VAX 6000 Error Handling: A Pragmatic Approach

By Brian Porter

1 Abstract

The VMS operating system's CPU-dependent support of the VAX 6000 family of computers implements a complex and sophisticated set of error-handling routines. At the start of a VMS session, these routines help construct the necessary framework to support the I/O subsystem as the system begins to emerge. For much of a VMS session, these routines then lay dormant within the SYSLOA image. Periodically, when aroused, they peer into hardware registers looking for signs of trouble. Often, all is well, and the routines return to hibernation. On those occasions when the hardware requires assistance, error handling takes complete control of the system. It has but one mission: identify the error, recover if possible, but at all costs ensure that the integrity of the system remains intact and that data is preserved.

2 Introduction

Error handling is the set of routines that resides in the CPU-dependent loadable image known as SYSLOA. Each processor model that supports the VAX system architecture and VMS operating system has its own SYSLOA image. Error handling is implemented with other common routines like console support and secondary processor start-up. Error handling is unique for each processor model. Individual processor models bring with them a wealth of error detectors and consistency checkers. Each device has to be independently interrogated and reset once triggered.

Error handling of one form or another resides throughout the VMS operating system. In some contexts, trying to edit a file in a directory structure that does not exist can be considered an error. This paper discusses only errors that deal with the underlying CPU and memory hardware on which the VMS system is running. It describes the development of error handling to support the CPU modules and memory controllers that make up the system kernel in the VAX 6000 series. This paper explains our error-handling strategy to not only reduce the amount of unique coding, but also provide an opportunity to enhance, mature, and improve existing VAX 6000 products.

3 Development of Error-handling Routines for the VAX 6000 Platform

The VAX 6000 platform provided a unique opportunity to develop error-handling routines. As shown in Figure 1, the XMI backbone of the system allows the creation of increasingly powerful systems that retain much of their operating characteristics. Increases in processor capability are gained by merely exchanging processor modules for more powerful models. We decided that error handling should not be any different. On prior systems,

a complete set of error-handling routines for each CPU model had to be implemented. We adopted an approach to error handling that could be carried

forward from one processor to the next with little or no change to the initial error-handling model. This approach handles identical errors in the same way with the same code base.

The protocol of the XMI bus was modified to allow support of write-back caching schemes of the VAX 6000 Model 500 and VAX 6000 Model 600. However, this had no ill effect on the overall error-handling model we decided to use in the support of the VAX 6000 family of processors.

VAX 6000 Family Error Delivery

Identical mechanisms were used to structure error delivery on each processor in the VAX 6000 family. Each processor has two system control block (SCB) interrupt vectors and a single SCB exception vector. The interrupt vectors deliver hard and soft errors. The exception vector delivers machine check exceptions.

Hard Error Interrupts. Hard errors can be categorized in the following way. Hard errors occur as conditions that are not synchronous to the program counter (PC). In almost all instances, systems cannot recover from hard errors. They indicate that data or machine state has been lost. Hard errors are normally fatal. Hard errors are delivered through SCB vector 60 (hexadecimal); interrupt priority level (IPL) is raised to 29 decimal.

Soft Error Interrupts. Soft errors, on the other hand, generally signal an asynchronous condition, with respect to the PC, that has been corrected by hardware, or that can be overcome with some software intervention. Soft errors are normally always benign to system operation. Soft errors are delivered through SCB vector 54 (hexadecimal); IPL is raised to 26 decimal.

Machine Check Exceptions. Machine check exceptions are internal processor conditions that are synchronous to the PC. If the condition can be corrected when the instruction that caused the exception is reexecuted, the result is the same as if the condition had not occurred. Many of the machine check exceptions that are reported by the VAX 6000 family of processors allow recovery so that normal operation can continue. Machine check exceptions are delivered through SCB vector 4; IPL is raised to 31 decimal.

4 Objectives

Error handling must identify the error and recover if possible. Above all, it must guarantee the integrity of the system and the preservation of data.

An important project goal was to produce a robust and quality product that would have predictable performance. We chose to have a single error-handling model that could be implemented for all VAX 6000 CPU

models. We also adopted an implementation methodology that included the capability to allow rigorous testing of the many code paths contained in the various configurations. To accomplish this goal, we designed the test and verification strategy in conjunction with the overall system design of the kernel error-handling subsystem. In addition, we designed and

implemented an object-oriented code base for errors that are common across the platform. Errors are handled in this way when they are associated with main memory, with XMI bus protocols, or with the support of vector processors.

Most frequently occurring errors are associated with main memory. The error handling for main memory is composed of three major functions. The first handles the complexity of support for two different memory controller types and their internal error conditions. The other two functions are logically split between single-bit error correction code (ECC) failures and double-bit ECC failures.

Common error-handling interfaces and routines were established for the VAX 6000 family of processors. The use of common files and interfaces ensures that errors are handled in exactly the same way for each CPU model.

5 Full Support of the Symmetric Multiprocessing Paradigm

The VAX 6000 family of CPUs are symmetric multiprocessing (SMP) systems. The error-handling model assumes that more than one CPU is always active. The synchronization of error handling throughout the system has numerous benefits. If an error condition were detected throughout the system, it would be a very complicated procedure to ensure that all CPUs reacted consistently. Such errors would clutter the error log with reports from every CPU and XMI device.

Error Logging Synchronization

In the VAX 6000 scheme, error logging is synchronized across the system. If an error affects all nodes, this information is included with the first CPU to respond to the error. Machine state is created that informs other CPU nodes that the event has been logged on their behalf. As each CPU node responds to the error condition, it can interrogate this state. In the event that all error conditions have been logged on behalf of a CPU, the error condition is cleared and the interrupt or exception is dismissed. The one entry in the error log for these types of errors clearly indicates that other nodes were active. Information about the nodes affected and state indicating how the node was affected is recorded in the single error log entry.

CPU Configuration Data in the Error Log

A CPU running with some of its hardware disabled may have operating characteristics that cause other CPUs to incur error conditions of some type. An error log entry from a VAX 6000 CPU always includes the configuration of other active CPUs on the system. For example, if the CPU at node 6 is running with its backup cache disabled, other CPUs

include this information with their error log data. Thus, potential error conditions can be easily identified.

6 Error Log Filtering

Some errors that occur at too high a rate are filtered from the error log. Errors that are delivered by the soft error vector are invariably benign to system operation. It is important that they be reported because they can indicate an impending fatal error in some subsystem. However, if these errors are occurring too often, only a subset is sent to the error log. The algorithm is based on an error count over time. If an error is occurring too rapidly, logging of the errors is inhibited. At a later time, logging is reenabled. Errors that do not appear in the error log are still counted, and the accumulated totals are displayed by other error conditions that are sent to the error log.

7 Message Facility

Error handling on the VAX 6000 has the unique ability to output formatted messages. Integral to the error-handling subsystem is a message processing facility that is composed of specialized routines and modified versions of several VMS system services. The modified system services include SYSFAO and SYSCVRTIM. The message facility provides the error-handling subsystem with the capability to output formatted messages that contain both text and data. These messages are time-stamped and sent to the system console device OPA0:.

Messages can be output in two different modes. Interrupt driven mode is the most common and uses the standard terminal driver functions of the running VMS session. Messages that use this mode describe the disabling of some part of the CPU kernel at system start-up or during the current session. The other mode of output is synchronous and is in line with error processing. This mode is reserved for hardware errors that are nonrecoverable and result in a system crash. The message is output just prior to calling the BUGCHECK mechanism that would terminate the current VMS session abnormally. Messages are always descriptive of the error or exception condition and contain all the machine state available at the time of the error.

Formatted messages allow for errors that occur as the system is being initialized to be reported and described should the system fail to boot. The output of messages is fully synchronized between the primary and secondary CPUs of SMP systems. The primary CPU outputs messages about errors occurring on secondary processors.

8 Error Rate Checking and Loop Detection

The VAX 6000 family of CPUs provides a great deal of error detection. The error conditions signaled in many cases are benign to the system if the appropriate action is taken. However, blind recovery from errors can be a

downfall in itself. It is not uncommon for so many benign failures to occur that error handling is the only task being performed by the system. Error

handling on the VAX 6000 family implements a system of rate checking and loop detection to combat this problem.

Rate and Loop Detection Time Base

The timing standard used by the rate checking and loop detection subsystems is the CPU TODR register. The TODR hardware register is independent of software and increments every 10 milliseconds.

Rate Checking of Errors

Each error condition has an associated rate check database. The database tracks TODR values for the three most recent errors. If these errors occur too fast, special action is taken in addition to that required to service the error. This may involve disabling the signaling of the error condition itself. For example, some errors that are reported outside the CPU can be turned off. When sufficient data about an error has been collected, the error may be disabled for a period of time. Hardware features such as internal cache errors can also be disabled. If cache errors occur that are recoverable, but are occurring too fast, the cache is disabled. The occurrence of multiple errors can indicate a broken structure, whereas a single error can indicate a single transient event.

Loop Detection

Multiple errors of different types can also occur frequently. In this situation, the system is operational, but it is continuously at high IPL, servicing error interrupts or exceptions. This operating scenario is detected by the frequency of transitions in and out of error handling. When error-handling code threads are entered and exited, the TODR value is saved. During execution of error handling, the enter TODR value is compared to the last exit TODR value. If the result is too close, a count is incremented. If the close relationship of exit to entry continues to occur, a loop condition is declared and appropriate action is taken. Most often this means the system is shut down.

9 Error-handling Model

The traditional approach to error handling in the VMS operating system has been to interrogate registers and act on the data directly in real time. Another approach has been to save only a subset of processor state that has a linkage to the error delivery vector and then act on this data during a later parse operation.

When designing the error-handling model for the VAX 6000 series, we decided to save all CPU state that is visible to macro programmers in buffers specific to each CPU. All interrogations are then made on the data in this

buffer. Information on the hardware state is saved as well as the current system time. Any action taken by error handling is also recorded in the buffer. This approach has several advantages. First, a distinct footprint of the last error is contained in the system image in memory. Should the system fail, the data is saved when a crash dump is taken. Second, the many

execution thread possibilities are made easier to test and verify. Finally, conditions are easier to diagnose if the original data that error handling processed and the actions that were taken are recorded in an error log.

The error-handling process for the VAX 6000 series consists of six distinct steps:

- Setup and synchronization
- Saving of state
- Parsing of data
- Processing and accounting of state
- Error logging
- Error reset and dismissal

This logical organization provides flexibility to the implementation being addressed at the time. The parsing of data step was added at the same time that support of the VAX 6000 Model 400 was implemented.

Setup and Synchronization

Synchronization is accomplished by acquiring the MCHECK spinlock. The use of spinlocks is a VMS technique that provides atomic access to code threads and data structures and ensures that only one CPU at a time is in error handling. Thus it is possible to compress an error condition occurring throughout the system into a single error log entry by the first CPU to service the error.

Following synchronization, the SMP sanity and spinlock acquisition timers are disabled. If an error occurs at the boundary of one of these timers, a false termination of the session can occur due to the time consumed by the execution of error handling. The SMP sanity and spinwait timers are mechanisms used in VMS to ensure that CPUs active in a multiprocessor system are interactive with each other and the synchronization primitives that control access to various resources. The sanity timer is used as a watchdog timer to ensure that CPUs respond to hardware clock interrupts on a regular basis. Each CPU active on the system monitors another CPU for its response to hardware clock interrupts. The spinwait timer guarantees that one CPU does not retain ownership of a spinlock resource for more than an allotted time period. Error handling is always executed at an IPL above which hardware clock interrupts can be serviced. As a result, it defeats the sanity timer mechanism. Some of the actions taken by error handling can cause a spinwait timer to expire if the error being serviced occurs too

close to the timer boundary. By disabling these SMP timers, a time period is started over when the error being serviced is dismissed and timers are reenabled.

The buffer associated with the CPU experiencing the error is initialized to zero and is ready to receive the latest error state. If the error is a machine check, stack space is allocated and initialized to allow for error dismissal and error-handling exit.

When machine checks or soft error interrupts are serviced, the cache subsystem is unconditionally disabled. Error handling does this to preserve any error state that may be in the cache. If the error happens to be cache-related, the state can be extracted at the appropriate time. Cache-related errors are not reported by hard error interrupts on some VAX 6000 models. When hard error interrupts are serviced on these processor models, the cache is not disabled by software.

The machine check flow has an additional check to determine if the error is associated with a recovery block. Recovery blocks on the VMS system provide kernel-mode macro programmers with the ability to protect an execution thread from the effects of fatal machine check exceptions. Normally when a machine check occurs in kernel mode, the code thread being executed loses control. Unless the instruction can be restarted, the VMS session is terminated. The placement of kernel-mode execution threads within the context of a recovery block ensures that a machine check will cause control to be passed to the end of the recovery block, along with status to indicate that the machine check has happened. The macro programmer can select what type of machine check to be protected from. In general, this is limited to those machine checks caused by references to the physical address space that do not respond or return data.

If it is determined that the current delivered machine check is protected by a recovery block and that error handling established this condition, the error is dismissed immediately without further action. More details are given in the following section.

Saving of State

All available CPU error state is saved regardless of error type or delivery mechanism. Machine checks also save the internal state passed by microcode on the stack. Each register is read into its local storage buffer within the context of a recovery block. A valid bit is associated with each local copy register cell according to its status as it exits the recovery block context. This is important because each cell has been initialized before use. A register value of zero may be significant, and a failed register read would allow the initialized value to be interpreted as not having any error or state bits set. Failed register read indications can help in the diagnosis of the original error condition.

The recovery blocks used when error state is collected have special flags to indicate that they were established by error handling. If an error

does occur, control is returned to the appropriate point in the error-handling execution thread. The original saved state of the first error is not overwritten.

Parsing of Data

The parsing of data step was added to support the VAX 6000 Model 400 and later models. The data collected in the saving of state routines is parsed as a separate step. When error data is parsed and processed in a single step, as in the VAX 6000 Model 200 and VAX 6000 Model 300, it is difficult to make the necessary errors invisible to the error log. When the error data is parsed and an error mask is produced that represents the error conditions present, it is much easier to detect if all current conditions have been serviced. Since it is also easier to detect conditions with only one error present, expected error conditions can be processed. The VAX 6000 Model 400 and later models have many benign error syndromes that have their logging filtered.

Processing and Accounting

During the processing and accounting step of error handling, the data in the CPU private local buffer is parsed and acted upon. These routines detect if a specific error condition is global and sensed throughout the system or local to the particular CPU. Global errors include the state from other CPUs and devices in the error log if required. If the global error is the only one present and it is expected, machine state is set to indicate that error logging should not occur. Should this CPU be the first to process the global error, it samples data in registers of the other CPUs and devices and leaves state to indicate the error condition has been serviced and is expected. Consequently, the context of global errors is included into a single entry in the error log. XMI parity errors are serviced in this way. Local errors record only the state from the CPU experiencing the error in the error log.

Error handling supports the notion of expected errors, or errors that sometimes occur as a result of operations performed by error handling. These errors are not reported to the error log. For example, duplicate tag parity errors can sometimes occur when the backup cache on the VAX 6000 Model 200 and VAX 6000 Model 300 is invalidated. To cause these errors to be invisible to the error log, a mask of error bits to ignore is set up when backup cache invalidate operations are executed. At the same time, a fork process thread that ultimately clears the appropriate mask bit is queued. If an error that would be invisible to the error log occurs, the "ignore-this-error" bit is sampled in the recovery thread for the error condition. If the bit is set, the error is ignored. If the error does not occur, eventually IPL is lowered until the queued fork process runs and clears the bit in the mask. This guarantees that later real errors that have previously been expected and have not occurred are not excluded from the error log.

Error Logging

The data collected by the error-handling routines is sent to the VMS system's error log after it is tallied for size. The amount of record space is allocated by internal VMS routines. The raw data that describes the context of the current error is copied to the VMS error log buffer along with the current values of accounting data for the CPU. The accounting data is a count of the individual error conditions that have occurred on this CPU for the current session. Any CPU that has some part of its functionality disabled includes that data in the error log as well. For example, a CPU that is executing with a disabled cache may cause errors to occur on other CPUs. It is useful to know that a CPU is running in a degraded mode when investigating problems that are occurring on a system. The error log records of all CPUs clearly indicate any CPUs operating at reduced capacity. If all CPUs are running unimpeded, the error log contains a flag to indicate this status.

The amount of data included in the error log for any given error can be different. The data describing the CPU context is the same except in the case of machine checks. These errors also include internal state passed by the microcode through the stack. Depending on the error condition, context from the XMI bus, the memory subsystem, or an external XMI adapter can be included. The error data is organized into various subpackets that are signaled to be present by a flags field contained in a header section of the CPU context packet. For example, an error can occur that describes a failure of a transaction between the CPU and memory. If the data collected from the memory subsystem during the processing and accounting step indicates an error is present, this is included in the error log record. If there is no indication of a memory subsystem error, a flag to indicate that no memory errors are present is set. This reduces the burden on the error log buffers of the VMS system and reduces the clutter and confusion of error registers from a device that does not have an error condition present.

Any error that ultimately causes the system to crash is also logged to the system console terminal through the SYSLOA message facility. Errors can occur during start-up before VMS error logging is available. Errors can also occur and terminate the session before the system completes initialization. For these reasons, fatal errors are always output to the console terminal before the session is terminated. Errors that occur at start-up of secondary processors are monitored by the primary processor. Any output required is done by the primary processor.

Error Reset and Dismissal

The last step of error handling resets error conditions that have been serviced and dismisses the interrupt or exception. The image of the

data saved is used as a mask to reset error conditions. This technique guarantees that double error conditions are not lost.

Registers that require initialization are reset using the contents that were read when the error was first serviced. Most error conditions are write-one-clear. That is, to reset the error condition, a mask of the error conditions set has to be written to the appropriate register to clear the error. The use of the original contents of the register as a mask guarantees that an error condition occurring during the processing of an error cannot be lost. Reset of the VAX 6000 Model 200 and VAX 6000 Model 300 error registers includes a later probe of the register for the absence of error indications. Should an error still be present, the error-handling process is restarted and it treats the condition as a new error. After errors are reset, the cache subsystem is invalidated and a check is made to determine if it should be reenabled. Processing of the error could determine that the cache or indeed the CPU should be taken off-line because of an error rate or finite count that is too high. If all is well, the cache subsystem is reenabled. The MCHECK spinlock is now released and the interrupt or exception is dismissed by executing a return from exception or interrupt (REI) instruction.

If the error being dismissed is a machine check, the additional storage allocated by error handling and error state left by the microcode has to be removed. As shown in Figure 2, the additional storage is an array of quadwords. These quadwords represent program counter/processor status longword (PC/PSL) pairs that direct control to routines that must be executed prior to control being returned to the exception PC. The additional postprocessing that takes place for noncorrectable memory errors and errors that cause a process to be aborted are dispatched using this mechanism.

Machine check processing takes place at IPL 31. Fatal memory error recovery uses the VMS system's page fault code threads. These threads use spinlocks that cannot be acquired from IPL 31. When dismissing machine checks, the PC /PSL pairs are interrogated to determine if they are nonzero. If the probed quadword is zero, the stack pointer is updated to unwind by the quadword allocated. This continues until all three array elements have been probed. If the array element is nonzero, the REI passes control to the PC and PSL described by that array element. Synchronization is thus preserved, and spinlock acquisition rules are obeyed. Eventually the array is traversed, and each element is removed. State left by the microcode is removed, control is passed back to the original exception PC, and instruction retry is attempted. If error handling determines that the execution thread should be aborted, the original exception PC is replaced by a PC/PSL pair that returns control to VMS exception routines. From these routines, control is normally passed to an appropriate mode condition handler. If a condition handler has not been established, the VMS process is aborted. Kernel-mode threads that experience fatal machine checks always result in the termination of the VMS session.

10 Support of the VAX 6000 Model 200 and VAX 6000 Model 300

The error-handling support for the VAX 6000 Model 200 and VAX 6000 Model 300 is identical. These two processor models are the same logical CPU. The VAX 6000 Model 300 is a selected faster component set of the VAX 6000 Model 200.

The VAX 6000 Model 200 system presented a unique problem for error handling because the primary cache is internal to the CPU chip. Errors from the primary cache do not cause an interrupt or exception. These errors can never cause a failure or wrong result should they occur. Because all cache structures on the VAX 6000 Model 200 are write-through, data can be both in cache and in memory, and it is always consistent. If a parity error occurs on either the data or tag section of the primary cache, microcode can always fetch another copy of the data from memory. If a primary cache tag or data error occurs, microcode sets a status bit to indicate the error in an internal processor register. The internal processor register is private to the local CPU. Previous CPUs with this type of error signaling used a polling technique to detect these failures. On SMP systems, only the primary CPU is interrupted on a regular basis to allow polling routines to run.

Since we had no precedent for reference, we designed a system whereby the primary CPU uses the interprocessor interrupt mechanism to interrupt secondary processors. When the secondary CPU receives the CPU-specific interprocessor interrupt, it reads the appropriate internal processor register, places the data in a known location, and sets an indicator flag. On later poll cycles, the primary CPU sees the indication from the secondary CPU and interrogates the known location for any error bits. If no errors are detected, each secondary CPU is polled once every ten seconds. Should an error be found, the secondary CPU with the error has its polling frequency increased to once every second. If ten successive polls indicate error conditions, the secondary CPU is signaled to disable its primary cache. If this occurs, entries are made in the error log and to the system console by the primary CPU on behalf of the secondary CPU.

During systems integration of the VAX 6000 Model 200, certain random-access memory (RAM) devices used for the backup cache exhibited excessive parity error failures. The problem was so severe that special error-handling software and additional CPU hardware functionality were developed to isolate and diagnose the failures. This work was so successful that the hardware functionality was made a permanent feature of the processor, and the error-handling routines were made a permanent part of the SYSLOA image. The hardware functionality and software routines allowed for the failing data bit in the backup cache to be identified at the time of the failure. The VAX 6000 Model 200 experience had a lasting effect on error handling across the VAX 6000 family. The ability to diagnose cache parity errors

to the bit level in the operating system remains a characteristic of error handling on all VAX 6000 systems.

11 Support of the VAX 6000 Model 400 and VAX 6000 Model 500

Although the CPU chips on the VAX 6000 Model 400 and VAX 6000 Model 500 are the same, the SYSLOA images are not. The major difference between the two systems is the write-back cache subsystem implemented by the VAX 6000 Model 500. To facilitate write-back cache strategies, the XMI bus was enhanced to support a directory-based broadcast coherence protocol.[1]

The VAX 6000 Model 400 and VAX 6000 Model 500 systems represented a dramatic increase in system complexity for error handling. The amount of error detection incorporated within each increased and became more complex. The overall model implemented on the VAX 6000 Model 200 was maintained, but a step was added between the steps of saving of state and processing and accounting. The new step, parsing of data, was previously a part of processing and accounting. Error handling support of the on-board CPU electrically erasable programmable read-only memory (EEPROM) was also added. The EEPROM was until now used only by CPU console support.

The overall model now became

- Setup and synchronization
- Saving of state
- Parsing of data
- Processing and accounting
- Error logging
- Error reset and dismissal

Storage space for machine check and hard error is shared in the VAX 6000 Model 200 system. However, this support became too complicated to manage. In the VAX 6000 Model 400 and later models, both sets of error state are available in crash dumps.

EEPROM Support

The experience gained from systems integration of the VAX 6000 Model 200 showed that real-time diagnosis by the operating system has many benefits. However, the scheme used by the VAX 6000 Model 200 was cumbersome and recorded only the resulting diagnostic data to the error log. The challenge was twofold: to make the mechanics of cache parity error diagnosis easier, and to make the data more widely available. We achieved both goals by using the EEPROM on the CPU module. The VAX 6000 Model 500 made additional improvements by using both on-board and high-speed RAM and EEPROM.

EEPROM and RAM structures exist within the physical address space of the VAX 6000 family. These structures are primarily used by the console for cross-session storage of data. High-speed RAM is used for general heap storage by the console. RAM and EEPROM structures have physical addresses that are in the I/O region of the address space. Address references to I/O address space do not cause cache lookups. The code threads that perform data extraction were placed in the EEPROM and RAM structures to avoid

special hardware operating modes. A few simple routines enabled easier diagnosis of cache parity error failures and a method of disabling the cache that does not disrupt error state. The error-handling SCB vectors were pointed to the EEPROM so the routines that disable the cache could do so without making cache references. (On the VAX 6000 Model 500, the cache disabling routines are placed in the high-speed RAM.)

When an error occurs, control is first passed to individual routines that reside in I/O space. These routines disable the cache subsystem and then return control to the SYSLOA image in main memory.

The VAX 6000 Model 500 has an error transition mode (ETM), which allows the backup cache to be partially disabled. New blocks are not allocated when in ETM mode. Data requests are filled from the cache. Error interrupts or exceptions on the VAX 6000 Model 500 dispatch to routines that execute from I/O space and place the write-back backup cache into ETM and disable the write-through primary cache.

The EEPROM on both the VAX 6000 Model 400 and VAX 6000 Model 500 is also used to store failure information. When errors occur, a counter that is associated with the specific error condition is incremented. The number of error conditions is finite and fully described by the error mask produced by the parsing of data routines. Writing to the CPU EEPROM is time-consuming compared to writing to main memory. A byte write to EEPROM takes on the order of 15 milliseconds. To avoid this overhead, the EEPROM VMS data actually resides in main memory during a VMS session. As each CPU is initialized by the VMS system, the contents of the VMS area are read into individual CPU memory regions. Updates that are required are made to these regions. When CPUs are stopped or when the system is shut down or has crashed, the region of memory associated with a particular CPU is written back to that CPU's EEPROM. In addition to error information, a count of seconds run in a VMS environment is tallied.

Vector Processor Support

One set of routines supports the VAX 6000 Model 400 and VAX 6000 Model 500 vector processors. These routines are organized in an identical manner to the CPU routines and follow the same steps related to CPU error conditions. During the processing and accounting of CPU error conditions, a check determines if any vector processor errors are present. If vector errors are detected, the appropriate support routines-soft error, hard error, or machine check-are invoked.

Error handling supports a maximum of four vector processors. If the number of errors or the rate of errors becomes too great, vector processors are removed from use. Error handling never removes the only or last remaining vector processor. Support of vector processor errors has the

same characteristics as support for CPU-related error conditions. Portions of the vector processor are disabled if the associated error rate becomes too great. If other errors continue, the unit is removed from use. The notions of expected errors and errors that are invisible to the error log also exist.

12 Support of the VAX 6000 Model 600

Error checking and detection on the VAX 6000 Model 600 are very complex processes. There are well over 160 unique soft and hard error conditions as categorized by the software. The actual count declared by the hardware is much greater. The disparity results from the way software groups error conditions. The VAX 6000 Model 600 error handling followed the enhanced model implemented on the VAX 6000 Model 400. The state saved for interrogation by VAX 6000 Model 600 error handling consists of 40 internal and XMI-addressable registers. Support of the VAX 6000 Model 600 also included support of the on-board CPU EEPROM for the long-term storage of failure information. The support of the EEPROM was extended to include the history of the cache subsystem performance in previous sessions.

Like the VAX 6000 Model 500 system, the VAX 6000 Model 600 implements a write-back backup cache strategy. The VAX 6000 Model 600 backup cache operates using a directory-based broadcast coherence protocol.[1] Each 32-byte cache block is in one of three states: invalid, valid/read-only, or valid/written. Multiple caches may hold read-only data simultaneously; written data may be held by only one cache in the system at a time. Write privilege for a block must be obtained before modifying the data in that block.

Certain backup cache error conditions are severe enough to disable the cache. The backup cache may contain written data that is unavailable elsewhere in the system. To access that data, the backup cache is put into ETM, a state which allows written data to be accessed by the cache controller, but disallows the use of read-only data.

A cache enters ETM as a function of either software or hardware. The cache is put into ETM by hardware only when cache data may have been corrupted, or when cache data may be inconsistent with data in memory. Thus, correctable backup cache errors do not cause a transition into ETM, but uncorrectable errors do. A parity error on the NVAX data and address lines (NDAL) interface causes the cache to enter ETM because an invalidate or write-back request may have been missed. A cache transition into ETM occurs when a request for write privilege or a write back does not complete successfully on the VAX 6000 Model 600. The state of the cache is likely to be inconsistent with that of memory.

Three requirements govern cache operation during ETM: (1) The state of the cache is preserved as much as possible to allow software to diagnose the problem. (2) Memory references that hit written blocks in the cache are processed, since this is the only source of data in the system. (3) Cache coherency requests from the NDAL are processed normally so that cache state remains consistent with memory.

Although complex, ETM allows the software to choose when and how to disable the cache. To make the process of error handling less cumbersome, the backup cache is unconditionally put into ETM by the software when any error condition is being serviced.

ECC protects both tag and data stores on the backup cache on the VAX 6000 Model 600. Correctable ECC errors in the backup cache have a record of failed syndromes kept by error-handling routines. Should the same syndrome fail on more than one occasion in a single VMS session, the backup cache is disabled.

If uncorrectable backup cache errors occur, the error-handling routines determine if the block is owned by the CPU and attempt to flush the block back to memory. If successful or if the block is not owned, the backup cache is disabled before returning the system to normal operation. If the data cannot be recovered, the VMS session is terminated.

If the backup cache is disabled by error handling for any reason, that fact is recorded in the CPU EEPROM. Records on disabled status are also kept for the primary cache and virtual instruction cache (VIC). Subsequent sessions of VMS interrogate the EEPROM and cause these structures to remain disabled if they were disabled in a previous session. When this occurs, an appropriate message is sent to the console terminal during system start-up.

Tag parity errors that occur in the VIC are diagnosed in an unusual manner. Unlike other caches on the VAX 6000 Model 600, the tag store of the VIC contains a virtual address. To determine which bit has failed when a parity error occurs, the tag store is probed to retrieve the contents of the failing tag location. The associated data store location is probed to retrieve its contents; each bit in the bad tag address is then flipped in turn. As each bit is flipped, the range of the resultant virtual address is compared to page tables to determine its validity within current context. The virtual address is translated, and the resulting physical address is mapped to allow error handling to read the contents of the page. The appropriate contents of the newly mapped page are compared to the contents read from the VIC data store. If one and only one match is found, the failing tag bit is identified. Masks of failing bits from all VAX 6000 Model 600 cache structures are stored in the CPU EEPROM along with other failure information.

The instruction pipeline complicates VAX 6000 Model 600 error handling. In many instances, errors experienced are in no way related to the current instruction being executed or interrupted. When an error does occur, care must be taken to fully understand in what context the error has an effect.

13 Correctable Memory Errors

Correctable memory errors are data errors that are corrected by the memory controller before data is returned to the requester. They occur primarily because of alpha particle radiation, affect only a single cell, and are transient in nature. Correctable memory errors are completely benign. To determine if a memory controller reporting correctable errors has real

defects, multiple errors must be viewed.

VAX 6000 Error Handling: A Pragmatic Approach

VAX 6000 error handling implements a scheme whereby error data reported by memory controllers for correctable errors is compressed into a structure called a footprint. The footprint reduces the data reported into a form that uniquely describes the error that just occurred. The intent of the footprint is to uniquely index the source component of memory, the dynamic RAM (DRAM). Hence, for a given memory subsystem, the number of valid footprints would equal the number of DRAMs. Furthermore, the footprint block maintained per footprint is used to track the context (repeat errors, scrubbing, etc.) of this error as well as other errors that match this footprint ID.

The assumption here is that most correctable memory errors are a result of DRAM component faults, hence the granularity of the unique DRAM. As shown in Figure 3, the footprint forms a 32-bit integer from the XMI node ID, the ECC syndrome of the error, and the memory controller bank in error. The integer is used to locate other correctable errors that have occurred in an internal database. Along with the footprint, the address of the correctable error is passed to a set of routines that performs all processing of correctable memory errors. The database tracks the range of addresses that have experienced correctable errors for the same footprint. This aids in the diagnosis of row and column failures with the DRAMs that make up memory controller storage. On the VAX 6000 Model 500 and VAX 6000 Model 600 systems, memory scrubbing status is also tracked.

Memory scrubbing is a technique for reducing the number of error interrupts from locations that are reporting errors caused by alpha particle disturbance. Scrubbing removes transient faults from the system, which in turn reduces the number of interrupts that result from such errors. In addition, it helps to differentiate transient errors from permanent DRAM component faults, as captured in the error log. This information was previously unavailable.

When the VAX 6000 memory controller detects a correctable memory error, circuitry in the controller corrects the data returned for the request. The data is not corrected in the storage DRAMs on the controller. If the location is read over and over again, the same error and correction cycle occurs each time. This continues until the location is updated with write data. An interrupt can be generated for each error correction cycle. Care must be taken when scrubbing memory locations. The data in any given memory address can be shared by any number of CPUs or I/O devices. When this is the case, a higher-level software protocol is normally used to synchronize access. Error handling would not be privy to these protocols. VAX 6000 Model 500 and VAX 6000 Model 600 memory scrubbing is possible because of the XMI2 bus protocol. Before a CPU can modify any location in memory, it must be the exclusive owner of the 32-byte block in which the address resides. Ownership is effected at the primitive hardware level and so exclusive access is guaranteed.

When a correctable error interrupt occurs on a VAX 6000 Model 500 or VAX 6000 Model 600 system, error handling rewrites the failing location with its contents. The ability to cause an interrupt is disabled in memory controllers that continue to report errors with the same footprint or that have not responded to scrubbing. This action occurs after sufficient data has indicated that something other than alpha particle disturbance has occurred and the memory controller may require service.

The rate of correctable memory error interrupts is checked to reduce the burden on the system. If the rate of errors occurring becomes too high, the ability to interrupt is disabled at the problem controller for a period of time. Correctable memory error data collected during a VMS session is sent to the error log at the end of the VMS session.

14 Uncorrectable Memory Errors

Uncorrectable memory errors experienced by the CPU are reported as machine checks. These machine checks are synchronous with the PC making the reference. Uncorrectable memory errors occur when data is lost by the memory controller and cannot be re-created by its ECC circuitry; fortunately, these errors seldom occur. Uncorrectable memory errors represent a serious problem to the execution thread that experiences them. The hardware cannot assist in the recovery of this type of error; recovery is totally a software function.

If the page that experiences an uncorrectable error is a process private page that has not been modified, and the code thread currently executing is at pageable priority, the error is not considered fatal. The error-handling routines arrange for the page to be re-created in a different physical page in memory by invalidating the necessary memory management structures. As a result, a translation-not-valid exception occurs when the instruction that experienced the exception is retried. The page fault mechanisms of the VMS system do the actual re-creation. The original page with the error is put on a list of bad pages internal to the VMS system. If the page does not meet the criteria for replacement, either the process is deleted or the VMS session is terminated. If the process is deleted, the page is marked "bad" by error handling, and the process run-down routines in VMS retire the page to the bad page list.

15 Testing

Early in the project, we decided the ability to test and verify had to be built into error handling to produce a predictable, robust, and quality product. Although the VAX 6000 family and CPUs in general have a number of features that allow errors to be generated, they tend not to be general-purpose. In most cases, they are designed for use by special diagnostic software that does not operate in the context of an operating system, e.g.,

the VMS operating system. We chose to implement a scheme whereby errors would be simulated in software on the target hardware. This approach gave us several clear advantages. The most important was that the approach could

be extended as the power and complexity of CPU models increased and that complete control was with the designers. No special hardware equipment or CPU feature would be required. The only precondition was that certain software implementation guidelines had to be followed to make use of the simulator.

Machine check test (MTEST) consists of two parts, a utility and an error-handling implementation methodology. The methodology consists of using main memory storage as the primary agent that is acted upon by error handling. This method also fit into our model of retaining data in memory. The other requirement was the strategic placement of the DEBUG_TRANSFER macro. DEBUG_TRANSFER expands to produce a code segment that determines if the current error being serviced is an error simulation or not. If it is, data that resides in memory that is being interrogated is modified, in concert with MTEST, to reflect the error condition being simulated. DEBUG_TRANSFER code segments represent synchronization points between an error-handling execution thread and the MTEST simulator.

The MTEST simulator is a privileged image and consists of a user interface, a number of nonpageable internal buffers, and simulator routines. The user interface allows the internal buffers to be selected and loaded with data patterns of the user's choice. The user interface also allows the user to pass control to the SCB vectors of the VMS system. In our case we used the vectors that are the linkage to error-handling routines. Once in control, error handling would execute its model until it reached a DEBUG_TRANSFER code segment. The segment would determine that this was an error being simulated and return control to MTEST. MTEST would then decide if the synchronization point was one for which the user has data. The data would be transferred from the buffer named in the DEBUG_TRANSFER code segment to the address also declared in the segment. By judiciously placing the DEBUG_TRANSFER synchronization points and carefully selecting an appropriate data pattern, we were able to simulate any and all error conditions for the appropriate CPU.

In this way, we were able to verify many complex algorithms and code paths that would have been difficult to exercise. We were also able to verify error handling and error logging from the point of error to the error log file. MTEST can be either interactive or procedure-driven. This aspect allowed us to maintain a library of procedures that could be used at any time to verify that operational characteristics for individual errors had not changed when code paths that affected many error types were modified.

MTEST was the primary tool we used for testing. During the test and verification phase, prototype hardware that had real error conditions became available, and we used these prototypes.

16 Conclusions

The VAX 6000 family now has a robust and complete set of error-handling routines that accomplished our project goals. In fact, many routines were never before part of the VMS system. These routines include the ability to report complete error context to the system console and the ability to group failures occurring across the system to a single error log entry. An important SMP feature is the ability to recognize and retire failing processors from the active set of a VMS session and allow the session to continue. These routines and others support the entire range of VAX 6000 CPU models. The object-oriented approach to error conditions not on the CPU module has made support and introduction of newer routines easier. The ability to test at will any or all error-handling routines has been a tremendous advantage.

17 Acknowledgements

Our success resulted from a number of factors, including the advantages of designing the ability to test into the product. There is no substitution for actually executing a code thread to determine the effectiveness of its design goal. The various engineering groups involved in designing the many 6000 CPUs showed great discipline in producing engineering specifications that met the needs of both hardware and software engineering groups. The many hours spent painstakingly describing intricate details of error conditions and the production of parse trees allowed the structured approach we set out to achieve. Special thanks to Mike Uhler for his parse trees and to Nick Carr, who suggested this paper be written.

18 Reference

1. G. Uhler et al., "The NVAX and NVAX+ High-performance VAX Microprocessors," Digital Technical Journal, vol. 4, no. 3 (Summer 1992, this issue): 11-23.

19 Biography

Brian Porter As a consulting software engineer in the Systems Group of VMS Development, Brian Porter was responsible for CPU error handling in the VAX 6000 family. Prior to this work, he was responsible for support of VAX systems and was an author and maintainer of the VMS error log utility SYE. Brian is the author of the original VMS striping driver, which was later developed by others into the VMS striping driver product. He currently works in the Executive Group of VMS Development and is responsible for symmetric multiprocessing. He has two patents pending on memory error handling. Brian joined Digital in 1973.

20 Trademarks

The following are trademarks of Digital Equipment Corporation: VMS, VAX, and VAX 6000.