The ACT (Automatic Computer Troubleshooting) Project

Dean Lance Smith

Department of Electrical Engineering
Memphis State University
Memphis, TN 38152

ABSTRACT

The goal of the ACT (Automatic Computer Troubleshooting) project is to develop portable software that will help diagnose computer failures. Some of the software will control test instruments and help the user make the diagnosis. Other software will predict the results of signature analysis tests. Functional tests of a target system using the latest available algorithms will be supported. The project will also explore new test algorithms and techniques for diagnosis.

INTRODUCTION

The ACT project started in January, 1989. The project's goal is to develop portable software that will help diagnose computer failures. The project will also explore new test algorithms and techniques for diagnosis.

The software will perform several functions. It will support and control several pieces of diagnostic equipment such as an emulator, signature analyzer, logic analyzer, EVM (Electronic Volt Meter) and DTDM (Digital Time Domain Meter). The software will permit data about a SUT (System Under Test) to be loaded into a data base. The data base can be queried by the instrument control programs to guide the user to a fault. The data base will also contain test routines that use the most effective test algorithms.

Test programs can be compiled with a cross compiler and loaded into a SUT with an emulator. Signatures for a new system can be predicted, and a signature analyzer can be used to confirm the signatures or troubleshoot a defective system. A logic analyzer can be used to diagnose restart failures caused by defects in RAM or ROM. The EVM and DTDM will help diagnose power supply, timing and wave shape failures.

The ACT project is also developing new algorithms for testing or predicting the test results of a system. Algorithms for testing ROMs, disks and interfaces appear ripe for development. Further progress on algorithms for predicting signatures is expected.

The details of the project are discussed in this paper. A brief report is given about the software and test algorithms under development.

SYSTEMATIC APPROACHES

Three systematic approaches can be used to diagnose computer failures: 1) Self-test or diagnostic programs, 2) Signature analysis and 3) Single-step.

[1] All three approaches will be supported by the ACT software.

These major approaches are used to isolate logic faults to a small section of the system and sometimes to the defective component. Small hand tools, such as a logic probe, logic pulser and digital current tracer, are used to isolate the defective component, if necessary. Occasionally an EVM (Electronic Volt Meter), oscilloscope or DTDM (Digital Time Domain Meter) is needed to help isolate a defective component.

Self-Test

The self-test or diagnostic program approach executes a test program on the SUT to verify that some function of the system (RAM, ROM, an interface, etc.) is working properly. The results are reported to the user's console.

No system can be exhaustively tested. [2] This is especially true when testing RAM and the CPU. Several algorithms that are relatively quick and effective when testing RAM have been developed in the last few years. (See Abadir and Reghbati [3] for an excellent survey.) Many new algorithms need to be developed, especially for some types of RAM failures, ROM and disk drives. New test
algorithms are expected to be developed as part of the ACT project. All effective algorithms will be programmed in portable software.

In practice, the ROM, RAM and interfaces of the system are tested using self-test. Rarely is the CPU tested. Testing the CPU takes too long. A CPU test program is also very long and difficult to write. As a result, the assumption is usually made that the CPU is working if the system passes the remaining tests. This is usually a fairly good assumption.

Self-test is quick. It is good at rapidly isolating defects to a section of the system and sometimes even identifying a defective component. However, the system must restart for a self-test program to run.

Signature Analysis

A test program is also run on the SUT for signature analysis. However, the program is run in a continuous loop. In addition, the results are not reported by the test program. They are measured with an instrument called a signature analyzer. [4]

Signature analysis will successfully diagnose some types of failures that would prevent a self-test program from working. ROM failures affecting the restart program and upper address decoders for RAM are two examples. Signature analysis is capable of isolating a defect to a node and sometimes even a defective component.

However, signature analysis is slow at isolating defects to an area. It also requires a lot of advanced preparation. Test programs need to be written and verified. Signature tables need to be measured and verified.

Signatures are deterministic and are fairly easy to calculate, especially for the first signature analysis test, which is sometimes called the free-run or kernel test. [5] This cuts down the time and expense of developing a signature analysis package.

New algorithms to predict signatures are expected to be developed as part of the ACT project. SIGNATURE GENERATOR will be linked with a data base and signature analyzer control software. The combination will speed up the development of signature analysis packages. It will also permit the software to guide the troubleshooter through the diagnosis. Software will also be written to implement test algorithms suitable for signature analysis.

Combining Self-Test and Signature Analysis

Self-test and signature analysis are to some extent complementary. Self-test is very quick at locating a hardware failure in a particular area but usually gives little or no additional information about the defect. Signature analysis is slow at isolating a defect to an area but will identify the defective node and often the defective component.

The two in combination can produce MTD (Mean Time to Diagnosis) of about 15 minutes. The MTD cited assumes a single board microprocessor system consisting of approximately 20 integrated circuits. It also assumes that support software and documentation have been properly prepared and that service personnel have been properly trained. This figure is based on the self-test and signature analysis software being built into the systems in ROM.

Similar MTDs would be a reasonable objective of the ACT system. However, less highly skilled personnel should also be able to obtain these MTDs provided the ACT software is properly customized for the target system.

Single-Step

The single-step approach uses a logic analyzer or a data latch to follow a test program as it executes on the SUT. [1] The instrument is configured to trace either the machine cycles or the instruction fetch cycles of the software. The program is traced to a defective instruction fetch or machine cycle. The fault can then be diagnosed if the instrument's display is correctly interpreted.

The test program is often an application program in the SUT. This approach is mostly used in development servicing since it does not require the advanced preparation that self-test and signature analysis require. It is used occasionally in field servicing to diagnose some types of restart failures [6] and intermittent failures.

The single-step approach requires more operator skill than the self-test or signature analysis approach. The user must understand the operation of a logic analyzer, the behavior of the machine cycles of the SUT and assembly language programming for the machine.

The single-step approach has other drawbacks. It is slow, primarily because of the time required for the operator to
analyze the data. An assembler listing output must be available for the program being traced. The program being traced must stimulate the fault.

The single-step approach can spot most computer failures. It is the only approach that will spot some failures.

The ACT project will develop software to control a logic analyzer and automate the single-step approach. The software will have provisions for loading and analyzing machine code from the SUT (System Under Test). The analysis can be used to guide a troubleshooter through the single-step process.

**DIAGNOSIS STRATEGIES**

Several deterministic troubleshooting strategies can be implemented with the ACT software.

**Signal Tracing**

Signal tracing assumes the SUT is a linear cascade of stages or modules. An appropriate test signal drives the input of the system. An appropriate measurement instrument is used to measure the progress of the signal through the system by measuring the output of each stage or module. The measurements start at the output of the first stage and proceed to the output of the system. If any interstage signal is found to be defective, the preceding stage is assumed to have failed.

Self-test, signature analysis and single-step all use signal tracing. In each approach, the software produces the signals in the hardware. The detection instrument varies with each approach. Self-test uses the software running on the system hardware to detect the output signals. Signature analysis uses a signature analyzer. Single-step uses a logic analyzer or data latch.

Signal tracing will not work if there is feedback from a later stage to an earlier one. The distortion produced by a defective stage will appear everywhere in the feedback loop.

Signal tracing can still be applied if the feedback loop can be broken where it connects to the earlier stage. Then the system can be traced from input to output and from the start of the feedback loop to the break.

Because of its importance, signal tracing will be programmed into the ACT software.

**Signal Injection**

Signal injection also assumes a system composed of a linear cascade of stages. A signal detector is placed at the output. A signal is injected at each of the connections between stages starting at the input to the final stage. If the output is distorted, the stage where the signal is being injected is assumed to be defective.

Signal injection will fail if the system contains a feedback loop. However, the technique can often be applied if the feedback path can be broken at its input. Then the system can be traced from the output to the input and back around the feedback path from where it connects to the input of a stage to the break.

Signal injection is not used as often as signal tracing. Signal injection is usually a more difficult technique to apply. General purpose signal injection equipment usually costs more than general purpose signal detection equipment.

Signal injection is rarely used in computer troubleshooting. There are some exceptions. A digital signal pulser will sometimes be used along with a logic probe or digital current tracer to isolate open outputs or shorted nodes when the defect has been isolated to an area. Some interfaces are sometimes tested using signal injection. It is not anticipated that signal injection will be programmed into the ACT software.

**Partitioning**

There are two partitioning strategies, functional and physical.

**Functional**

Functional or logical partitioning is sometimes used when a system is so complex it cannot be interpreted as a linear cascade system, even if its feedback loops are broken. It is also used when it is difficult to break the feedback loops, or breaking the feedback loops would produce a meaningless test. All of these justifications apply to a computer system, and functional testing is often appropriate.

In functional partitioning, a system is divided into its logical functions and each component is tested separately. Self-test does this to a certain extent, although the CPU and in some cases ROM or RAM are always an implicit part of the function being tested.

**Signature analysis**

Signature analysis does this type of partitioning in two ways. It tests each
function separately as self-test does. It also separates the write and read operations from each other when testing a function.

Single-step also performs functional or logical partitioning. It can be used to separate read and write operations. It can be used to separate operation code fetches from other memory read operations. Memory and I/O operations can be separated in systems with separate I/O spaces. The reads and writes of ROMs, RAMs and peripheral chips can be separated. However, a logic analyzer does require the user to make the separation either by understanding how the system works and properly connecting the equipment to the system or properly interpreting the data displayed.

Functional partitioning will be supported by the ACT software because of its importance to computer troubleshooting.

Physical partitioning is a technique for speeding up signal tracing and signal injection. The most common strategy is half-splitting. A cascade system is tested at the half point. The defect is isolated to one-half of the system as a result. The half with the defect can be tested at its half point, etc. until the defect is found.

Physical partitioning is equivalent to searching a list. Half-splitting is equivalent to a binary search. Half-splitting or a binary search is an easy algorithm to implement without a computer. It can be shown, however, that other search strategies are better. [6] Unfortunately, the better search strategies are more difficult to implement without some kind of calculation aid. Optimum physical search strategies are therefore ideal candidates for computer software and they will be used in the ACT system.

Restart Failures

A significant number of computer failures are restart failures. The system fails to restart after a power failure or after it is turned on.

There is a systematic combination of functional partitions, signal tracing and signal injection that can be used to diagnose the problem. [6] This strategy will be written into the ACT software. A general restart failure diagnosis will be built into the software. The strategy can be customized for a particular system.

TARGET SYSTEM DESCRIPTION

The ACT software will access tables that will describe features of a particular target SUT. The user will be permitted to alter the tables that describe the SUT. Some general information applicable to all systems will be built into the software. This information will be useful even if no tables have been generated for the target system.

TEST EQUIPMENT SUPPORT

The ACT software will support several pieces of equipment through either a GPIB (IEEE-488) or RS-232C interface. The software will support an EVM (Electronic Volt Meter), DTDM (Digital Time Domain Meter) and an emulator in addition to the signature and logic analyzers discussed earlier.

An EVM is needed to diagnose some restart failures. [6] A DTDM is sometimes useful for measuring frequency, rise time, pulse width and other properties of pulse waveforms. An emulator will be used to insert test programs into SUTs once they have been cross compiled on the host system.

Control programs will be built into the ACT software to integrate the instruments into the system's data base. The user will be advised on where to make measurements when developing a servicing package or diagnosing a failure in a system. The control programs will support equipment from several vendors.

PROGRAMMING LANGUAGE

The ACT software will be written in the C programming language. C is a higher level language and is very portable. Yet it has many features, such as bit manipulation and pointer variables, that are usually associated only with lower level languages. [8]

PROGRESS

Several pieces of software have been developed and tested since the start of the project. ATS and ATS+, [9] MATS and MATS+, [10] and marching [11] RAM test algorithms have been written in C. The software will be republished in users groups.

A control program for a signature analyzer and a signature analyzer test program are under development. An improved version of SIGNATURE GENERATOR has been written. All of these programs should be available by the end of this year.
Progress has been made on improved ROM test algorithms. A paper reporting the results is in progress.

ACKNOWLEDGEMENTS

The author wishes to thank Mohammad Khurrum, Chaiyos Ruengsakulrach, Guiwei Su, Saiya Gadde and Syed Haque for their assistance in this project. The author also wishes to thank Ann Brock for her editorial assistance.

REFERENCES


