SIDE-EFFECT FAULT MODEL FOR TESTING VLSI CIRCUITS

P. GÄRTNER

Department of Electronic Devices,
Technical University, H-1521 Budapest

Received June 5, 1986
Presented by Prof. Dr. K. Tarnay

Abstract

The paper presents a new fault model for testing sequential digital circuits, preferably processor-like systems. The concept is based upon the functional specification consisting of instructions. Correct operation of the system is tested by checking not only correct execution of the instructions but their side-effects, as well. Testing for side-effects can be carried out by executing instruction-pairs.

Tests for highly complex integrated circuits are often generated on the basis of their functional description [1], [2], [3]. It is carried out with the assumption of various fault models, or even without them. The paper presents a new fault model which provides general strategy for generating functional (behavioural) tests.

The problem is approached from the event-oriented structure of the functional specification. The set of input-events is defined as elementary events consisting of only one single change at one of the inputs and/or as complex events comprising a given sequence of certain input vectors.

The specification maps the elements of this set into events defined upon the outputs and internal storage elements, only the final modified state has to be — and usually is only — specified while the others remain unchanged. The main question of the behavioural test is:

"Does the network perform as it is prescribed?" — i.e. whether the input event results in the specified changes. An answer to this question is necessary but not sufficient for the proper test. To be sufficient, also another question has to be answered: "Does the result comprise anything not prescribed?" — that is whether there is any *side-effect* present. (Considerations will be restricted to side-effects upon the internal storage elements because primary outputs are directly observable and so detection of their faulty behaviour is trivial.)

In accordance with the Mealy model the output Y and the internal state Z of an automaton at time t is:

$$Y^t = \lambda(X^t, Z^{t-1})$$

$$Z^t = \delta(X^t, Z^{t-1})$$

If there are p internal memory elements then Z can be detailed as a binary vector of length p:

$$Z = z_1, z_2, z_3 \ldots z_p$$

The change of the internal memory elements with respect to the previous state is:

$$\Delta^t = Z^t \oplus Z^{t-1}$$

The next-state function δ of a faulty automaton is changed to δ^* and then the state of the memory elements is:

$$Z^{t*} = \delta^*(X^t, Z^{t-1})$$

and their change can be written as

$$\Delta^{t*} = Z^{t*} \oplus Z^{t-1}$$

The presence of fault automatically renders $\Delta^{t*} \neq \Delta^t$. If, in addition, $\Delta^t \subset \Delta^{t*}$ then this is a pure side-effect and the set of the storage elements involved can be expressed as

$$\Delta^s = \Delta^{t*} \backslash \Delta^t$$

where $\backslash$ denotes subtraction of sets. As a minimum, Δ^s may comprise one single bit which is faultily set/reset.

The side-effect fault model can be applied to any network provided its functional description is given, but most favourably it is applied to processor-like networks where data and control logic separate well. Here functional specification mostly consists of definition of instructions and the test is to check their correct execution. (For the sake of simplicity hardware control inputs like interrupt or bus request are omitted now but the method can easily be extended to them, too.)

According to the side-effect fault model, testing is carried out in two steps. First, the correct execution of the instruction under test is checked and then the lack of side-effects. In principle, correct execution can be checked on the basis of the specification without difficulties. Checking the side-effects is not as simple because it requires checking of all internal storage elements. Concerning VLSI circuits, data published for users do not usually contain information on *all* internal storage elements. Only those available for users are *known* and a part of them is *easily accessible.* (As to processors, only registers which can be read out and written to by one single instruction are considered easily accessible.) The existence and function of some not-known storage elements might be reverse-engineered but by no means all of them. Therefore, another method of checking side-effects is proposed.

If the execution of an instruction I_j induces a side-effect S_j — which is an unwanted change of the state of the internal storage elements — then there exists at least one instruction I_k that, in the presence of S_j, can not be executed correctly. Consequently side-effects can be detected in such a way that after the

execution of a given instruction another one is executed, too, and the result is checked. Then this will be carried out for every possible pair of instructions. Hence the structure of such a side-effect detecting program is:

$$\sum_{j=1}^{n} \sum_{k=1}^{n} (I_j I_k) \tag{1}$$

where $\sum$ denotes cyclic execution of instructions.

One question, however, arises: How can a fault be found if the side-effect S_j of the instruction I_j will only be detected by the *second* subsequent instruction I_p in a sequence of $I_j I_k I_p$. The answer is as follows: The test sequence (1) contains each instruction-pair $I_j I_k$ — with $I_j I_p$ among them. If none of these detects the side-effect, then the only possibility is that the execution of I_j induces a side-effect S'_j which does not influence the execution of *any instruction*. Then I_k, while performing correctly, will cause S'_j to change into S''_j which, however, can already be detected by I_p. This is a trilateral interaction and when S'_j comes into being the system enters a *non-specified* state. (The number of possible states of the system has increased by one!) Presuming that no design error exists, it is not probable that some manufacturing failure (*s*) would produce a fault which *only* appears in form of a trilateral interaction.

The complexity of a test program constructed with regard to the above strategy can be expressed in terms of the number of instructions N_i:

$$|T| = O(N_i + N_i^2)$$

The first term stands for the tests checking the correct execution of the instructions, while the second one refers to the instruction-pairs of the sequence (1). Of both terms the latter dominates — $O(N_i^2)$ — and so the same complexity is obtained as was by Thatte and Abraham, but by means of a different (and simpler) reasoning [2].

When testing the execution of instructions the problem of operands has to be mentioned. This leads to the problem of testing the data-logic (data paths, registers and data modifying units). As to data paths and registers, the literature unanimously recommends the so-called data path checking patterns 0101..., 0011..., etc. (checkerboards).

They check the individual bits of the data path and whether there is any short or coupling between them. If this latter is regarded as a side-effect, then it is easily realized that this method is in accordance with the side-effect fault model. As to the ALU, it is a combinational network. Any sensible test for it can only be generated on the basis of structural information.

References

1. YOUNG, J. H.: "Functional Completeness as a Criterion for Digital Testing". Proceedings of the IEEE Test Conference, Cherry Hill, 81–84 (1977).
2. THATTE, S. M.–ABRAHAM, J. A.: "Test Generation for Microprocessors". IEEE Trans. Comput., vol. C-*29*., 429–441, June 1980.
3. BELLON, C. et al.: "Automatic Generation of Microprocessor Test Programs". Proceedings of the IEEE 19th Design Automation Conference, 566–573 (1982).

Dr. Péter GÄRTNER H-1521 Budapest