TESTING OF MICROPROCESSORS

A. PATARICZA

Department of Measurement and Instrument Engineering,
Technical University, H-1521 Budapest

Received June 5, 1986
Presented by Prof. Dr. L. Schnell

Abstract

With the growing use of the microprocessors the problematics of testing become more and more important for the reliability of the instrumentation. The paper gives a survey of the usual strategies and methods for CPU testing in microprocessor controlled equipment. The effects of the state-of-the-art field service methods on the self-test technology are discussed. Description of a new algorithm based on information compression is given together with some related realization experiences.

Introduction

The advent of the microprocessors caused a radical change in the architecture of the measurement equipment. The majority of the control and data processing functions are realized by the microcomputer. The earlier strong correlation between the function and the structure has become loose. The complexity of the realized control sequences increases radically with the use of sophisticated algorithms, the substitution of analog data processing with digital etc.

The functional complexity increases the ratio of the possible difficult-to-diagnose faults and therefore the cost and efforts needed to the fault localization and repair. Simultaneously, the wide-spread application of the microprocessor controlled instrumentation requires a certain reliability growth.

The solution of this conflict is based upon the (at least partially) self-testing device-building principle. A self-testing system incorporates the capability of checking its functionality without external resources, using the built-in intelligence.

The widely used field service methods are based upon the extension of the error detecting self-test to a diagnostic level [1–3].

The main problems related to CPU testing

Usually, the microprocessor controlled equipment incorporates only a single intelligent device, the central processing unit (CPU). Thus, the CPU test becomes an essential problem in the self-test process, because the test of other components in the system uses the CPU as resource. Despite of its fundamental importance there are essential problems related to the CPU testing especially in the user environment:

- No unambiguous statistical data are available to the user related to the reliability and defects of the CPU.
- The efficiency of the classical, gate- and flip-flop level methods is strictly limited due to the unknown internal structure of the CPU, the order of the problem, the poor controllability and observability and the dubious validity of the generally used stuck-at type fault model.
- For the manufacturers the test-friendliness in user environment is no design criterion (with the exception of some newly developed CPU's [4]).

Strategies for CPU test

Due to the difficulties mentioned above, different approaches exist for the CPU testing problem.

- The „*non-testing*" strategy assumes high reliability of the CPU and the easy-detectable manifestation of its faults (eg. catastrophic program sequence distortion). The main inconvenience of this method is the uncertainty in the existence of faults (eg. pattern sensitivity etc.).
- In *random-testing* we assume that the controlled execution of any, sufficiently long program exposes with high probability the potential faults. The advantage of the method is that it requires no fault modelling and test-vector evaluation. The disadvantage is the insufficient fault coverage in the case of real program length and run-time.
- By *heuristic instruction-level* testing the proper execution of all elements in the instruction set of the CPU is checked. Some instructions (the kernel of the test) are assumed to be fault-free. The other instructions are examined in an increasing "complexity-order" using only instructions previously classified as "good" (belonging to the kernel or pretested).
- The *algorithmic methods* give solution for a processor-category starting from a generalized model of the CPU and its faults.

In the next sections this important strategy will be dealt with in details.

Algorithmic CPU test

The high level description and modelling of the CPU elements became more and more necessary due to the difficulties mentioned above. First fundamental results in this field were obtained by Thatte and Abraham [5].

Their method has the following features:

— The data flow mechanism of the CPU (register set, the effect of transfer and branch type instructions causing data flow) is modelled by a so-called S-graph, evaluated from the user level specification.

— Functional fault models and test algorithms were developed for the subset of the CPU structure and operations described above. The design criterion was the high fault coverage. The fault diagnosis was not a design goal.

— The model is independent from the testing environment, the mechanisms for test generation and response evaluation influence it only implicitly.

— The data manipulation (eg. arithmetical, logical) instructions are not included in the testing.

The faulty functions in the model could be:

— register decoding
— instruction decoding
— data storage and transfer.

Test of register decoding

The fault model for the register decoding is:

— no register selection for some register(s)
— instead of the addressed register another one is selected,
— in addition to the addressed register some other(s) is selected.

The register set treated functionally is an embedded RAM, so its test can be based on the well-known memory test algorithms, modified due to

— the implicit addressing of the registers;

— the indirect accessibility of some registers (their read and/or write can be accomplished only through other registers eg. the I interrupt register of Z80)

— the autonom manipulations on some registers by the CPU hardware (eg. program counter) [6]).

The realisation of the usual memory test algorithms requires only small amount of program space due to the regularity of the structure and of the pattern to generate. In the case of CPU test this regularity can not be utilised

for statical program length reduction (eg. by cycle organisation) primarily due to the implicit addressing. Accordingly, the register test is statically long (eg. for the Z80 CPU 1.5 Kbytes).

Test of the instruction decoding

The fault model for the instruction decoding is:
— no instruction execution,
— instead of the selected instruction some other is executed,
— in addition to the selected instruction some other is executed.

The fault model described above seems to be simple, but the testing of the control section performing the instruction decoding is significantly more difficult than that of the register decoding. The cause of this complexity is the indirect observability of the control signals. Accordingly, the test generation algorithm is difficult to survey and handle. The complexity is well-characterized by the fact that the length of the test program segment for the instruction decoding of an 8 bit microprocessor (developed by the authors of [5]) is 8 Kbyte long. Therefore the rationalization of the testing of the instruction decoding is subject of intensive research [7–11].

The remarkable approach in [7] assumes essentially a microprogrammed CPU structure, but it is adaptable for microprocessors with random logic control section. An instruction is modelled as a series of microinstructions, and the microinstructions are sets of parallel executed microorders. This is based on that physical correspondence, that an instruction is a series of different machine states and during any machine state the changes in internal control signals are performed simultaneously.

The fault model for the control section from [7] is
— one or more microinstructions are inactive,
— one or more microorders or
— a microinstruction is additionally active.

The fault detection algorithm for the above described fault model is significantly simpler than that of the S-graph model and it provides a good fault coverage.

The fault model of the data storage and -transfer in the S-graph model contains the following faults:
— stuck- at 0 or 1 faults in any register(s) or on the links between them
— OR or AND type coupling between bits of a register or of a link. The fault detection algorithm is quite simple (a test vector series must be transferred through every input-output path).

We may conclude that the main advantage of the S-graph type methods is the possibility for automatic test-pattern generation from the user level description of the CPU.

The disadvantage of this methods are:
— the great redundancy caused by the generalization (the CPU is modelled as a black box with functional description, but without any structural information);
— too long test sequences, resulting from the resource minimalization principle;
— some functions (arithmetical-logical unit, interrupt system) are not included in the scope of the method.

The specification of CPU (self-) test programs

For the implementation as self-test the usual algorithms must be significantly modified to assure protection for catastrophic program failures and reliable result comparison.

The test algorithms and their realizations are essentially influenced by [11]

— the resources allowed for the test,
— the method of the test response evaluation.

Historically, the CPU testing is evaluated from the checking technology of the mainframes in computing centers. This method is based upon the execution of the self-test routines on the potentially faulty computer, initiated by the operator. The field service repairs the wrong CPU function with board or component exchange upon the basis of the error messages produced by the self-test so the main criteria for this tests are the safe execution and the good diagnostics.

The set of the resources used by the test without previous checking (the kernel of the test) is usually minimized in correspondence to the above mentioned criteria, because

— their fault and the faults of the CPU are mostly inseparable;
— they (possibly) increase the probability of a catastrophic distorted, incontrollable program execution (eg. a fatal program execution error caused by a return from subroutine to an address stored in a faulty RAM).

Most algorithms in the literature are based upon the above mentioned criteria without revision and therefore upon the kernel-minimization principle. The kernel of the test contains usually some basic functions of the CPU, a part of the test ROM and the CPU-ROM link. The extension of the kernel means the addition of the subsequently tested CPU resources (eg. instructions, registers) to the kernel.

A fatal error in the program execution is relative easy to detect (eg. by timeout checks, by the checking of the control sequences or by the absence of

the "good CPU" signal as result etc.). Therefore the minimization principle is mainly implied by the requirement for diagnosis.

For the self test of microprocessor based equipment usually there is no need for diagnosis, its goal is only fault detection. For this application field the fault diagnosis can be executed easily even in the case of the most primitive field service technology (eg. by the exchange of the "dubious" CPU to a known-good one). In the case of a modern field service technology the additional resources for the fault localization (eg. diagnostic program and other hardware resources etc.) are provided fault-free by a service instrument. The absence of the requirement for diagnosis provides a good possibility for the simplification of the algorithms. Therefore, the essential revision of the previous algorithms became reasonable for the author.

For this case the use of arbitrary — even potentially faulty — resource is allowed, if

— it causes no significant increase in the probability of fatal program execution errors,

— the error coverage in the evaluation of the test responses is sufficiently high.

The evaluation of the test responses

For the evaluation of the test responses the observability of the results generated in the CPU registers and their comparison with the references must be assured.

The possibilities for response evaluation are:

— *External error detection*, in which case the comparisons between the responses and the reference values are executed by a special response-evaluator hardware unit. This hardware redundancy results in a simplification of the testprogram (by the absence of the programmed evaluation of the go/nogo syndrome) and in some increase of the reliability.

In the case of *internal error detection* the test responses are evaluated by the test program itself. The reliability of the test is essentially determined by this phase, therefore the resources used for it must be thoroughgoingly pretested.

The possibilities to assure observability for the test responses are:

— *Control-sensitization*, by which the program execution-sequence is modified by the results (typically by means of result-reference comparison and a corresponding conditional branch execution)

— *Data-sensitization*, in which case the results (or the result-reference differences) are saved in each test phase and the test evaluation is executed as the final phase of the test on the stored data.

In microprocessor controlled equipment the usually applied error diagnosis method is internal, on account of its smaller cost. The usual realizations are control-sensitized, resulting in smaller requirements is resources. (For data sensitization storing of the results until the response evaluation must be solved.)

The main characteristics of the realized self-test program

The CPU self-test program developed by the author is an element from the self-test system of the MMT Microprocessor Application System. Its basic specification was essentially similar to the usual self-test criterion system for microprocessor controlled equipment.

— The test must assure a fault coverage as good as possible.
— Its goal is only fault detection. No fault diagnosis is required.
— The test runs directly after power-on, therefore the run-time is in wide range unlimited.
— In the absence of external checking hardware it works upon the basis of internal error detection.
— The program must be as short as possible.

The good fault coverage can be assured only by an algorithmic method.

The absence of requirements for diagnosis and — as described earlier — the addition of resources allows the use of compressed data sensitization. The principle of this method is the storage of the generated partial result series in a compressed form instead of the full extent. The partial results are compressed continuously at the generation phases. The final result is evaluated in the final phase of the test. In the realized program the information is compressed by a 16 bit, parallel software signature analyzer [12]. Therefore, the required reference storage is — instead of several hundred bytes — only 16 bit. The reference value is generated by a program execution on a known-good reference system, and by a subsequent storage of its result (stored hardware reference). The information loss related to the response compression and the potential faults in the compression algorithm can result in an additional fault escape (nondetected faults). For a 16 bit long signature — as used in our algorithm — the probability of the identity of the signature of a random result with the reference is 2^{-16} ($<0.002\%$). The usual characteristic of the algorithms is 90–95% fault coverage. Accordingly, the increase is negligible compared with the 5–10% fault escape of the algorithm itself. Therefore, even the information compression can be untested or faulty. The reliability of the response evaluation is determined by the instruction sequence comparing the measured result and the reference (two 16 bit words). This can be easily pretested by means of control sensitization.

For the realization a so-called copy RAM of the same size as the register set of the CPU is assigned in the memory. In each response-generating phase the whole register-set of the CPU is stored into this RAM. In the memory a location for the signature is reserved. The execution of a typical test program segment has the following execution sequence (Fig. 1):

1. At first a test instruction sequence segment is executed, usually generating results in the CPU registers;
2. The whole register set of the CPU is saved into the copy RAM;
3. The signature containing the accumulated compressed information is modified according to the contents of the copy RAM;

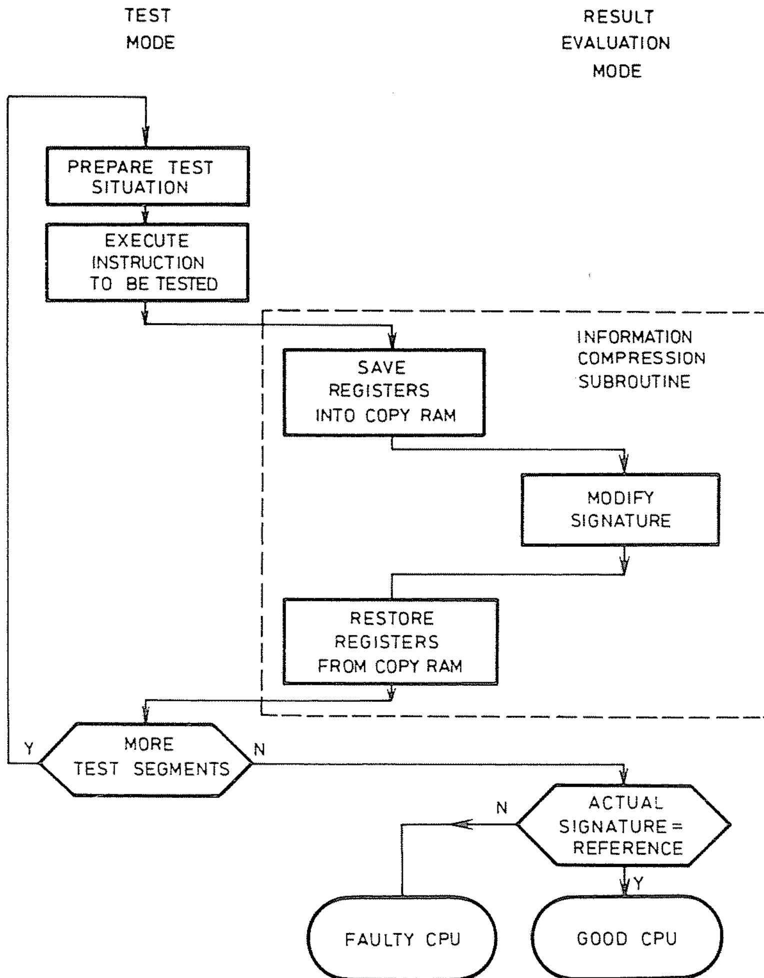

Fig. 1. Typical test program flow chart (simplified)

4. The register set is restored from the copy RAM and the test continues.

Steps 2–4 emulate an external response evaluator and a data compressor by the CPU, with observability of the content of the internal CPU registers. It is important that steps 2–4 are the same for all test segments, therefore they can be implemented as a single subroutine for the whole test. Each test segment contains only a corresponding subroutine call instruction. This results in a significant reduction in the static program length at the expense of increased run-time. The fault coverage is randomly increased (the data from all registers are always evaluated, independently of which register contains the result).

For protection from distorted program execution due to faults in the additional resources in theory the previous testing of the RAM locations used for program- and test control (eg. program counter-copy, cycle variables in pattern generation etc.) is sufficient. Due to the small size of the used RAM and the easy implementability of memory tests with good fault coverage the test of the whole used RAM area (for Z80 cca 40 bytes) is reasonable.

The main steps of the CPU test are as follows:

— Testing the response evaluating instructions (comparison and conditional jump) by control sensitization,
— Checking the ROM containing the test program by means of comparison between the burned-in reference and the generated ROM signature,
— Testing the RAM used by the test,
— Register and data path test,
— Test of arithmetical and logical unit (ALU) and the flags,
— Evaluation of the compressed response.

The separate register and data path tests of the S-graph method are extensive and complicated. The implementation of the following algorithm with similar fault coverage was reasonable:

— The patterns of a memory test with good fault coverage [13] are stepwise generated in the copy RAM (with the exception of the PC copy).
— The content of the copy RAM is loaded into the register set.
— The register save — information compression — register restore sequence is executed as described earlier.
— The next memory test pattern is generated etc.

The "black-block" tests of the ALU were of extreme long run-time, eg. the exhaustive test of an operation on two, eigth bit wide operation is of 2^{16} test steps. Accordingly, the implemented test is based upon assumption for the structure and faults of the ALU — as usual for the heuristic tests. (Eg. for the logic operations on two operands — with the assumption of bitwise operation, stuck-at and coupling faults — 10 patterns are sufficient as test).

Conclusions

The algorithm presented in this paper is based on the usual specification for self-test in microprocessor controlled instrumentation. It is based upon the "no-diagnosis" requirement for this category, and accordingly by the fundamental use of information compression methods it provides an important decrease in static program length and complexity. The method of the test program development use no CPU-specific features, so the algorithm can be adopted to each CPU type.

References

1. HORVÁTH, I.–PATARICZA, A.–SELÉNYI, E.: Field Service System for Intelligent Measuring Equipment; Periodica Polytechnica-Electrical Engineering, Vol *28*. No 1., 29–41.
2. JOBBÁGY, A.–PATARICZA, A.–SELÉNYI, E.: Self-test and Field Maintenance System for Microprocessor Controlled Instrumentation; Xth World Congress of IMEKO, 1985., Vol. 5., 69–77.
3. JOBBÁGY, A.–PATARICZA, A.–SELÉNYI, E.: Service and Maintenance for a Microprocessor Application System: Problems and Solutions; ATE'85 Conference, 1985., pp. 3.5.1–3.5.16.
4. KUBAN, J. R.–BRUCE, W. C.: Self-Testing the Motorola MC6804P2; IEEE Design & Test, May 1984., pp. 33–41.
5. THATTE, S. M.–ABRAHAM, J. A.: Test Generation for Microprocessors; IEEE Transactions on Computers, Vol C-*29*., No 6., 429–441 (1980).
6. SUN, Z.–WANG, L. T.: Self-Testing of Embedded RAMs; IEEE 1984 International Test Conference, pp. 148–156.
7. BRAHME, D.–ABRAHAM, J.: Functional Testing of Microprocessors; IEEE Transactions on Computers, Vol C-*34*., No 6., 475–485 (1984).
8. THEVENOD-FOSSE, P.–DAVID, R.: Random Testing of the Control Section of a Microprocessor; FTCS-13., IEEE 13th International Symposium on Fault-Tolerant Computing, 1983. pp. 366–373.
9. SALUJA, K. K.–SHEN, L.–SU, S. Y. H.: A Simplified Algorithm for Testing Microprocessors; IEEE 1983 International Test Conference, pp. 668–675.
10. ANNARATONE, M. A.–SAMI, M. G.: An Approach to Functional Testing of Microprocessors; FTCS-12, IEEE 12th International Symposium on Fault-Tolerant Computing, 1982. pp. 158–164.
11. ABRAHAM, J.–PARKER, K.: Practical Microprocessor Testing: Open and Closed Loop Approaches; IEEE COMPCON Spring, 308–311, 1981.
12. PATARICZA, A.: Some Properties of Signature Analysis Newsletter; Technical University of Budapest, *4*., 27–31 (1986).
13. MARINESCU, M.: Simple and Efficient Algorithms for Functional RAM Testing; IEEE 1982 International Test Conference, 236–239.

András PATARICZA H-1521 Budapest