PERIODICA POLYTECHNICA SER. EL. ENG. VOL. 34, NO. 4, PP. 233-244 (1990)

AN APPROACH TO DEVELOP INTELLIGENT DIGITAL TEST SYSTEMS

R. UBAR

Department of Computer Engineering
Tallinn Technical University
Tallinn, Estonia, USSR

Received: July 2, 1990. Revised: March 22, 1991

Abstract

A new test generation method is developed for digital systems on the basis of alternative graphs. Tests are generated using symbolic signal values and are organized in the compact way — in form of symbolic test programs and data arrays. A new architecture is proposed for test systems which is suited for on-line generating deterministic test patterns in algorithmic way. Special features are implemented in test generator and tester to support event driven testing, which makes it possible to test dynamically devices that work at higher clock rates than the tester does.

Keywords: digital systems, test generation, digital testers, alternative graphs.

Introduction

Automatic test equipments (ATE) can be regarded as important means to help improving the quality of digital devices. The development and application of these systems is expanding as testing requirements become more complex and the need for accurate and reliable test data grows.

Today's electronic products are faster and more complex than ever. Most use microprocessors. As a result, traditional functional 'edge' board testers have reached their limits. Certain classes of faults associated with LSI and caused by interaction between devices have become burdensome for in-circuit testers to test for and detect. Internal self test is a common method of performance verification but it suffers from very limited diagnostic resolution.

An innovative approach that is becoming popular is the use of emulator-based techniques (MORRIS, 1986). The drawback of processor emulation is the requirement of removing the microprocessor. A newly developed emulation technique, called memory emulation, eliminates the need for processor removal (SARGENT, 1983) and contributes to increasing the diagnostic resolution. A disadvantage of the memory emulation approach is the lack of efficient tools for automated generation of test programs.

The limitation of today's CAD tools for test generation are: they do not encompass a strategy for test compaction, the tests are large and unstructured, they have no repetitive patterns, they do not make use of the pattern compaction facilities of testers, they also contain a lot of redundancy (ALBROW, 1983).

By FICHTENBAUM (1987) a novel approach to memory testing was developed which uses test pattern compaction strategy and implements algorithmic test pattern generation within a tester's control structure.The disadvantage of this approach is the use of the same memory for the test program and test data that leads to the need of large random access memories. Also, no methods are proposed for automated test pattern generation.

In this paper a new approach is proposed for testing complex digital devices. This approach integrates automated test pattern generation, test pattern compaction techniques supported by both test pattern generator software and test equipment hardware and emulation-based technique which gives the possibility to run automatically generated test patterns at the DUT's customary speed and in an undisturbed environment. This integration proved to be possible due to the introduction of a new memory pseudo-emulation approach in which the address bus of the DUT is not used.

As the theoretical basis of this approach, the conception of alternative graphs (UBAR, 1988a) is used. Tests are generated using symbolic signal values and are organized in the compact way — in form of symbolic test programs and data arrays. A new architecture is proposed for the test equipment, which is suited for online generating deterministic test patterns in algorithmic way. Special features are implemented in test generator and tester to support event driven testing, which makes it possible to test dynamically devices that work at higher clock rates than the tester does.

Alternative Graphs

Alternative graphs (AG), introduced in (UBAR, 1976), were proposed at first to describe digital devices on the logical level. In relation to this area, AGs and Binary Decision Diagrams (BDD) introduced by AKERS (1978) nearly cover each other. AGs differ from BDDs in their ability to represent the structure of the original circuit (UBAR, 1981).Later, general AGs were introduced to describe a wide class of digital devices on the mixed logical and functional levels (UBAR, 1983 and 1988b).

A given digital device will be represented by a set of AGs, where each AG describes one of the functions of the device. AG can be treated as a program for calculating values of the corresponding function. Internal

nodes in AG are marked by variables and represent conditional branches. The value of the node variable determines the direction of branching. Nodes with n-bit word variables have as maximum 2^n branches. Terminal nodes are marked by functions, variables or constants. To each path from the starting node up to some terminal node corresponds a test vector and the function in the terminal node describes the operation mode for the given module, activated by the vector.

The traditional stuck-at fault model at the gate level is replaced here by a more general model of stuck-at faults on node outputs. A fault can cause a faulty branching out of the path activated by a test vector. The physical meaning of faults associated with node outputs depends on the physical meaning of nodes. Depending on the adequacy of representing the structure of the DUT, the fault model proposed can cover a wide class of structural and functional faults introduced for digital devices at different levels.

Fig. 1 shows a part of the AG-model for the output behaviour of the microprocessor Intel 8080. The complex variable $OUT = DB.AB$ represents the output word of the microprocessor, where two parts, data and address variables are concatenated. The variables $I = I_1 \cdot I_2 \cdot I_3$, PC, A, RP, L, H, $IN2$ and $IN3$ represent correspondingly, the instruction word (with its three fields), program counter, accumulator, register pair, registers L and H and the input data words loaded, correspondingly, during the second and third machine cycles. The variable RP is decoded by its own AG, where the BC, DE, HL represent register pairs and SP denotes the stack pointer. The variable t serves to model the time; its values correspond to different events (machine cycles, in the case of the microprocessor). Physically, the condition $t = j$ means the time instant j when the corresponding activated path in AG actually will be carried into effect (the variable OUT will be equal to the value of the function in the terminal node of the activated path). The formatting variable i is introduced with the aim of compactly describing functions for complex words (without using this variable we should have to describe DB and AB separately by different AGs).

Bold lines in *Fig. 1* denote an activated path on the AG which physically can be interpreted as a test: $I_1 = 0$ & $I_2 = 4$ & $I_3 = 2$ (it means if the instruction code is SHLD — store registers H and L direct) and if $t = 4$ (it means that the 4th machine cycle is being observed) then $DB = L$. Note that for this case only the values of I_1, I_2 and I_3 represent the generated test pattern. The condition $t = 4$ for this test pattern is regarded as an event for which the reaction of the DUT must be observed. Introduction of variables like t to represent specific events in the object, makes it possible to formalize the test generation process for the devices that contain inter-

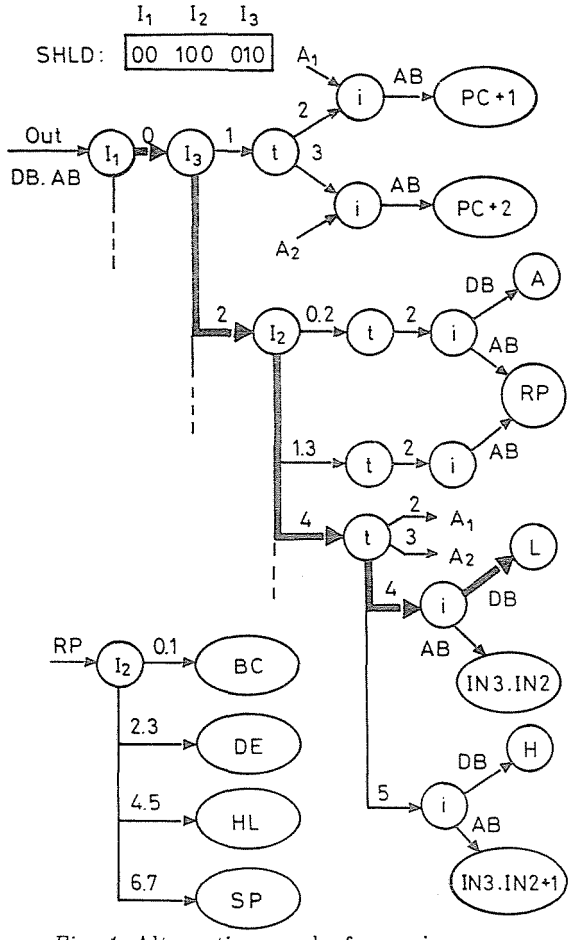

Fig. 1. Alternative graphs for a microprocessor

nal clock generators and so themselves are active in relation to the tester. Traditionally, automatic test pattern generators (ATPG) are used under the presumption that testers are active in relation to the DUT.

Test Generation

The process of test generation can be carried out formally on the AG model by activating successively all the paths on AGs so that all nodes are tested. The test pattern generation procedure for the node m consists of the following steps:

1) activating a path in the AG from the starting node up to the node m;

2) activating nonoverlapping paths for each successor of m up to different terminal nodes;

3) solving the system of inequalities $f(m_i) \neq f(m_j)$ for each pair (m_i, m_j) of terminal nodes reached at the 2nd step (here, $f(m)$ is the function at a terminal node m).

The test pattern generated will be executed cyclically for all the values of the variable at the node m. The algorithm described uses symbolic path activation and organizes the results in a compact way — in a form of symbolic programs and data arrays.

The interpretation of these three steps for different classes of digital devices can be different. For the case of microprocessors, in the first and second steps an instruction sequence (test program) will be generated to set up the needed state of the DUT for executing the instruction to be tested and to observe (or to transfer for observing) the result of the instruction on the output; in the third step the generation of test data (a set of operands) takes place — these operands are needed for activating the faults under test.

An example of carrying out the procedure described is shown in *Fig. 2*. The AG represented in *Fig. 2* is equivalent to the following function:

$$
\begin{aligned}
Y = {} & F_1, \quad \text{if} \quad I = 0, \\
& F_2, \quad \text{if} \quad I = 1 \quad \text{and} \quad x_1 \,\&\, x_2 = 1, \\
& F_3, \quad \text{if} \quad I = 1 \quad \text{and} \quad x_1 = 1 \quad \text{and} \quad x_2 = 0, \\
& F_4, \quad \text{if} \quad I = 1 \quad \text{and} \quad x_1 = 0, \\
& F_5, \quad \text{if} \quad I = 2 \quad \text{and} \quad x_3 = 1, \\
& F_6, \quad \text{if} \quad I = 2 \quad \text{and} \quad x_3 = 0, \\
& F_7, \quad \text{if} \quad I = 3 \quad \text{and} \quad x_4 = 1, \\
& F_8, \quad \text{if} \quad I = 3 \quad \text{and} \quad x_4 = 0.
\end{aligned}
$$

Here we can have the following interpretation: Y is a word which represents a register in a microprocessor, I is the instruction word variable, x_i represent flags and F_j represent elementary functions (microoperations). The bold lines in the graph depict all the activated paths needed to test the faults for the output of the node I at the value $I = 1$. So, carrying out the first two steps of the test generation procedure results in a test vector $T = 11110$ (I, x_1, x_2, x_3, x_4). In the third step we must find the needed data operands for functions F_i by solving the inequalities $F_2 \neq F_1$, $F_2 \neq F_5$ and $F_2 \neq F_8$. To test all the faults related to the node I, we have to solve the inequalities $F_2 \neq F_1 \neq F_5 \neq F_8$ and carry out a set of test vectors $T = var$, 1110 (I, x_1, x_2, x_3, x_4) where var is an element of the set of values $VAR = (0, 1, 2, 3)$.

The following test program results from the test vector T generated:

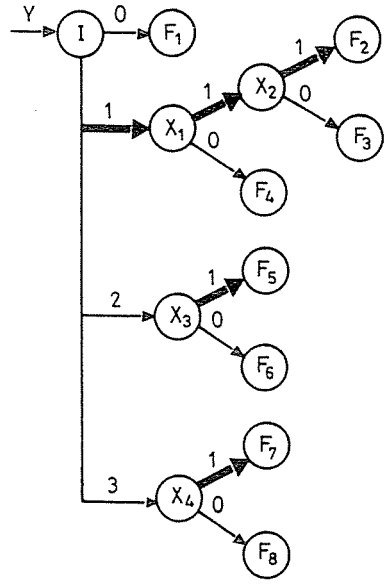

Fig. 2. Alternative graph for a complex function

For all values of *var* from the set *VAR*, do

1) load the needed values for x_1, where $i = 1, 2, 3, 4$, and for all registers which serve as arguments for functions F_j, where $j = 1, 2, 5, 8$;

2) carry out the instructions $I = var$;

3) observe the value $Y = et$, where the variable *et* takes the values from the set $ET = (f_1, f_2, f_5, f_7)$, corresponding to the values *var*; here, by f_j we denote the value of the function F_j.

As we see, the program must be carried out cyclically for all values from the set *VAR*. Here, the values *var* and *et* can be regarded as symbolic values in the test vector for the variables, I and Y. Here, *et* represents the reference values (etalons or expected results of elementary test steps). During execution of the program, the symbolic values *var* and *et* in it will be replaced by the corresponding real values from the arrays *VAR* and *ET*.

In the general case, to solve the inequalities above, we may need more than one set of data operands (UBAR, 1988a). From this situation, the need of the next level cycle (nested loop) arises. So, we have to introduce a new symbolic value *op* for data words as operands (arguments of functions F_j). During the execution of the test program, the values *op* must be replaced by real values of operands from the corresponding array *OP*.

To sum up, in the general case, the test generation procedure above leads to the construction of tests with a specific structure, consisting of the following components:

1) test program P;
2) data array VAR of values to modify P;
3) data array OP of operands and
4) data array ET of reference values (expected results of the test).

As a result of the test pattern generation procedure described above we get the test data automatically in a compact form. The length of this structural test can be measured as

$$L_T = L_P + L_{VAR} + L_{OP} + L_{ET},$$

where L_P is the length of the test program and other L_i denote the number of data words in the corresponding arrays. For the traditional way of the test organization as a simple linear sequence of test patterns, we would have the following estimation: $L_{T'} = L_P \cdot L_{VAR} \cdot L_{OP}$, for the test length with $L_T \ll L_{T'}$. The general test structure is illustrated in *Fig. 3*.

Architecture of the Test Equipment

A new architecture is proposed for test equipments to be used for testing a large class of digital devices like microprocessors, microcontrollers, PCBs, etc. This architecture supports the approach where the off-line automated test data generation, test data compaction, on-line algorithmic test vector generation in the tester and memory pseudo-emulation approaches are integrated.

To minimize testing costs, the tester's throughput must be maximized by reducing the most limiting factor, the time involved in downloading the test pattern data from the system controller's disc memory to the test station electronics. A traditional architecture of the past would typically address these problems by using more and more expensive high-speed RAM to store a greater number of test vectors. This results in high hardware cost and poor system throughput due to the large amount of test pattern data downloaded.

This paper proposes a new architecture (*Fig. 3*) to support the pattern compaction strategy, which is based on the structure composed of a test program and of three different data memories. By this architecture large complex test pattern sequences can be created from less downloaded information. The principles implemented in the architecture result from the test structure which was developed in the previous section.

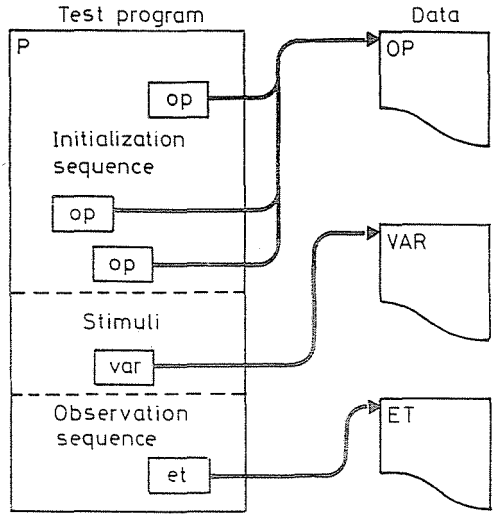

Fig. 3. General test data structure

This paper proposes a new architecture (*Fig. 3*) to support the pattern compaction strategy, which is based on the structure composed of a test program and of three different data memories. By this architecture large complex test pattern sequences can be created from less downloaded information. The principles implemented in the architecture result from the test structure which was developed in the previous section.

The role of the test equipment lies in the execution of the second part of the test generation process — e.g. in the on-line test pattern generation from the test data produced off-line in the way which was described in the previous section. Note that an arbitrary digital device can be represented by alternative graphs. So, the intermediate form for representing test data, proposed here and to be generated off-line on the basis of AG-model, can be regarded as a general form for a wide class of digital devices. Hence, it can be concluded that the algorithmic on-line test pattern generation method implemented in the test equipment is also general — it can be used for a wide class of digital devices, including microprocessor based devices.

Traditionally, algorithmic test generation strategies supported by-hardware were used only for the case of random testing or for the case of simple and regular objects like memory devices, timers, counters, shifters with very simple deterministic test algorithms but not for general case, for example, microprocessors.

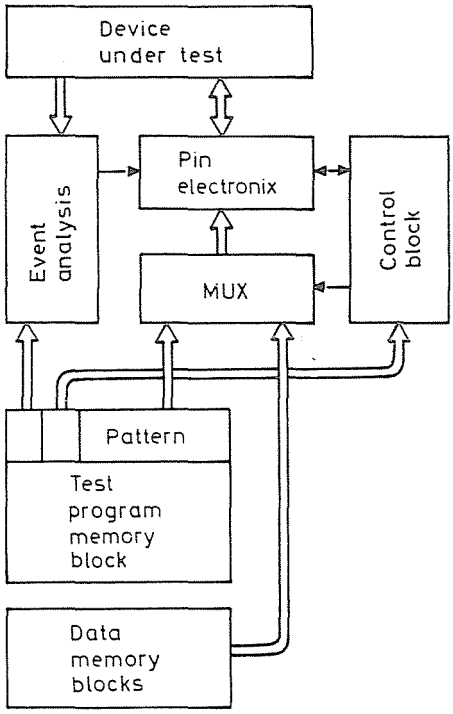

Fig. 4. Block diagram of the test equipment

The algorithmic hardware-supported test generation system proposed here consists of a test program memory block, independent data memory blocks for storing data arrays like *VAR*, *OP* and *ET*, a multiplexer, an event analysis block and a control unit. The simplified block diagram of the test equipment is represented in *Fig. 4*.

Test vectors are composed by the multiplexer and the control unit in the on-line mode using the information stored in the program memory and in the data memory. The program word contains two parts: the data (test vector) to be transferred through the multiplexer to the pin electronics and the instruction part. The instruction determines the operation mode of the multiplexer. In the normal mode only data from the program memory will be transferred and data memory will not be used. The other modes allow to mix the information from the program and data memories. In this case the instruction determines a window in the test vector, where the data normally read from the program memory will be read from the data memory (the exact source — *VAR*, *OP* or *ET* memory block will also be determined by the instruction). The mode of mixing information from different sources

corresponds to the case where symbolic values in the test program are to be replaced by the real numerical values from the arrays *VAR*, *OP* or *ET*.

To give the possibility to run automatically generated test patterns at the DUT's customary speed and in an undisturbed environment, the memory pseudo-emulation approach was introduced, in which the address bus of the DUT when reading tester-supplied memory is ignored. In this approach the tester essentially becomes the program memory. It applies data or instructions to the data bus while activating the ready line and putting the microprocessor in a 'WAIT' state between applications.

Therefore, the tester unit controls the write-access to the data bus to apply instructions and also the read-access to check the results. It must be able to disable on-board memory so that when the microprocessor address bus is cycling through the memory address (program counter), the tester will respond, not the memory in the DUT.

In addition, the tester must be able to write on to the data bus to perform its function during single step testing. The data bus is read by the microprocessor during its program memory accesses when the tester drives the bus.

The advantages of this memory pseudo-emulation approach are the following:

- the diagnostic resolution will be increased because of eliminating addressing functions of the DUT from its operational activity under test;
- the test program can have the form of linear test pattern sequence the generation of which is easy to automate.

Special features are implemented in the test equipment to support by the hardware event-driven testing. A part of the test vector is reserved for programming time instants at which the corresponding test vector is to be executed. The programming is actually reduced to only setting the corresponding values for some bits which code the corresponding events on the output pins of the DUT, characteristic of the time instant of interest.

In such a way the test program consists of IF/THEN/DO clauses. The IF portion of the clause specifies an event; the THEN/DO portion specifies the test vector to be executed when that event occurs.

Using the possibility of event programming introduced into the test equipment, different interface protocols between asynchronously working tester and the DUT can be easily programmed. On the other hand, by introducing the corresponding information related to the interface protocols (like variables t in *Fig. 1*) into the AG-description of the device, it will be easy to generate such conditional test programs automatically.

The event-oriented tester architecture makes it possible to test devices which contain internal clock generators and are not synchronized by test

equipment and it also makes it possible to test dynamically devices that work at higher rates than the tester itself.

Conclusions

New test generation and testing methods for microprocessor based digital devices and digital systems in general were developed. The following advantages of the proposed approach compared to the known solutions can be emphasized:

1. A reasonable compromise between software and hardware supports in automated test generation process was found. As a result, a considerable gain in reducing the memory space and the time needed for loading test data was achieved.

2. The class of digital objects for which hardware-supported algorithmic test generation is reasonable was expanded, compared to the present day practice.

3. Implementing the memory pseudo-emulation technique allows to reach simultaneously high diagnostic resolution (because of minimizing the amount of functionality to be tested in each step) and high test coverage (because of running tests in a DUT's customary speed and in an undisturbed environment).

4. Event-orientation in both test generation and test implementation simplifies conditional test program automated synthesis and makes it possible to test dynamically devices that work at higher rates than the tester itself.

References

AKERS, S. B. (1978): Binary Decision Diagrams. *IEEE Trans. on Comp.*, June 1978, pp. 509–516.

ALBROW, R. (1983): Test Pattern Compaction in VLSI Testers. *Proceedings of the 1983 International Test Conference*, pp. 528–531.

FICHTENBAUM M. L. (1987): A Memory Test Approach for the General-Purpose Digital Board Tester. *Proceedings of the 1987 International Test Conference*, pp. 805–808.

MORRIS, D. S. (1986): In-circuit, Functional or Emulation — Choosing the Right Test Solution. *Computer-Aided Engineering Journal*, June 1986, pp. 94–101.

SARGENT, B.J. (1983): Implementation of a Memory-Emulation Diagnostic Technique. *Proceedings of the 1983 International Test Conference*, pp. 528–531.

UBAR, R. (1976): Test Generation for Digital Circuits Using Alternative Graphs. *Dig. of Technical University Tallinn (USSR)*, No 409, pp. 75–81. (in Russian)

UBAR, R. (1981): Testing of Digital Devices. parts I and II. *Tallinn Technical University*, pp 226. (in Russian)

UBAR, R. (1983): Test Pattern Generation for Digital Systems on the Vector AG-model. *Proceedings of the Fault Tolerant Computing Symposium*, Milano, pp. 347–377.

UBAR, R. (1988a): Alternative Graphs and Technical Diagnosis of Digital Objects. *Electronic Techniques*, Vol. 8, No 5 (132), pp. 33–57. (in Russian)

UBAR, R.– LOHUARU, T. (1988b): Description of Digital Objects with AG for Test Generation Purposes. *FTSD*, Suhl (GDR), pp. 157–163.

Address:

Raimund UBAR
Department of Computer Engineering
Tallinn Technical University
Ehitayate tee 5,
200108, Tallinn, Estonia, USSR