Test Routines Based on Symbolic Logical Statements*

RICHARD D. ELDRED

Datamatic, Newton Highlands, Massachusetts

1. *Introduction*

In order for the successful operation of a test routine to guarantee that a computing system has no faulty components, the test conditions imposed by the routine should be devised at the level of the components themselves, rather than at the level of programmed orders. Therefore, it seems that the proper approach to writing test or maintenance routines is by way of the logical diagrams of the system, and not the list of machine orders. This is the only way in which all conditions of operation of each logical function can be uniquely and completely defined and all logical components within each logical function can be made to perform the task to which they are assigned. Orders can then be programmed to present all conditions, and further orders can be programmed to detect improper performance of the logical functions, thereby producing a minimum program which tests and detects failure in each logical component in the system.

2. *Background and Definitions*

A method of producing such a test routine was developed and applied to the Central Processor of the Datamatic 1000 system. The logical functions in the Datamatic 1000 were implemented by tubes and diodes. The functions are in the form of buffer-gate-buffer and were originally represented by logical statements. Since almost any system is originally represented by logical statements, the method described here could be applied equally well to another system.

3. *Description of Method*

A method for selecting the minimum number of test conditions necessary to determine whether all tubes and diodes within a gating structure are operating properly will now be described. External items, such as auxiliary amplifiers, delay circuits, transmission lines, and so forth, are also tested by the tube-diode tests for the gating structures with which they are associated.

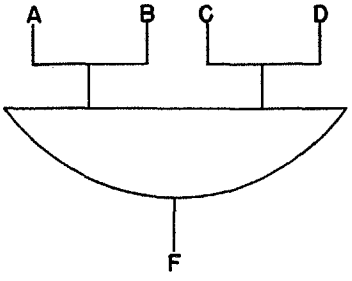

FIG. 1

* Presented at the meeting of the Association, Aug. 11–13, 1958.

Consider a simple buffer-gate-buffer structure as shown in figure 1. In logical notation this is represented by $F = (A \vee B)\cdot(C \vee D)$. It is desired to present a minimum set of conditions to this function to determine whether each component within this function is operating properly. Specifically, it is required to test:

(1) That each input buffer diode conducts when activated. Buffer inputs will hereafter be referred to as *or* inputs.

(2) That each gate leg diode inhibits when not activated. Gate legs will hereafter be referred to as *and* legs.

(3) That the output of the gating structure is not lost when traveling via output components.

(4) That the output components do not provide a signal when the input conditions are inappropriate.

Suppose we initially apply the input combination $A\cdot\overline{B}\cdot C\cdot\overline{D}$ which should activate the function and then check for an output at F. This checks requirement (1) for *or* inputs A and C. It also checks requirement (3). Now shift to a different combination of input variables, changing the active inputs at every *and* leg whenever it is possible to do so, but always keeping one, and only one, *or* input variable active to each *and* leg so that this input buffer diode bears the sole responsibility for activating the gate leg. This gives us the second input combination $\overline{A}\cdot B\cdot\overline{C}\cdot D$ which checks requirement (1) for *or* inputs B and D, and again condition (3).

Now apply the combination $\overline{A}\cdot\overline{B}\cdot(C \vee D)$, which should not activate the function, and check for no output at F. This checks condition (2) for *and* leg $(A \vee B)$ and also condition (4). Now shift to a different combination of input variables activating in turn all *and* legs except one so that each *and* leg has the sole responsibility for inhibiting the output of the function. This gives us the combination $(A \vee B)\cdot\overline{C}\cdot\overline{D}$, which tests the $(C \vee D)$ *and* leg and again condition (4).

In summary then, we have two sets of tests: the activation tests and the inhibition tests. The activation tests for the example given are:

$$\overline{A}\cdot B\cdot\overline{C}\cdot D \text{ and } A\cdot\overline{B}\cdot C\cdot\overline{D}.$$

The inhibition tests are:

$$\overline{A}\cdot\overline{B}\cdot(C \vee D) \text{ and } (A \vee B)\cdot\overline{C}\cdot\overline{D}.$$

To briefly consider a slightly more complex example, take the function represented in figure 2.

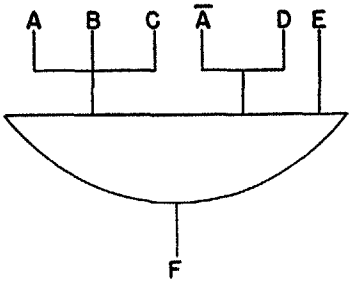

FIG. 2

The activation tests are as follows:

(1) $A \cdot \overline{B} \cdot \overline{C} \cdot D \cdot E$

(2) $\overline{A} \cdot B \cdot \overline{C} \cdot \overline{D} \cdot E$

(3) $\overline{A} \cdot \overline{B} \cdot C \cdot E$ (The state of D is arbitrary.)

For the inhibition tests we have:

(1) $\overline{A} \cdot \overline{B} \cdot \overline{C} \cdot E$ (The state of D is arbitrary.)

(2) $A \cdot \overline{D} \cdot E$ (The state of B and C is arbitrary.)

(3) $(A \vee B \vee C) \cdot (\overline{A} \vee D) \cdot \overline{E}$.

In activation test (1), we hold A active and B and C inactive, thereby testing *or* input A which now has the sole responsibility for activating the gate leg. We already have A active so we hold D active to test *or* input D. E, of course, must be active to activate the function. In test (2), we switch to holding B active and A and C inactive, and since we already have $\overline{A}$ we inactivate D to test *or* input $\overline{A}$. In test (3) we switch to C being active and since the second *and* leg has already been fully tested, the condition of D is arbitrary. In inhibition test (1), we hold the first *and* leg inactive by the combination $\overline{A} \cdot \overline{B} \cdot \overline{C}$. The second leg is now active by $\overline{A}$ so that the condition of D is arbitrary. E holds the third leg active. Gate leg 1 has now been tested for inhibition as it has the sole responsibility for inactivating the function. In test (2), $A \cdot \overline{D}$ holds the second *and* leg inactive. A and E hold the other two legs active so that the condition of B and C is arbitrary. In test (3), $\overline{E}$ holds the third leg inactive, and, as can be seen, any of several combinations will hold the other two legs active.

The illustrated examples were for one gate functions. If a function contains more than one gate, the tests are applied to each gate individually.

It can be seen that the maximum number of activation tests for a function with *or* inputs is equal to the total number of *or* inputs. The minimum number of tests for such a function is equal to the largest number of *or* inputs on one *and* leg. If there are no *or* inputs then the number of activation tests equals one. The number of inhibition tests for any function is always equal to the number of *and* legs. Whether the number of activation tests is at a maximum or minimum, of course, depends on any logical relationship which exists between the input variables.

4. *Application of Method*

This method was applied to develop a test routine for the high speed packages of the central processor of the Datamatic 1000 system. Charts were prepared and filled out by subsystem for each logical function in the system. The columns in the chart were defined to contain the following information:

(1) Symbolic name of logical function.

(2) The logical combinations necessary for all activation and inhibition tests.

(3) The necessary programmed order or orders to produce the logical combination. This included the name of the order and any requirements of addresses, operands, previous orders, and so forth, to produce the necessary logical combination.

(4) How the failure of the logical function under these conditions would affect the operation of the machine, assuming no built-in machine checking.

(5) Whether or not this failure would be detected by built-in machine checking.

(6) How the failure of the logical function under these conditions would affect the expected operation of the program, assuming no machine checking.

(7) Whether or not the effect on the program can be detected by a programmed check.

Every logical function in the system was listed in this manner, with the exception of some second and third level checking functions which assumed an error in their input conditions. Since a program is not capable of knowingly producing incorrect data, it was not possible to list these functions.

In order to write the test program then, it was necessary only to write a minimum set of orders which produced all the logical combinations in column (3). This guaranteed that all required test conditions were presented to all logical functions. To test the checking functions which could not be listed, a programmed check was placed opposite each test order. The programmed checks were defined by the effect of "the failure on the program" as listed in column (6).

5. *Results*

The result of this endeavor was a program containing approximately 110 test orders and an additional 290 machine words comprising constants, control, and programmed checks. This program checked approximately 700 high speed and associated packages containing 900 tubes and 21,000 diodes, for all conditions of activation and inhibition. The total time for one pass through this program is 0.08 seconds. From these specifications, it can be seen that this program is well suited for operation in conjunction with marginal checking.

This program has been used successfully during the debugging and production stages of the Datamatic 1000 and is now being used in the field as the principle maintenance routine for the Datamatic 1000 Central Processor.

It is recommended that the writing of a program such as this be done in conjunction with the logical design checking of the various machine units as it is a natural by-product of such a study.