Verification of the First Fault-tolerant VAX System

By William F. Bruckert, Carlos Alonso and James M. Melvin

Abstract

The fault-tolerant character of the VAXft 3000 system required that plans be made early in the development stages

for the verification and test of the system. To ensure proper test coverage of the fault-tolerant features, engineers would build fault-insertion points directly into the system hardware. In the verification process, test engineers would use hardware and software fault insertion in directed and random test forms. A four-phase verification strategy was devised that would ensure the VAXft system hardware and software was fully tested for error recovery that is transparent to applications on the system.

Introduction

The VAXft 3000 system provides transparent fault tolerance for applications that run on the system. Because the 3000 includes fault-tolerant features, strategy outlined a four-phase approach which would require hardware to be built into the system specifically for test purposes.

This paper presents a brief overview of the VAXft system architecture and then describes the methods used to verify the system's fault tolerance.

VAXft 3000 Architectural Overview

The VAXft fault-tolerant system is designed to recover from any single point of hardware failure. Fault tolerance is provided transparently for all applications running under the VMS operating system. This section reviews the implementation of the system to provide background for the main discussion of the verification process.

The system comprises two duplicate systems, called zones. Each zone is a fully functional computer with enough elements to run an verification of the system was unlike that ordinarily conducted on VAX systems. To facilitate system test, the verification

operating system. These two zones, referred to as zone A and zone B, are shown in Figure 1, which illustrates the duplication

of the system components.

The two independent zones are connected by duplicate cross-link cables. The cabinet of each zone also includes battery, power regulator, cooling fans, and AC power input. Each zone's hardware has sufficient error checking to detect all single faults within that zone.

Figure 2 is a block diagram of a single zone with one I/O adapter. Note the portions of the zone labeled dual-rail and single-rail. The dual-rail portions of the system have two independent sets of hardware performing the same operations. Correct operation is verified by comparison. The fault-detection mechanism for the single-rail I/O modules combines checking codes and communication protocols.

The system performs I/O operations by sending and receiving message packets. The packets are exchanged between the CPU and various servers that include disks, Ethernet, and synchronous lines. These message packets are formed and interpreted in the dual-rail portion of the system. They are protected in the single-rail portion of the machine by check codes which are generated and checked in the dual-rail portion of the machine. Corrupted packets can be retransmitted through the same or alternate paths.

In the normal mode of fault-tolerant operation, both zones execute the same instruction at the same time. The four processors

system. A defective CPU module and its memory are automatically removed from service by the hardware, and the remaining CPU and memory continues processing.

(two in each zone) appear
to the operating system
as a single logical CPU.
The hardware supplies the
detection and recovery
facilities for faults
detected in the CPU and
memory portions of the

Error handling for the I/O interconnections is managed differently. The paths to and from I/O adapters are duplicated for checking purposes. If a fault is detected, the hardware retries the operation. If successful, the error is logged, and operation continues without software assistance. If the retry is unsuccessful, the Fault-tolerant System Services (FTSS) software performs error recovery. FTSS is a layer software product that is utilized with every VAXft 3000. It provides the software necessary to complete system error recovery. For system recovery from a failed IO device, an alternate path or device is used. All recoverable faults have an associated maximum threshold value. If this threshold is exceeded, FTSS performs appropriate device reconfiguration.

Verification of a Fault-tolerant VAX System

This section entails a discussion of the types of system tests and the fault-insertion techniques used to ensure the correct operation of the VAXft system. In addition, the four-phase verification strategy and the procedures involved in each phase are frequently in computer system verification and follow a strict test sequence. Complex systems, however, cannot be completely verified in a directed fashion. [1] As a case in point, an operating system running on a processor has innumerable states. Directed tests verify functional operation under a particular set of conditions. They may not, however, be used to verify that same functionality under all possible system conditions.

In comparison, random testing allows multiple test processes to interact in a pseudo-random or random fashion. In random testing, test coverage is increased with additional run-time. Thus, once the proper test processes are in place, the need to develop additional tests in order to increase coverage is eliminated. This type of testing also reduces the effects of the biases of the engineers generating the tests. While directed testing can provide only a limited level of coverage, this coverage level can be well understood. Random testing offers a potentially unbounded level of coverage; however, quantifying this coverage is difficult if not reviewed.

There are two types of system tests: directed and random. Directed tests, which test specific hardware or software features, are used most

impossible.

To achieve the proper level of verification, the VAXft verification utilized a balance of directed and random testing. Directed testing was used to achieve a certain base level of functionality, and random testing was used to expand the level of coverage.

To permit testing of system fault tolerance in a practical amount of time, some form of fault insertion is required. The reliability of components used in computer systems has been improving, and more importantly, the number of components used to implement any function has been dramatically decreasing. These factors have produced a corresponding reduction in system failure rates. Given the high reliability of today's machines, it is not practical from a verification standpoint to verify a system by letting it run until failures occur.

Conceptually, faults can be inserted in two ways. First, memory locations and registers can be corrupted to mimic the results of gate-level faults (software fault insertion). Second, gate-level faults may be inserted directly into the hardware (hardware fault insertion). There are advantages to both techniques. One advantage of software-implemented fault insertion is that no embedded hardware support is required.[2] The advantage of hardware fault insertion, on the insertion, a mechanism must either be designed into the system, or an external insertion device must be developed once the hardware is available. Given the physical feature size of the components used today, it is virtually impossible to achieve adequate fault-insertion coverage through an external fault-insertion mechanism.

The error detection and recovery mechanism determines which fault insertion technique is suitable for each component. Some examples illustrate this point. For the lockstep portion of the VAXft 3000 CPUs, software fault insertion is not suitable because the lockstep functionality prevents corruption of memory or registers when faults occur. Therefore, hardware faults cannot be mimicked by modifying memory contents. However, the software fault-insertion technique was suitable to test the I/O adapters since the system handles faults in the adapters by detecting the corruption of data. Hardware fault insertion was not suitable because the I/O adapters were implemented with standard components that did not support hardware fault insertion.

other hand, is that faults are more representative of actual hardware failures and can reveal unanticipated side effects from a gate-level failure. To utilize hardware fault

Because the verification strategy for the 3000 was considered a fundamental part of the system development effort, fault insertion points were built directly into the

system hardware. The amount of logic necessary to implement fault insertion is relatively small.
The goals of the fault-insertion hardware were to

- o Eliminate any corruption of the environment under test that could result from fault insertion. For example, if a certain type of system write operation is required to insert a fault, then every test case will be done on a system that is in a "post fault-insertion" state.
- o Enable the user to distribute faults randomly across the system
- o Allow insertion of faults during system operation
- o Enable testing of transient and solid faults

The fault-insertion points are accessed through a separate serial interface bus that is isolated from the operating hardware. This separate interface ensures that the environment under test is unbiased by fault insertion.

Even with hardware support for fault insertion, only a small number of fault-insertion points can be implemented relative to the total number possible. Where the number of fault-insertion points is small, was considered sufficient for data path coverage. Since a significant portion of the chip area is consumed by data paths, a high level of coverage of each chip was achieved with relatively few fault-insertion points. The remaining fault-insertion points could then be applied to the control logic. Coverage of this logic was important because control logic faults result in error modes that are more unpredictable than data path failures.

The effect that a given fault has on the system depends on the current system operation and when in that operation the fault was inserted. In the 3000, for example, a failure of bit 3 in a data path will have significantly different behavior depending upon whether the data bit was incorrect during the address transmission portion of a cycle or during the succeeding data portion. Therefore, the timing of the fault insertion was pseudo-random. The choice of pseudo-random insertion was based on the fact that the fault-insertion hardware operated asynchronously to the selection of the fault-insertion points is important to achieve a random distribution. Fault-insertion points were designed into most of the custom chips in the VAXft system. When choosing the fault-insertion points, a single bit of a data path

the system under test. This meant that faults could be inserted at any time, without correlation to the activity of the system under test.

Faults may be transient or solid in nature. For design purposes, a solid fault was defined as a failure that will be present on retry of an operation. A transient fault was defined as a fault that will not be present on retry of the operation. Transient faults do not require the removal of the device that experienced the fault; solid faults do require device removal. Since the system reacts differently to transient and hard faults, both types of faults had to be verified in the VAXft system. Therefore, it was required that the fault-insertion hardware be capable of inserting solid or transient faults. Solid faults were inserted by continually applying the fault-insertion signal. Transient faults were inserted by applying the fault-insertion signal only until the machine detected an error.

As noted earlier, the verification strategy utilized both hardware and software fault insertion. The hardware fault-insertion mechanisms allowed faults to be inserted into any system environment, including diagnostics, exercisers, and the VMS operating

Each of the four verification phases built upon the previous phases.

1. Hardware verification under simulation
2. Hardware verification with system exerciser and fault insertion
3. System software verification with fault insertion
4. System application verification with fault insertion

Figure 3 shows the functional layers of the VAXft 3000 system in relation to the verification phases. The numbered brackets to the right of the diagram correlate to the testing coverage of each layer. For example, the system software verification, phase 3, verified the VMS system, Fault-tolerant System Services (FTSS), and the hardware platform.

system. As such, it
was used for initial
verification as well as
regression testing of the
system. The verification
strategy for the 3000
involved a multiphase
effort.

The following sections briefly describe the four phases of the VAXft verification.

Hardware Verification under Simulation

Functional design verification using software simulation is inherently slow in a design as large as the VAXft 3000 system. To use resources most efficiently, a verification effort must incorporate a number of different modeling levels, which means trading off detail to achieve other goals such as speed.[3]

VAXft 3000 simulation occurred at two levels: the module level and the system level. Module-level simulation verified the base functionality of each module. Once this verification was complete, a system-level model was produced to validate the intermodule functionality. The system-level model consisted of a full dual-rail, dual-zone system with an I/O adapter in each zone. At the final stage, full system testing was performed.

Over 500 directed error test cases were developed for gate-level system simulation. For each test, the test environment was set up

The simulation controller provided the following control over the testing:

- Initialization of all memory elements and certain system registers to reduce test time
- Setup of all memory data buffers to be used in testing
- Automated test execution
- Automated checking of test results
- Log of test results

For each test case, the test environment was selected from the following: memory testing, I/O register access, direct memory access (DMA) traffic, and interrupt cycles. In any given test case, any number of the previous tests could be run. These environments could be run with or without faults inserted. In addition, each environment consisted of multiple test cases. In an error handling test case, the proper system environment required for the test was set, and then the fault was inserted into the system. The logic simulator used was designed to verify logic design. When an illegal logic condition was detected, it produced an error response. When a fault insertion resulted in an illegal logic condition, on a fully operational system model and then the fault was inserted. A simulation controller was developed to coordinate the system operations in the simulation environment.

the simulator responded by invalidating the test. Because of this, a great deal of time was spent to ensure that faults were inserted in a way that would not generate illegal conditions. Each test case was considered successful only when the system error registers contained the correct data and the system had the ability to continue operation after the fault.

Hardware Verification with System Exerciser and Fault Insertion

After the prototypes were available, the verification effort shifted from simulation to fault insertion on the hardware. The goal was to insert faults using an exerciser that induced stressful, reproducible hardware activity and that allowed us to analyze and debug the fault easily.

Exerciser test cases were developed to stress the various hardware functions. The tests were designed to create maximum interrupt and data transfer activity between the CPU and the I/O adapters. These functions could be tested individually or simultaneously. The exerciser scheduler provided a degree of randomness such that the interaction of functions was representative of a real operating system. The fault-insertion hardware was used to achieve a random distribution of fault cases across suite of tests worked correctly, fault insertion was performed while the system continually switched between all functions. This testing was more representative of actual faults in customer environments, but was less reproducible.

As previously mentioned, the hardware fault-insertion tool allowed the insertion of both transient and solid failures. The VAXft 3000 hardware recovers from transient failures and utilizes software recovery for hard failures. Since the goal of phase 2 testing was to verify the hardware, the focus was on transient fault insertion. Two criteria for each error case determined the success of the test. First and foremost, the system must continue to run and to produce correct results. Second, the error data that the system captures must be correct based on the fault that was inserted. Correct error data is important because it is used to identify the failing component both for software recovery and for servicing.

Although the simulation environment of phase 1 was substantially slower than phase 2, it provided the system. Because it was possible to insert initial faults while specific functions were performed, a great degree of reproducibility was achieved that aided debug efforts. Once the full

the designers with more information. Therefore when problems were discovered on the prototypes used in phase 2, the failing case was transferred to the simulator for further debug. The hardware verification also validated the models and test procedures used in the simulation environment.

System Software Verification with Fault Insertion

In parallel with hardware verification, the VAXft 3000 system software error handling capabilities were tested. This phase represented the next higher level of testing. The goal was to verify the VAX functionality of the 3000 as well as the software recovery mechanisms.

Digital has produced various test packages to verify VAX functionality. Since the VAXft 3000 system incorporates a VAX chip set used in the VAX 6000 series, it was possible to use several standard test packages that had been used to verify that system.[1]

Fault-tolerant verification, however, was not addressed by any of the existing test packages. Therefore, additional tests were developed by combining the existing functional test suite with the hardware fault-insertion tool and software fault-insertion routines. Test cases used included cache failure, clock failure, memory failure, was running. The completion criteria for tests included the following:

o Detection of the fault

o Isolation of the failed hardware

o Continuation of the test processes without interruption

System Application Verification with Fault Insertion

The goal for the final phase of the VAXft 3000 verification was to run an application with fault insertion and to demonstrate that any system fault recovery action had no effect on the process integrity and data integrity of the application. The application used in the testing was based on the standard DebitCredit banking benchmark and was implemented using the DECintact layered product. The bank has 10 branches, 100 tellers, and 3,600 customer accounts (10 tellers and 360 accounts per branch). Traffic on the system was simulated using terminal emulation process (VAX RTE) scripts representing bank teller activity. The transaction rate was initially 1 transaction per second (TPS) and was varied up interconnect failures, and disk failures. These failures were applied to the system during various system operations. In addition, servicing errors were also tested by removing cables and modules while the system to the maximum TPS rate to stress the system load.

The general test process can be described as follows:

1. Started application execution. The terminal emulation processes emulating the bank tellers were started and continued until the system was operating at the desired TPS rating.
2. Invoked fault-insertion. A fault was selected at random from a table of hardware and software faults. The terminal emulation process submitted stimuli to the application before, during, and after fault insertion.
3. Stopped terminal emulation process. The application was run until a quiescent state was reached.
4. Performed result validation. The process integrity and data integrity of the application was validated.

All of the meaningful events were logged and time-stamped during the experiments. Process integrity was proved by verifying continuity of transaction processing through failures. The time stamps on the transaction

The proof of Data Integrity consisted of using the following consistency rules for transactions:

1. The sum of the account balances is equal to the sum of the teller balances, which is equal to the sum of the branch balances.
2. For each branch, the sum of the teller balances is equal to the branch balance.
3. For each transaction processed a new record must be added to the history file.

Application verification under fault insertion served as the final level of fault-tolerant validation. Whereas the previous phases ensured that the various components required for fault tolerance operated properly, the system application verification demonstrated that these components could operate together to provide a fully fault-tolerant system.

Conclusions

The process of verifying fault tolerance requires a well architected test plan. This plan must be developed early in the design cycle because executions and the system error logs allowed these two independent processes to be correlated.

hardware support for testing may be required. The verification plan must demonstrate cognizance

of the capabilities and limitations at each phase of the development cycle. For example, the speed of simulation prohibits verification of software error recovery in a simulation environment. Also, when a system is implemented with VLSI technology, the ability to physically insert faults into the system by means of an external mechanical mechanism may not be adequate to properly verify the correct system error recovery. These and other issues must be addressed before the chips are fabricated or adequate error recovery verification may not be possible. Inadequate error recovery verification directly increases the risk of real, unrecoverable faults resulting in system outages.

The verification plan for the VAXft 3000 system consisted of the following phases and objectives:

o Hardware simulation with fault insertion verified error detection, hardware recovery, and error data capture.

o System exerciser with fault insertion enhanced the coverage of the hardware simulation effort.

o System software with fault insertion verified software error recovery and reporting.

o System software

The test of any fault tolerant system is to survive a real fault while running a customer application. Although pulling a module out of a machine may seem impressive, machines rarely fail as a result of modules falling out of the backplane. The intitial test effort of the VAXft 3000 showed that the system survived most of the faults introduced. However, problems were found which would have resulted in a system outages if left uncorrected. System enhancements were made both in the area of system recovery actions and the repair call out. While some of the problems were simple coding errors, others were errors in carefully reviewed and documented algorithms. Simply put, the collective wisdom of the designers was not always sufficient to reach the degree of accuracy desired for this fault tolerant system.

As the VAXft product family evolves, performance and functional enhancements will be available. The test processes described in this paper will remain in use, so that every future release of software will be better than the previous version. The combination of verification with fault insertion verified the transparency of the system error recovery to the application running on the system.

hardware and software fault insertion, coupled with physical system disruption allows testing to occur at such a greatly accelerated rate, that all testing performed will be repeated for every new release.

References

1. J. Croll, L. Camilli, and A. Vaccaro, "Test and Qualification of the VAX 6000 Model 400 System," Digital Technical Journal, vol.2, no.2 (Spring 1990): 73-83.

2. J. Barton, E. Czeck, Z. Segall, and D. Siewiorek, "Fault Injection Experiments Using FIAT (Fault Injection-based Automated Testing)" IEEE Transactions on Computers vol. 39, no. 4 (April 1990).

3. R. Calcagni and W. Sherwood, "VAX 6000 Model 400 CPU Chip Set Functional Design Verification, Digital Technical Journal, vol. 2, no. 2 (Spring 1990): 64-72.