PHILIPS TECHNICAL REVIEW

VOLUME 35, 1975, No. 10

Structural testing of digital circuits

W. G. J. Kreuwels

Without the aid of a computer it would be almost impossible to develop large logic networks like LSI circuits. Designing the circuit, testing the design, designing and manufacturing the masks, testing during manufacture, all these stages require so much computing work that only computer-aided design can provide the answer. At the same time it is necessary to have good software, otherwise the computing time may become impractically long, even for a fast computer. This article deals with the functional test, whose purpose is to check whether a new design of circuit does in fact perform the required logic function. If the test were to be carried out in full on a network with 64 inputs — not an unusual number for an LSI circuit — it would require 2^{64} experiments. Even with test equipment specially built for the purpose and checking at a rate of once per nanosecond, the test would take more than 300 years. Such a complete test, however, would cover very many more possibilities than those of any practical interest. The programs dealt with in this article, which are based on the structure of a network to be tested, can be used in setting up a much shorter test for checking the functional specifications. Depending on the type of network, this procedure requires only a few minutes to a few hours of computer time, while the actual test takes only a few seconds during manufacture.

Introduction

In the manufacture of logic circuits test procedures have to be carried out during both the design phase and manufacture. The most extensive of these is the 'functional' test, which determines whether a circuit meets the functional specifications, in other words whether the circuit performs the desired logic function correctly.

The functional test for small circuits with few gates (AND, OR, NAND and NOR elements) can be carried out quite simply by successively applying all possible combinations of the logic values '1' and '0' to the inputs. The logic values at the outputs must then correspond to the response of a fault-free circuit. In the design phase this procedure can be simulated on a computer. In a production test the various test patterns can be generated by a counter circuit and fed to the network under test and to a calibration circuit: the two responses are then compared (*fig. 1*).

Ir W. G. J. Kreuwels is with Philips Research Laboratories, Eindhoven.

Such a procedure, even using a very fast computer for generating the test patterns, is not feasible for large circuits that have several hundred gates and scores of inputs and outputs. In a network with n inputs the number of different combinations of input signals is

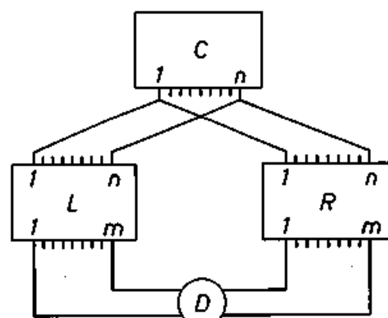


Fig. 1. Testing a simple logic circuit with n inputs and m outputs. A counter C generates the test patterns and simultaneously feeds them to the inputs of the circuit under test L and to a calibration circuit R. The output signals are fed to a comparator circuit D, which delivers a signal when the two responses are not identical.

2^n, and 2^n test patterns therefore have to be applied. Even if such patterns are generated at a rate of one per nanosecond, such a test takes far too long. For a network with 64 inputs it would take no less than 300 years. A further complication is that the response is not entirely determined in all the circuits by the instantaneous combination of input signals. This is only the case in *combinational* networks. Usually, however, large networks are *sequential*, and in such circuits the response is dependent not only on the instantaneous inputs but also on a number of input patterns preceding them and on the order in which they are presented. For circuits of this type the computer would have to generate different series of test patterns, which would make the test last even longer. For large networks, therefore, a very much shorter functional-test procedure is needed.

Even with a shorter test procedure it is only the input and output signals that are examined. Indeed, this is the only way of testing large-scale integrated (LSI) circuits, since the separate components are so small as to be inaccessible: but even with networks on printed-wiring panels it is much simpler to make the measurements only at the input and output connectors. We have therefore tried to find ways of arriving at a group of test patterns that is smaller, yet still sufficient for the functional testing of digital networks.

The existence of such a small group of test patterns may be demonstrated as follows. The starting point of the complete functional test is the truth table; this gives the functional specification of a combinational network, this is to say the relation between all the combinations of input and output signals (a sequential network is described by a series of these truth tables). A large truth table may not only correspond to a single network, it may also correspond to many other kinds of network of widely different structure. Since all these networks have their own characteristic types of fault the complete functional-test procedure is capable of detecting all possible faults in all these various networks. This is much more than is required: all we want to know is whether the particular design or network being tested meets the specifications. It is obvious that a much smaller group of test patterns will suffice for this. These patterns should not be derived from the truth table but from the structural design data. To describe the abbreviated functional test, which only uses this small group of test patterns, we have therefore intruduced the term *structural* test.

This article will give an account of the computer programs that generate the patterns for the structural test. The programs are meant to be used in the design phase of a network, and we shall see that certain design faults can be directly brought to light by such a program even at the stage when the test patterns are being designed. The principle we use for deriving the test patterns was introduced many years ago [1], but has not yet been used in practical programs.

Once the test patterns have been derived, the actual structural test during manufacture takes very little time. Of course the number of test patterns required will depend to a great extent on the type of network, but a circuit with a thousand gates will require a few hundred to a few thousand patterns. If these patterns are generated during the manufacturing test at a speed of one per microsecond, the test will take no more than a few milliseconds. It should be realized when applying the test patterns that the method aims simply at answering the question of whether the network functions correctly or not: the test patterns are therefore primarily meant for fault *detection* and not for fault *location* [2].

Before dealing with the procedure for deriving the test patterns, the scope of the structural test will be indicated by describing the faults that can occur in the various stages of designing and manufacturing a network.

The faults in a digital circuit

The design of a digital network can be subdivided into three phases, each with its own distinct types of fault.

1) In the first phase the time variation of signals in small circuits consisting of a few dozen transistors at most is calculated for various values of the physical parameters that determine the characteristics of the circuit. This circuit analysis gives a general circuit description from which the digital behaviour can be deduced by the introduction of the logic levels and time sampling. This yields descriptions of gate circuits with various specifications for the transistor parameters and for the tolerances ('margins') within which other parameters such as supply voltages, load, temperature, etc. must remain. If one or more of these specifications is not met, the result may be that a signal does not reach the required logic value but a value somewhere in between the two. Such an out-of-tolerance value is referred to as a '½' value (*figs. 2a* and *b*).

Faults of this type are detected during the manufacture of an integrated circuit by measuring the parameters of a special test transistor, which is formed on the same chip as the circuit and is regarded as being representative of the transistors in the circuit. 'Marginal tests' are also carried out to verify whether the circuit is unaffected by variations of the other parameters within the specified limits.

2) In the second phase the gate circuits are combined to form a network. If all the specifications mentioned

under (1) are now taken into account, only the time factor need be investigated. In a large network the delay times of the gate circuits are additive over various paths, and as a result the signals at various places in the circuit can get out of step. A signal may then reach the required level too late and thus be unknown at the sampling time: this case is also referred to as a '½' value (fig. 2c).

These dynamic errors in a design are found by simulating the dynamic behaviour of the network on a computer. The programs required for this are very much simpler than those for circuit analysis but they have to be applied to a much larger network. The amount of computation can be substantially reduced, however, if the designer indicates where difficulties in the design are likely to occur. The dynamic investigation of a design produces a number of test patterns, which can also be used for testing the dynamic behaviour of the circuits during production.

3) Even if all faults of the first type are eliminated, so that all signals reach a logic level at the instants of sampling, design or production faults can still cause a wrong response in the network. These faults are detected by the functional test or by its abbreviated form, the structural test. A fault analysis shows that at this stage we need only consider situations in which the signal has a *fixed* value ('0' or '1') at one particular point in the circuit and does not respond to the other signals in the desired manner. These are referred to as 'stuck at' faults: either 'stuck at zero', abbreviated to s-a-0, or 'stuck at one', abbreviated to s-a-1. Production faults that can cause s.a. faults are broken connections and short circuits. Possible design faults are of course difficult to indicate, but by analogy with open- and short-circuits it is possible to think of a necessary connection that might be missing or an unnecessary one whose inclusion could introduce a fault.

The fault analysis is performed in two phases. First the possible physical faults in the circuit are converted into s.a. faults (other types of fault have already been detected in the other tests). Some physical faults correspond to one s.a. fault, others such as short-circuits may result in a group of s.a. faults at various points in the circuit. In the second part of the analysis it is found that all these physical faults can be detected by a test program for single s.a. faults. It has been found that this program can also detect all groups of s.a. faults.

It will be clear from what has been said above that the design of a large digital network is only possible with the aid of a computer: it is an example of computer-aided design. The designer generally has a large number of existing programs available (sometimes called 'tools') for the computing work. Examples at Philips are the PHILPAC programs for circuit analysis, PHILSIM for simulating the dynamic behaviour of networks, and the TESTCC and TESTSC programs, described in this article, for deriving the test patterns for the structural test.

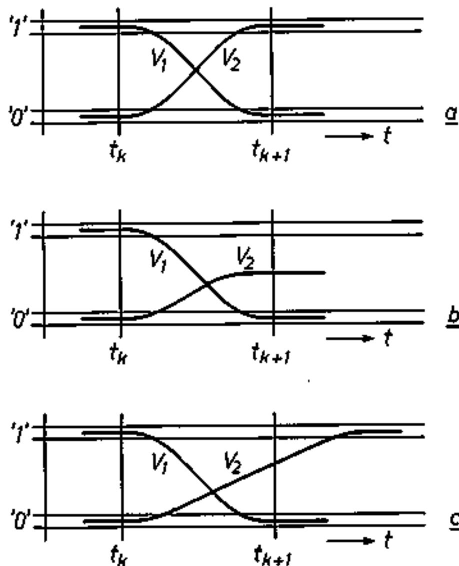


Fig. 2. Illustrating the signal faults in a digital circuit. *a*) Behaviour of two signals V_1 and V_2. Two regions are defined in which the logic values '0' and '1' are assigned to the signals. Only the values at the times t_k; t_{k+1}, etc. are important; in a *synchronous* circuit these times are determined by an external clock signal, in an *asynchronous* circuit by a signal from the circuit itself. *b*) Signal V_2 does not reach level '1', so that at the time t_{k+1} the value of V_2 is indeterminate. It is then said to have a value '½'. *c*) The signal V_2 reaches level '1' but it does so too late. When this dynamic fault occurs, the signal is also said to have a '½' value.

Test patterns for a combinational network

The procedure used in deriving the test patterns for the simple case of a combinational network will first be described. The computer program developed for this has been given the name TESTCC.

Structure input

The computer that works out the test patterns is supplied with the structural data for the network in the form of a table; *fig. 3* shows how this is arrived at for a combinational network. The network (*a*) is presented in the form of a 'directed graph' (*b*), in which all the connections are shown but the components are reduced to nodal points. Inputs and nodes are numbered in such a way that the information in the circuit is always

[1] R. D. Eldred, Test routines based on symbolic logical statements, J. Ass. Computing Mach. 6, 33-36, 1959.
D. B. Armstrong, On finding a nearly minimal set of fault detection tests for combinational logic nets, IEEE Trans. EC-15, 66-73, 1966.

[2] A preliminary investigation with a simple network has revealed that the structural-test program gave almost the same amount of information about the location of the faults in this network as the complete functional test. We therefore intend to design fault-location programs on the basis of the structural test.

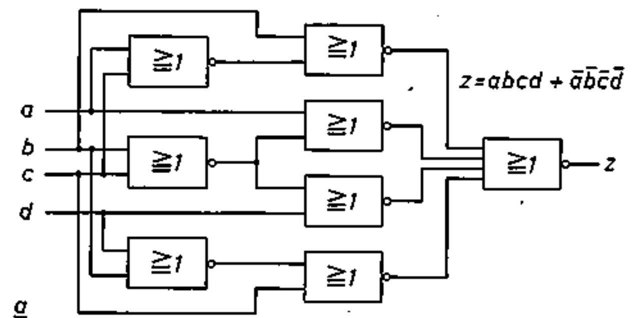
passed on from a lower to a higher number. Each line in the table (*c*) corresponds to an input or a node point: the first column gives the number of this point, the second column gives a code for the type of component represented by the node (1, 2, 3, 4 for OR, AND, NOR and NAND gates). The third column contains a '1' if the node is 'observable' in other words if it represents an output gate of the network (in fig. 3 only the last NOR gate), otherwise the column contains a '0'. The other columns give the numbers of the other nodes that are connected with the inputs of the component. In this way the whole network is described in a form that can easily be fed to the computer, e.g. on punched cards. The truth tables for the various types of component still have to be added to this table, of course.

In addition to the structural data for the network, the computer also receives information about the faults to be detected by the test patterns. For the structural test these are only the 'stuck at' faults, which can occur on any line connecting two nodes. If the network branches at a particular node, the fault need not always affect all the lines branching out from that point; it is possible that the fault will only appear on one of the lines beyond the node, and hence only at the input of one of the next nodal points. It is then called a 'gate-input fault'.

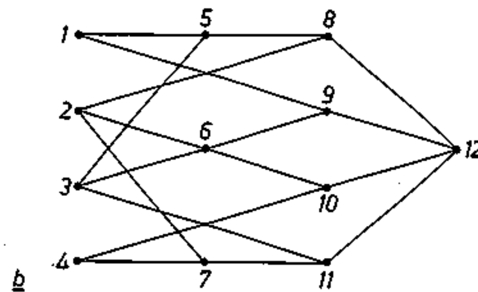
The critical path

The procedure for deriving the test patterns is based on finding *critical paths*. A critical path is a path through the network that makes the effect of an s.a. fault on a particular line in the network visible at an output. To each critical path there corresponds a pattern of input signals, which is therefore the test pattern for the particular fault. The term 'critical path' is used to indicate that all the signals on that path are 'critical': any change in the value of one of the signals propagates to the output. It therefore follows that the corresponding test pattern shows up not only the original s.a. fault but also all the other faults that can occur along that path and hence bring out the same change of value at the output. It does not matter which output gives the error signal because the test checks the total response of the network.

For some faults several critical paths will be possible, and therefore several test patterns. However, since we wish to derive the smallest possible group of test patterns, the search process is stopped as soon as a critical path is found. Of course, a check is made for each test pattern discovered, to see if any other faults can be detected with it at the same time. These need not necessarily be faults occurring along the critical path: in networks with more than one output faults can appear at the same time at other outputs.



a



b

1		0				
2		0				
3		0				
4		0				
5	3	0	1	3		
6	3	0	2	3		
7	3	0	2	4		
8	3	0	2	5		
9	3	0	1	6		
10	3	0	4	6		
11	3	0	3	7		
12	3	1	8	9	10	11

c

Fig. 3. Coding of the structure of the network for feeding the structural information to a computer. *a*) Example of a combinational network. *b*) Simplified diagram of the network (directed graph) in which the inputs and components are represented as nodes. *c*) Table containing all the data relating to the structure of the network. The first column contains the numbers assigned to the nodes, the second column gives a code number for the component represented by the node (a NOR gate in all cases here), the third column contains a 1 where an output gate is referred to, otherwise a 0, and the other columns give the numbers of the other nodes that are connected to the inputs of the component.

The signal under test will be indicated by the symbol k or $\bar{k}$ so that we can follow it through the circuit. If we are looking for a test pattern that detects an s-a-0 fault we call the signal k, if we want to detect an s-a-1 fault we call the signal $\bar{k}$. We can now use the rule that the fault is *not* present if k is 1 but *is* present if k is '0'. *Fig. 4* shows part of a critical path in a network. It can be seen from this figure that the other input signals of the components forming the critical path have to meet a number of conditions, referred to as 'imperative implications'. Of course, these conditions may lead to contradictions in other parts of the network, in which case the path cannot be used and another must be looked for. It will be evident that isolating a path in which no contradictions occur, and doing this for all

s.a. faults that can occur in a network, is a very considerable task. We cannot therefore discuss the complete TESTCC program here but will have to confine ourselves to the algorithm used by the computer.

Non-testable faults

When a critical path for a particular fault has been found, this shows in itself that the fault is capable of being tested, and at the same time the test pattern to be used has been found. The question that now arises, of course, is whether a critical path can in fact be found for every s.a. fault. We know that a critical path exists for all faults that are detected by the *complete* functional test, since this test also makes the faults observable at an output. Thus when all the possible critical paths have been explored systematically in the derivation of test patterns for the structural test, test patterns have to be found for all these faults. There are also faults, however, that are *not* discovered by the complete test. There is no critical path for a fault of this kind and therefore no test pattern can be found for it. This relates mainly to faults in parts of a network that are redundant in the functional logic. In some cases such parts of the circuit are added for technical reasons, but the discovery of a non-testable fault in the design phase of a network more usually points to a design fault. The ability to reveal such faults at an early stage is therefore a valuable feature of the structural procedure.

Generation of test patterns

Let us suppose we wish to find a test pattern that will show whether there is an s-a-0 fault on a connection line j in a network. The algorithm for looking for test patterns consists of a number of procedures: the first one, called FOR (for forward) is started by assigning the value '2' to all lines in the network. Expressed in words, this means that the values of the signals are unknown. For the component that has the line j as an output procedure FOR makes use of the truth tables to look for a combination of input signals for which this line has the value '1'. If there are various ways of doing

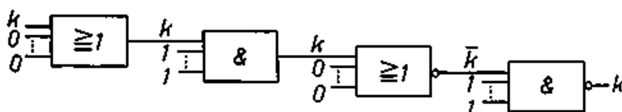


Fig. 4. Example of a chain of components forming part of a *critical path*. At one of the inputs of the component on the far left there is a signal k; the level '0' or '1' of k indicates whether or not a 'stuck at' fault is present somewhere on the left of this component in the network from which this chain has been taken. (A 'stuck at one' or 'stuck at zero' fault means that the signal at a particular point remains fixed at the value '1' or '0', irrespective of the input signals applied to the network.) For the signal k to be transferred to the right, the other inputs of the OR and the NOR gate must all be '0'and those of the AND and NAND gate must all be '1'.

this, FOR remembers them. In accordance with the first possibility investigated, the values '2' at the inputs of the component are now replaced by a '0' or a '1'. The signal on the line j acquires the value k. FOR then starts to look for a critical path along which k appears at the output. The various possible paths are investigated one by one, and in each case the values at the inputs of the other components are changed from '2' to '1' or '0', in the manner shown in fig. 4. If contradictions are found, FOR tries the next possible path. If the filling in of these 'imperative implications' leads to contradictions in all paths, FOR carries out the same investigation for the other combinations of input signals of the component that has j as its output.

In the search for a critical path there are two alternatives: either FOR encounters contradictions in all possible paths, or FOR finds a critical path along which an output can acquire the value k (of course a path can also be used if the output acquires the value $\bar{k}$). In the first case there is no test for the fault investigated; FOR then receives the instruction to look for a critical path for another fault. If FOR does find a critical path it remains available, as there may be more possibilities for the line j and there is not as yet any certainty about the usefulness of the path that has been found. Contradictions may yet arise during the introduction of the 'non-imperative' implications. Before there is any question of a test pattern it is, of course, also necessary to substitute '0' or '1' for the value '2' on all lines in the network that are *not* directly connected with the critical path, and these values must also not be contradictory, either with one another or with the values already assigned. These other signals are filled in by a procedure called BACK, which begins at the output side of the network and works through it in the opposite direction to FOR. In general BACK will also encounter various possible choices, which must be dealt with in succession.

When the BACK procedure encounters no contradictions, a test pattern has been found. If inputs of the network remain with the value '2', this means that the signals here may be chosen arbitrarily. If BACK continuously comes up against contradictions the path found by FOR does not after all lead to a usable test. FOR must then look for the next possibility, which in turn is investigated by BACK. In this way the interplay between FOR and BACK leads either to a test pattern or to the conclusion that there is no test for this particular fault. The combination of FOR and BACK is called LOOK.

When a test pattern has been found for an s.a. fault on a particular line in the network, this same test pattern will also reveal faults on a number of other lines, as we saw earlier. To prevent the whole LOOK procedure from being repeated for these faults as well, the

algorithm contains a procedure MORE. This procedure examines each test pattern to find out which other faults it will also discover. A procedure BOOK keeps a tally of the faults for which a test pattern has already been found in this way, and on the basis of this information a procedure CAND determines the next 'candidate', in other words the particular fault for which a test pattern must now be sought. The interplay of BOOK and CAND in fact 'directs' the generation of the test patterns and is therefore called DIRECT. The DIRECT, LOOK and MORE procedures together constitute a 'fault-detection strategy' STRAT, which can solve the whole problem.

Fig. 5 illustrates schematically the interaction of the various procedures that together constitute STRAT. Of course, depending on the actual execution of the procedures FOR, BACK, etc. there are various possible STRAT procedures. Fig. 5 shows only one example, the procedure followed by the TESTCC program, which derives the test patterns for a combinational network.

Test patterns for a sequential network

As already mentioned above, the response of a sequential network to a particular input pattern depends not only on this one pattern but also on a number of preceding patterns and on the sequence in which they are presented. This means that *series* of test patterns are necessary for the structural testing of a sequential network. The next section will give an account of the TESTSC program used for deriving these 'test sequences'. This program is a collection of a number of strategies that work in much the same way as STRAT. First, however, we shall look at the way in which the structure of such a circuit can be fed to the computer.

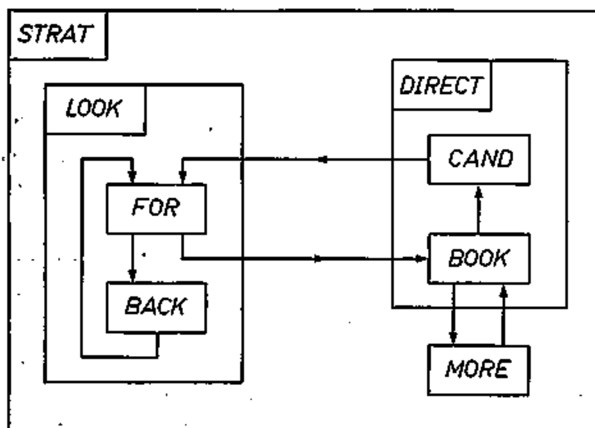


Fig. 5. Diagram of the procedures that together form the strategy STRAT for generating the test patterns for a combinational network. The procedures FOR (forward) and BACK, together forming LOOK, generate the test patterns; MORE investigates the total fault-detection capacity of the test patterns found; BOOK and CAND, together forming DIRECT, determine which new faults will require a search for a test pattern.

Structure of a sequential network

The characteristic feature of the structure of a sequential network is the occurrence of feedback loops in the information flow through the network. In describing this structure by the method illustrated in fig. 3 it is not therefore possible to number the nodal points in the directed graph in such a way that the information flows only from a low to a high number. At certain places information will inevitably flow back from a high to a low number. Since the FOR and BACK procedures that look for a critical path are unable to handle such feedback loops, we use a special approach in which a sequential network is represented as a series of combinational networks.

In a sequential network all the internal feedback loops can be considered as being broken and taken to the outside (see *fig. 6a*). This produces a number of new inputs and outputs, denoted by Y, which differ from the existing inputs and outputs, denoted respectively by X and Z, because they are not accessible in the actual network. The signals on these lines cannot therefore be directly affected or measured. The feedback lines in this model are produced by connecting the Y outputs located outside the circuit to the corresponding Y inputs. The resultant network — the inside block in fig. 6a — contains no further feedback lines and is therefore combinational. It can be described by a table of the type shown in fig. 3c: it is, however, necessary to indicate beside each input or output whether it relates to an X, Y or Z line.

Our next step is to replace the feedback lines in the model shown in fig. 6a by 'feed-forward' lines to copies of the same network. This yields the model in fig. 6b, which shows the sequential circuit as a series of identical combinational circuits, interconnected by the Y lines. This model gives a good picture of the actual situation in a sequential network: at the time t_0 to t_n the input patterns X_0 to X_n are successively presented to the network, giving the responses Z_0 to Z_n. The output pattern Z_n depends on the input signal X_n and on the values on the Y lines at the time t_n. If we indicate this functional relation by f_Z, then $Z_n = f_Z(X_n, Y_n)$. Similarly, Y_n is a function of the preceding input pattern and of the Y values at that moment of time: $Y_n = f_Y(X_{n-1}, Y_{n-1})$. Repeated substitution of this expression in the expression for Z_n yields the relation illustrated in fig. 6b (the figure must then be read from right to left). The signals X_0 to X_n actually presented in the network at the times t_0 to t_n are fed in this model to the copies 0 to n at the same times.

Generation of the test sequences

The principle of the derivation of a test sequence is illustrated in fig. 6. A search is made with the LOOK procedure for a critical path for a particular 'stuck at' fault in the combinational network from the model in fig. 6*a*. We thus look for a test pattern that makes the fault visible at an output (we shall return below to the initial values of Y_i assigned to the Y inputs for this purpose). If the fault signal appears via this path at a Z output, this test pattern alone is all that is necessary for detecting the fault. If, however, k appears at a Y dashed line in fig. 6*b*): the test patterns X_i to X_j calculated in these copies then form a test sequence for this particular s.a. fault.

The initial values Y_i that we fed in when calculating the path in copy i are the result of 'initialization'. This is done by means of a series of patterns X_0 to X_{i-1}, which starts from an arbitrary state of the network in which the values on the Y lines are completely unknown, and applies a known pattern Y_i to the Y lines. All the subsequent Y and Z responses on input patterns X can be calculated from these initial values. The

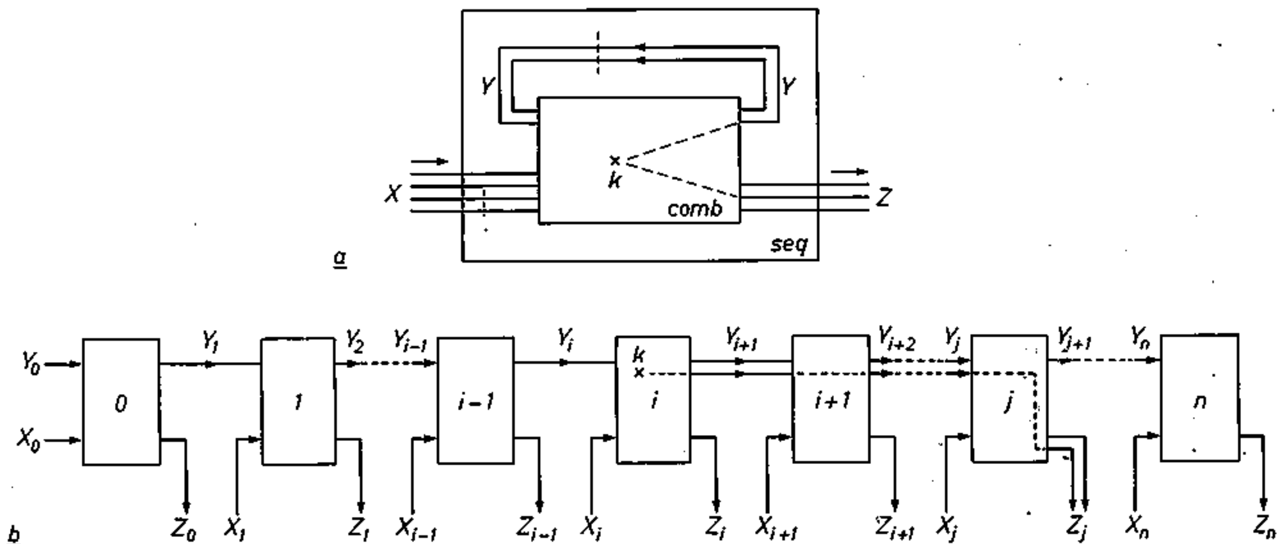


Fig. 6. *a*) Model of a sequential network in which it is represented as a combinational network with external feedback lines Y. The response Z in this model depends not only on the instantaneous input pattern X but also on the values on the Y lines. These in their turn are determined by what has been presented earlier on the X lines. For an s.a. fault k a search for a critical path can be made in the combinational model. If k appears at a Z output, one test pattern is sufficient for detection; if k appears at a Y output, a series of patterns must be derived. *b*) Extension of (*a*) in which the feedback lines are replaced by connecting lines to copies of the same network. (The various lines that together transport an X, Y or Z pattern are all represented here as a single line.) A critical path in copy i that ends at a Y output is continued in copy $i+1$, starting from the calculated values Y_{i+1}. By continuing until the fault appears at a Z output we find a test sequence X_i to X_j. The critical path is schematically represented as a dashed line.

output, the fault can then no longer be observed, since the Y lines are not observable. We again look for a critical path in the model given in fig. 6*a*, but now starting from the values Y_{i+1} on the Y lines obtained with the first test pattern. In fig. 6*b* this procedure means that the first pattern is looked for in copy i and the second pattern in copy $i+1$. In this copy as well k may again appear at a Z or at a Y output; if it appears at a Y output we repeat the procedure in the next succeeding copy. If different test patterns are found in a copy with k appearing at a Y output, we must then make a further examination of all these different Y patterns in the next copy. As soon as the fault appears at a Z output in one of the copies, a critical path has been found and the search is stopped. If we call this copy j, then the critical path runs through copies i to j (the initialization sequence X_0 to X_{i-1} should be specified by the network designer and must precede every test sequence. The complete test for an s.a. fault thus consists of the series of patterns $X_0, \ldots X_{i-1}, X_i, \ldots X_j$.

When calculating a critical path we must bear in mind that the fault to be detected is of course present *in each copy*, i.e. including the copies 0 to $i-1$ that are traversed during the initialization. Apart from the result of the initialization for the fault-free circuit, we must therefore also calculate the pattern Y_i for every possible s.a. fault. This means that in deriving a critical path for a particular fault we must always proceed from the initialization result Y_i computed for this fault.

During the calculation of the critical path the transition to a subsequent copy is always commanded by the clock signal that generates the times t_0 to t_n. The model

in fig. 6*b* assigns this signal to the input pattern X; the X lines corresponding to it are called clock lines and the others are called functional X lines. If we vary only the values on the functional lines and keep the clock signal constant, the response Z_i changes but the subscript i does not. This is the situation in which we look for a critical path in copy i. Changing the clock signal causes i to go to $i+1$, so that the search can be continued in the following copy.

The TESTSC program

The TESTSC program begins with a START procedure for feeding the data to the computer (*fig. 7*). Information on the structure of the network is read in by the procedure READ and processed by the procedure BOOK. The latter procedure derives from the structure the list of candidates, i.e. the list of possible 'stuck at' faults for which test sequences have to be sought. The initial information X_0 to X_{i-1} is presented by the INIT procedure. After this has been entered, a MORE procedure calculates for each of the fault candidates the initial Y values Y_i from which a critical path is to be sought. This also reveals the faults that have already been detected by the initialization sequence: it is of course possible that certain faults will appear at the Z output when this sequence is applied. The results of MORE are similarly stored by BOOK.

A number of faults that the designer knows to be undetectable are 'masked' in a sub-procedure of START called MASK; no sequence will be sought for these faults. These might typically be faults in parts of the network that are technically necessary but logically redundant.

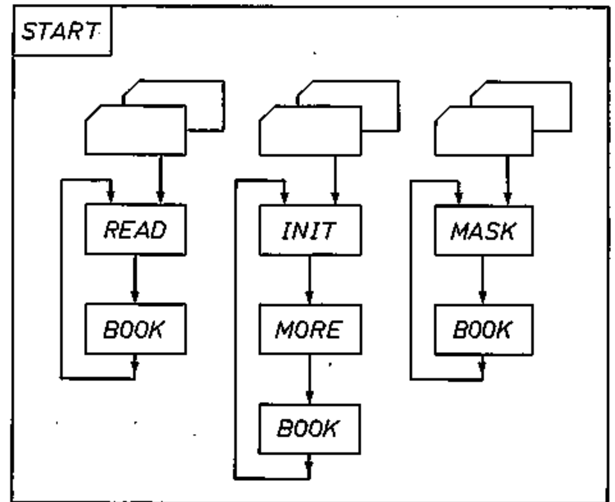


Fig. 7. START procedure for entering the data into the computer. The information sources are symbolically represented by punched-card packages. The structural information of the network is read in by the READ procedure and stored in BOOK. The initialization sequence is read in by INIT; the procedure MORE then calculates the starting pattern Y_i (see fig. 6) for the various possible faults and at the same time investigates to find out which faults have already been detected during the initialization. The MASK procedure receives data relating to faults which the designer knows to be undetectable. All this information is collected in BOOK.

The strategy STRAT

The principal fault-detection strategy STRAT (*fig. 8*) generates the test sequences in the manner described above. Since we always look for the critical path in a combinational network, this strategy is virtually identical with that in fig. 5. The only difference is a procedure CLOCK, interpolated between LOOK and DIRECT. As long as this procedure is not active, a search is made for a critical path in one particular copy in the manner described in fig. 5. The CLOCK procedure always ensures the transition to the next copy. For each fault selected by the CAND procedure the search is started in copy i, always proceeding from the Y_i values relating to the fault. Under the control of CLOCK a series of copies i to j is then run through for each fault and a test pattern is calculated in each copy. The sequences for the various faults will generally be of different length. The MORE procedure investigates to find the other faults detected by each sequence. BOOK keeps a continuous tally of particular faults found by the different sequences.

The fact that STRAT fails to find a test sequence for a particular fault does not permit the conclusion that tests cannot be made for the fault. The strategy STRAT does not investigate *all* detection possibilities but confines itself to the most promising patterns — those that contain the fault signal k. If there are too many of these in a particular copy, even these may not all be used. To restrict the computer time required a limit is

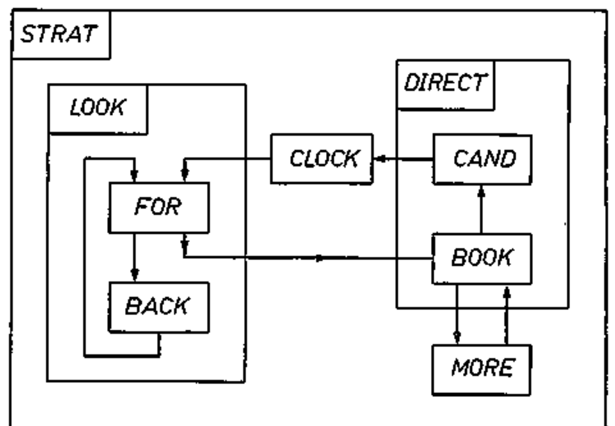


Fig. 8. A fault-detection strategy STRAT for sequential networks. For the various faults indicated by the procedure START test sequences are derived by generating test patterns in a series of copies i to j (see fig. 6*b*), for each fault starting from the input pattern Y_i for the given fault. The test pattern in a particular copy is generated in the same way as illustrated in fig. 5; the CLOCK procedure is responsible for the transition to the next copy. Here again a MORE procedure investigates to find out the total fault-detection capacity for the test sequence found.

also set to the length of a test sequence, so that the search procedure is stopped after a certain number of copies. In these latter points the program is very flexible: the user may indicate how far he wishes the search to be continued.

The strategies STRATZ and STRATYZ

In a well designed network the strategy STRAT will generally be sufficient for computing the whole test program. If faults remain for which STRAT finds no sequence, it is very important for the designer to know whether these faults are in fact non-testable, or whether a test may yet be found for other Y values. To investigate this we added two additional strategies to the program TESTSC. Our reasoning was as below.

If a fault is testable a 'last' copy j exists in which the fault appears at a Z output. There is then also a pattern Y_j which, together with a pattern X_j, gives rise to a critical path to a Z output. It must be possible to find this path by means of the LOOK procedure provided we allow *all* the possible values at the Y inputs. Having found one or more Y_j patterns in this way, we proceed to derive the test sequence. We do this by working through the model in fig. 6*b* from right to left until we find a Y pattern that corresponds to the initialization result Y_i. *Fig. 9* illustrates schematically the strategy STRATZ that carries out this search; since *all* the Y patterns now have to be investigated in each copy, this strategy requires a great deal more computer time than the STRAT strategy, in which only specific Y values were involved.

If the STRATZ strategy also fails to find a test for particular faults there still remains one possibility to be investigated. Depending on the places where the loops in the model shown in fig. 6*a* happen to have been broken, faults will exist that cannot appear directly at a Z output. These faults — *if* they are detectable — first have to emerge via a Y output of the copy before last. This means that such faults are only detectable if the value k appears at one of the Y inputs in the last copy. The FOR procedure only propagates k values to the right, and BACK cannot introduce any new k values at all. If we want to have k values at Y inputs in the last copy, we must therefore obtain them with a LOOK procedure in the penultimate copy. For each of the faults remaining after STRATZ we must thus investigate whether a Y pattern exists that will cause the fault to appear at a Y output and in the next succeeding copy at a Z output. This is investigated by the strategy STRATYZ (*fig. 10*). Here again *all* the Y values in a copy must be investigated, and therefore this strategy also requires a great deal of computer time. STRATYZ is consequently used only if faults without a test sequence remain after STRATZ. Only when neither

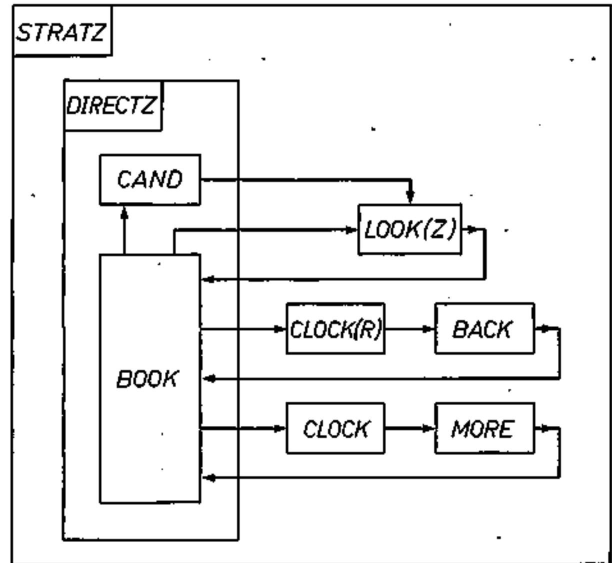


Fig. 9. Strategy STRATZ for investigating the testability of faults for which STRAT has not found any test sequence. For each of these faults, indicated by CAND, the procedure LOOK(Z) searches for patterns Y_j with which the fault appears at a Z output. The existence of one or more such patterns indicates that the fault is testable. To investigate which of the patterns may emerge from the initialization result Y_i via a particular test sequence, the CLOCK(R) procedure starts the BACK procedure in the preceding copy and calculates in this the corresponding input patterns Y_{j-1}. In this way the model in fig. 6*b* is computed from right to left (CLOCK(R) is a CLOCK procedure that works in reverse) until a Y pattern is found that is identical with Y_i, thus yielding the X patterns that form the test sequence. Finally the CLOCK and MORE procedures investigate whether this sequence is capable of detecting other faults as well.

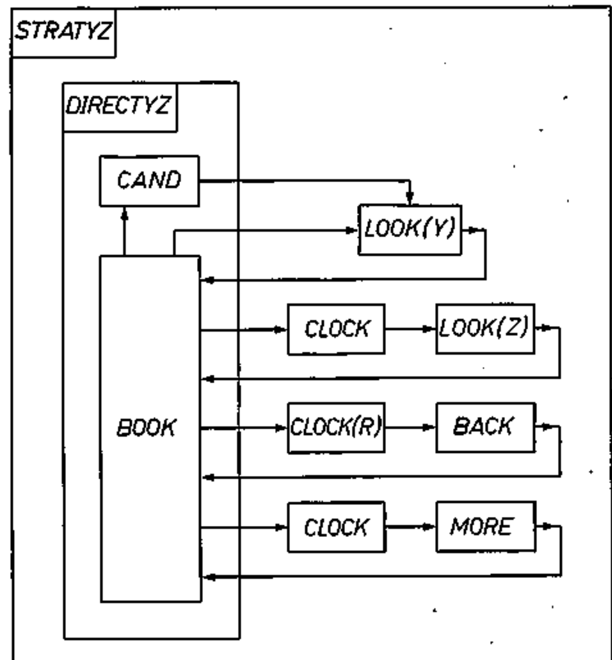


Fig. 10. The strategy STRATYZ used for investigating whether a fault in the penultimate copy appears at a Y output and in the last copy at a Z output. The procedure LOOK(Y) first calculates all the Y patterns with which the fault appears at a Y output. Then the CLOCK and LOOK(Z) procedures investigate to find the cases for which the fault appears in the next copy at a Z output. If such a pair of patterns Y_{j-1}, Y_j is found, the fault is testable. The test sequence is then determined in the same way as in STRAT(Z) by calculating back to the initialization result.

STRATZ nor STRATYZ can find a test may we conclude that a fault is fundamentally incapable of being tested.

In addition to the strategies described above the program TESTSC contains various other facilities, all of which are designed to limit computer time. The designer may for example supply a number of input sequences that he thinks likely to be able to detect faults. The ability of these sequences to detect faults is then determined by means of the procedure which does this in any case for the initialization sequence. It is also possible to generate sequences at random: in this way a large number of test sequences may sometimes be made available with relatively little computer time. These latter procedures are run through in the program TESTSC before the strategy STRAT.

Applications

The TESTCC and TESTSC programs can be used for a wide range of networks, including LSI circuits and networks formed on printed-wiring boards. More complex systems, such as computer subsystems, can also be handled as single entities. For practical reasons a limit is set to the number of lines in a network on which faults can be detected. Networks with a maximum of about 1000 gate circuits can be handled.

It is not easy to give figures for the computer time required, because it depends to a great extent on the type of network to be tested. Shift registers and counters in particular tend to increase the time considerably. However, if good use is made in such cases of the possibility of investigating in advance the usefulness of test sequences supplied, this effect can be eliminated. Taking as an example a sequential network with 1000 gate circuits, the TESTSC program on a computer such as the IBM 370/165 will generally need a few hours of computer time (and a storage capacity of 250 k bytes) to generate the whole structural test program, provided no untestable faults are found. The generation of one test pattern takes about a few hundredths of a second on the computer. If untestable faults are found, which tends to happen in the design phase, it may cost a few hours of computer time before they can be established with certainty. If it is possible however, to divide up the design of a sequential network in such a way that the investigation can be performed on a few combinational networks, a computer time of a few minutes to about half an hour will be sufficient.

What takes most time and is therefore most expensive is determining whether faults are fundamentally incapable of being tested. Nevertheless this is in fact one of the most important features of these programs, since they enable design faults to be detected that would otherwise not appear until a later stage, possibly not until manufacture or later. This can lead to considerable savings, expecially in the development of LSI circuits.

Summary. The functional test is made to determine whether the design of a logic circuit properly reproduces the desired logic function. In the complete test all possible combinations of input signals are fed to the network and the responses to these test patterns are compared with the responses of the fault-free circuit, which is known from the truth table. For a large circuit, such as a digital LSI network, this is not feasible; even if the test patterns were generated by a computer the test would take far too long (up to a few hundred years). It is possible, however, to make do with far fewer test patterns. Starting from the structure of a network a small group of test patterns can be derived that enable the test to be carried out in a short time with 100% certainty. This abbreviated test is referred to as 'structural'. This article describes two programs for deriving these patterns: the program TESTCC is used for combinational networks, in which the response depends only on the instantaneous input signal and in which a fault is detected by one test pattern. The TESTSC program is used for deriving test sequences for sequential networks, in which the response is determined by series of input patterns.

The article begins with a brief review of the faults that can occur during the various phases in the design of a digital network. In the structural test it is only necessary to take account of 'stuck at' faults on the various connecting lines in the network; the value on such a line remains fixed at '0' or '1' and does not respond to the other signals. A strategy is described that searches for a critical path for each s.a. fault in a combinational network. This is a path through the network that enables the fault to become visible at an output; the input pattern that brings this about is then the test pattern for this fault. To derive test sequences for a sequential network a model is used in which the network is represented as a series of identical combinational networks. These are derived from the sequential network by taking the internal feedback lines outside the circuit and breaking them. In this model successive input patterns are fed to a series of successive copies of the network. For each s.a. fault a search for a critical path is then made for a number of these copies. If the search is successful, the appropriate input patterns form the test sequence. If no critical path is found for a particular fault, the conclusion can be drawn that the fault is not capable of being tested. This usually indicates a design fault; the discovery of such faults, apart from finding the test patterns and sequences, is the principal purpose of the programs. Networks with a maximum of 1000 gates can be processed. The computer time required depends largely on the type of network; with a conventional large computer it may vary from a few minutes for a simple combinational network to a few hours for a complicated sequential network.