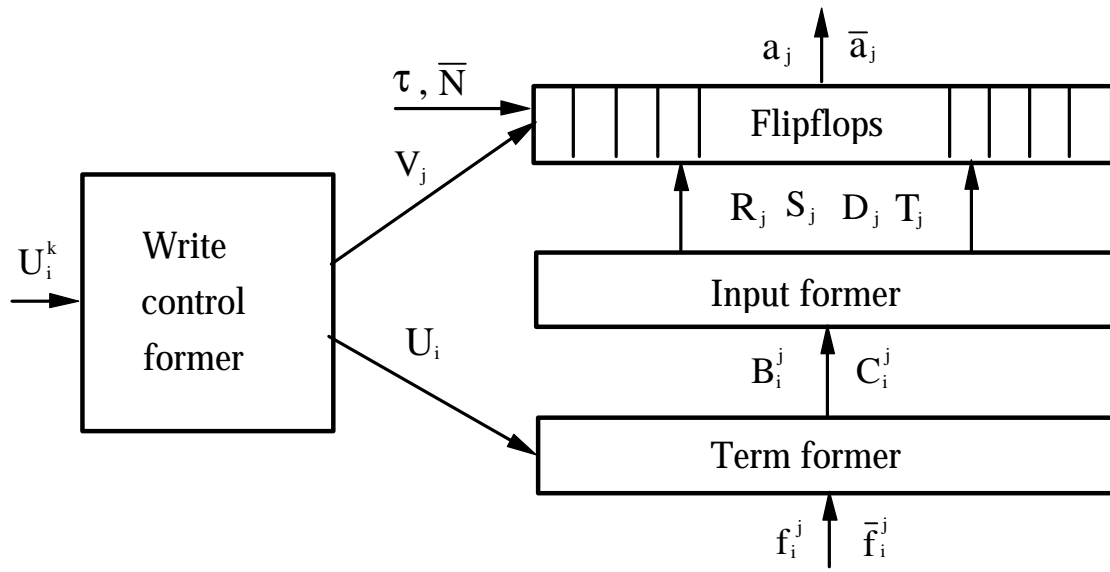


Fig. 1

**TABLE 1.** Transition Tables for Different Types of Flipflops

Inputs		Output for flipflop types			
		RS	R	S	E
R	S	a	a	a	a
0	0	a	a	a	a
1	0	0	0	0	0
0	1	1	1	1	1
1	1	-	0	1	a

a

D	a
0	0
1	1

b

T	a
0	a
1	$\bar{a}$

c

D	V	a
0	0	a
1	0	a
0	1	0
1	1	1

d

T	V	a
0	0	a
1	0	a
0	1	a
1	1	$\bar{a}$

e

J	K	a
0	0	a
0	1	0
1	0	1
1	1	$\bar{a}$

f

R	S	T	a
0	0	0	a
1	0	0	0
0	1	0	1
0	0	1	$\bar{a}$

g

D	V	T	a
0	0	0	a
1	0	0	a
0	1	0	0
1	1	0	1
0	0	1	$\bar{a}$
1	0	1	$\bar{a}$

h

**TABLE 2.** Phase Structure for Operations and Flipflops of Different Types

Operation and comment	a*	Phase structure for different flipflops											
		RS BC	R BC	S BC	E BC	D BC	DV BC	T BC	TV BC	JK BC	RST BC	DVT BC	
Direct code write	f					D -	D -					D -	
Inverted code write	$\bar{f}$					- D	- D					- D	
Paraphase direct code write	$f_2$	SR	SR	SR	SR					JK	SR		
Paraphase inverted code write	$\bar{f}_2$	RS	RS	RS	RS					KJ	RS		
One-phase direct code write (only after register clear)	$f_1$	S -	S -	S -	S -			T -	T -	J -	S -	T -	
One-phase inverted code write (only after register clear)	$\bar{f}_1$	- S	- S	- S	- S			- T	- T	- J	- S	- T	
Bitwise disjunction of $a$ and $f$ (equivalent to $f_1$ after register clear)	$a \vee f$	S -	S -	S -	S -					J -	S -		
Bitwise disjunction of $a$ and $\bar{f}$ (equivalent to $\bar{f}_1$ after register clear)	$a \vee \bar{f}$	- S	- S	- S	- S					- J	- S		
Bitwise conjunction of $a$ and $f$ (equivalent to $f_1$ after writing 1)	$a \& f$	- R	- R	- R	- R					- K	- R		
Bitwise conjunction of $a$ and $\bar{f}$ (equivalent to $\bar{f}_1$ after writing 1)	$a \& \bar{f}$	R -	R -	R -	R -					K -	R -		
Bitwise mod-2 addition of $a$ and $f$ (equivalent to $f_1$ after register clear; equivalent to $\bar{f}_1$ after writing 1)	$a + f$							T -	T -	T -		T -	
Bitwise mod-2 addition of $a$ and $\bar{f}$ (equivalent to $\bar{f}_1$ after register clear; equivalent to $f_1$ after writing 1)	$a + \bar{f}$							- T	- T	- T		- T	
Writing the constant $q = 0$ (when $B \equiv 0, C = U_i$ )	0	- R	- R	- R	- R	- -	- V			- K	- R	- V	
Writing the constant $q = 1$ (when $B = U_i, C \equiv 0$ )	1	S -	S -	S -	S -	D -	D -			J -	S -	D -	
Forced clear (writing zeros when $B \equiv 0, C = U_i$ )	0'	- R'	- R'	- R'	- R'	- R'	- R'	- R'	- R'	- R'	- R'	- R'	
Forced writing of ones (when $B = U_i, C \equiv 0$ )	1'	S'	S'	S'	S'	S'	S'	S'	S'	S'	S'	S'	

\* Designation of operation.

**TABLE 3.** Register Chart for  $R_n$

$R_n$	$j_1^j j_2$	Flipflop type	$V_{ff}$	$T_{sw}$	$Q_{op}$	$Q_{lng}$	$Q_{in}$	$Q_{out}$	$Q_{el}$
$i$	$U_i$	$j_i$	$F_i^j$	$a_i$	$B_i$	$C_i$			

**TABLE 4.** Register Chart for  $R_1$

$R_1$	$1^{20}$	RS			$Q_{op} = 5$		$Q_{in}=64$	$Q_{out}=20$
$i$	$U_i$	$j_i$	$F_i^j$	$a_i$	$B_i$	$C_i$		
0	$\bar{N}$	$1^{20}$	0	0	-	$R'$		
1	$U_1$	$1^{20}$	0	0	-	R		
2	$U_2$	$1^{20}$	$d(1^{20})$	$f_1$	S	-		
3	$U_3$	$1^{20}$	$a(2^{20}, 1)$	$f_2$	S	R		
4	$U_4$	$5^{10}$	$b(5^{10})$	$\bar{f}_2$	R	S		
		$11^{20}$	$c(1^{10})$	$a \vee f$	S	-		