High Availability and Reliability in the Itanium Processor

Nhon Quach
Intel

THE ITANIUM PROCESSOR OFFERS RELIABILITY AND AVAILABILITY FOR MISSION-CRITICAL APPLICATIONS. THE AUTHOR DESCRIBES THE IMPLEMENTATION OF THE MAJOR FEATURES ON THE PROCESSOR AND EXPLAINS THE MOTIVATION AND DESIGN DECISIONS BEHIND THEM.

•••••• The Itanium processor is the first implementation of the Intel IA-64 architecture.[1] Designed for the high-end server market, the processor provides features (see the sidebar, next page) that maximize system reliability and availability. These features include fault coverage at the hardware level, a configurable multilevel error containment strategy, an availability feature set, and a machine check abort (MCA) architecture.

Reliability features

Soft errors are unintended transitions of logic states in a processor typically caused by external sources of ionizing radiation. The ionization creates excess free carriers, which recombine with the charges in the storage nodes, thereby corrupting information. There are two primary sources of external radiation that can contribute to soft errors. The first is alpha particles, found, as by-products of the radioactive decay process, in materials such as metal interconnects, solders, ceramics, plastics, or epoxies. The other source comes from high-energy neutrons. Ultra-high energy particles that originate in deep space generate these neutrons.

A soft error may or may not be detected, depending on the fault coverage in a processor. When undetected, a soft error can result in the undesirable outcome of silent data corruption. When detected, either the hardware or the software can handle the soft error in some manner, depending on the capability of the processor.

On a processor, memory cells are the most susceptible to soft errors because they constitute the majority of the devices and are typically built with small devices. More important, unlike static logic, which has the ability to mask errors,[2] memory cells "remember" any soft errors that occur as long as these cells are in use.

The Itanium processor is highly reliable as all large memory structures are protected by either error-correcting code (ECC) or parity. Figure 1 shows protected memory arrays as well as the availability features on the processor. In the figure, the hardware page walker handles translation look-aside buffer (TLB) misses and enforces coherency between the two TLB levels. The L1I and L1D caches are backed by the L2 and L3 caches. Table 1 lists the important attributes of the caches.

Together, these reliability features let the processor achieve low soft-error rates. Note

Key features

Certain features in the Itanium processor support the system:

- Either error-correcting code or a parity technique in the processor cache memory, state arrays, and translation look-aside buffers ensures reliability. Additionally, extra hardware resources allow error handling even when the processor is fielding another interrupt or exception.
- A configurable, multilevel error containment strategy to prevent errors from propagating outside the processor—at the system level via a hardware bus reset pin, at the cluster (or node) level via an early error indicator, and at the process level via a mechanism called data poisoning.
- Availability features including extensive error-logging capability for error analysis and diagnostics, a watchdog timer to prevent system hang (not executing) and a low-priority interrupt on all corrected errors to save the operating system from polling the processor error logs periodically.
- Enhanced machine check abort architecture lets the firmware, in close cooperation with the hardware, correct less-critical errors and communicate critical error information to the system firmware and the operating system more effectively.

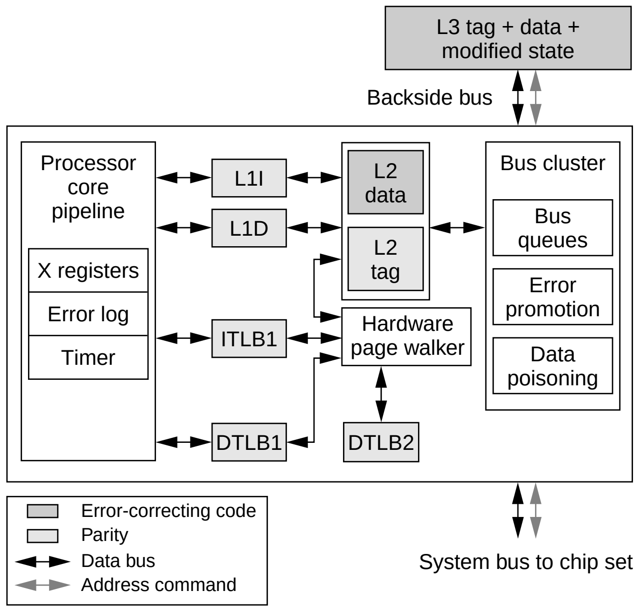

Figure 1. Itanium processor reliability and availability features.

Table 1. Major attributes of the Itanium processor caches.

Cache	Size (bytes)	Associativity	Line size (bytes)	Write policy
L1I	16 K	4 way	32	Read only
L1D	16 K	4 way	32	Write through
L2	96 K	6 way	64	Unified; write back
L3	2-4 M	4 way	64	Unified; write back

that the branch prediction and branch target buffers aren't protected because errors in these structures only cause harmless branch mispredictions.

Error detection and correction

Each structure has its own protection scheme.

TLBs. The critical fields in the instruction TLB1, data TLB1, and data TLB2 are protected by parity. When an error is detected in any of the TLBs, a hardware bus reset contains the error. This reset action is needed because by the time an error is detected in the data array, the memory transaction may have been sent to the memory subsystem. An alternative would be to delay submitting the memory transaction until after the parity detection. The Itanium processor design team deemed this as too costly in terms of performance and was therefore not adopted.

L1I instruction and L1D data caches. The processor uses parity for error detection in the L1I instruction tag and data arrays. An error in the tag or data array causes the cache controller to signal a local MCA. As defined in the IA-64 architecture, the PAL (processor abstraction layer)[3] error handler will then invalidate the entire L1I instruction cache to correct the error.

L1D uses a similar parity scheme for the tag and data arrays. However, the data arrays have one parity bit per byte. This finer parity granularity is motivated by the need to support byte (and multiple-byte) store operations without requiring a read-modify-write operation to recompute and update parity.

L1I errors are detected early in the processor pipeline and are fully correctable. L1D errors are detected one clock cycle before the instruction retirement stage, just in time to stop the errant data from being consumed, but too late to stop the errant data from corrupting the destination register on a data load. For this reason, L1D tag and data parity errors won't be correctable if the offending instruction is a data load. This is unfortunate, as data loads constitute a majority of the data traffic.

To solve this destination register corruption problem, the designers used a technique called redundant error detection in L1D. A local

MCA is signaled as long as a set being accessed contains an error in any of the four ways in a set. Therefore, a problem would only result when a data load hits the L1D error way exactly; in the other cases the PAL error handler will correct the errors by invalidating the entire cache before the errors become uncorrectable. This simple technique reduces the L1D error by a factor of four.

Multiple bit errors in a single cache line caused by a single upset event are prevented in all caches by interleaving bits from adjacent cache lines in the physical layout. All caches use this bit-interleaving technique; when multiple bit upsets occur, the result is multiple cache lines with single-bit errors rather than a single cache line with multiple-bit errors.

L2 unified cache. The L2 data array is protected for every 64 bits of data by an 8-bit ECC, which provides single error correction and double error detection.[4] However, 8-, 16-, and 32-bit stores require a read-modify-write operation in the cache to recompute and update the ECC bits. A 32-bit ECC chunk would eliminate read-modify-write operations for 32-bit stores. This requires 7 ECC bits for each 32-bit data group, or 14 ECC bits for 64 data bits—almost twice as much overhead as needed for 64-bit ECC chunks.

The L2 tag array is protected by parity and employs the same redundant error detection technique as the L1D cache. On error detection (in any of the ways in a set), the L2 cache controller will signal a local MCA if the cache line state is either shared or clean and will signal a global MCA if the cache line state is modified. A parity error detected in a clean or shared cache line may or may not be correctable, again depending on whether the offending instruction is a data load. Raising a local MCA gives the PAL error handler a chance to correct the error. Raising a global MCA ensures termination of all potentially affected processes.

L3 unified cache. Three parity bits protect the L3 cache tag. While these parity bits detect tag errors, the eight ECC check bits shared between the tag and the 64-bit data correct the errors. The ECC check bits detect data errors as well.

The hardware corrects both tag and data errors without the intervention of the PAL error handler (that is, no MCA is signaled). When the parity bits detect a tag error, the processor aborts the offending transaction, suspends all other operations to and from the L3 cache, and starts a tag correction state machine to correct (or scrub) the tags using the ECC check bits. The scrubbing is performed on all four ways in a set in the L3 cache. After scrubbing, the state machine accesses the cache tags again to ensure proper correction. If the scrubbing is successful, the aborted offending transaction is retried and normal operation resumed; otherwise, a hardware bus reset is initiated for a hard error (that is, stuck-at faults).

In contrast, single-bit ECC errors in the data arrays are detected and corrected on the fly as the data returns to the processor core. No cache scrubbing is performed, and the errant line remains uncorrected in the cache. It's possible for a second, unrelated, error to happen to the same line, turning a single-bit ECC error into a double-bit ECC error. The probability of this event, however, is extremely small since soft errors are relatively infrequent. The errant line would most likely be evicted before it had a chance to accumulate additional errors. The processor will correct on the fly a cache line with a single-bit ECC error as it is evicted to the system memory (or to other processors).

Because of its large size, a sparse encoding scheme protects the L3 cache state bits. In a conventional encoding scheme, two bits are used to encode the modified, exclusive, shared, and invalid states of a cache line. The Itanium processor uses four bits for this purpose as shown in Table 2.

Table 2. Itanium processor L3 cache state encoding, error detection, and correction.

Encoding	State	Single-bit error detected if:	Processor corrects to the following state
1111	Modified	Any of the state bits is zero	Modified
0001	Exclusive	Either none or two bits set	Invalid
0010	Shared	—	—
0100	Invalid	—	—

The modified state is encoded such that its Hamming distance from the exclusive, shared, and invalid states is three. On each access, the L3 cache returns the state information and an error indicator showing the integrity of the state bits. On an error, the processor writes back a line with bad modified state (that is, 1101) to a line with clean modified state (that is, 1111) and invalidates a line with other bad states.

Processor backside and system buses. The processor backside and system address as well as command buses are parity protected. The system and backside data buses have 64 and 128 bits, respectively. Eight ECC bits per 64 bits of data protect these data buses. The ECC logic on the system bus can detect 4-bit block errors. A parity error in the address or command bus causes the processor to initiate a hardware bus reset. A single-bit ECC error in the data bus is corrected on the fly by the processor hardware. A double-bit ECC error in the data bus triggers a data-poisoning event.

X resources

To handle exceptions and interrupts, the IA-64 architecture defines a set of interruption registers for holding critical processor states. During exception or interrupt handling, interrupts are masked and further updates of these interruption registers are disabled. The operating system is responsible for ensuring that no exceptions occur. In this way, the saved processor states' integrity is guaranteed. However, since MCA cannot be masked, an MCA that occurs during exception or interrupt handling will corrupt critical processor states of the executing process. For these reasons, the IA-64 architecture defines a set of X resources[3] to handle these instances. These resources include a register set (X registers) for holding critical processor states and a scratch register set for use by the PAL error handler during error handling.

On an MCA, the X registers receive the contents of the interruption registers. The interruption registers in turn receive critical states of the executing process. The X registers enable MCA handling when the processor is fielding another interrupt or exception.

Error containment

A critical, but often overlooked, aspect of processor user features is error containment. Error containment means that the processor should never allow a detected error to get to the system bus. A common industry error containment strategy reboots the entire system on any detected errors that hardware cannot correct. While this strategy might be effective in desktop systems, it's hardly desirable for high-end servers. To maximize system availability, the Itanium processor uses a more refined strategy for error containment.

As the first line of defense, instructions with single-bit errors detected by parity in the instruction cache are never executed, as guaranteed by the processor design. Single-bit errors in data detected in the L2 and L3 caches will be corrected on the fly. This is instruction-level error containment.

Second, double-bit errors from the system bus and the L2 cache (detected by ECC), or single-bit errors in the L1 and L2 caches (detected by parity) will signal an MCA before the bad data is consumed. A data-poisoning mechanism handles double-bit errors in data from the system bus or the L2 cache that aren't immediately consumed by an instruction. This is process-level error containment.

Third, single-bit errors in addresses detected in the cache tags involving dirty lines that don't require immediate error containment (for example, during data load and store operations) will signal a global MCA. Assertion of the global MCA pin provides an early error indicator, letting the systems take proper action, such as turning off all memory traffic to the I/O regions and so on. The affected cluster (or node) is then gracefully shut down. This is cluster-level error containment.

Finally, single-bit errors in addresses detected in the cache tags involving dirty lines, or in the TLBs requiring immediate error containment (for example, snoops hitting a dirty line with a bad address) cause the processor to assert the hardware bus reset pin. This is system-level error containment.

Data poisoning

Because of the large main memory size, double-bit ECC errors in the main memory may propagate into the processors. Traditionally, double-bit ECC errors detected in the main memory will cause the platform to assert a global MCA, bringing down the system.[5] The Itanium processor implements an error containment scheme called data poisoning. When a double-bit ECC error is detected in data returned on the system bus, the processor marks the data as bad (poisons it) in the processor cache. Processes consuming poisoned data are terminated; the rest of the system remains intact.

As an optimization, the processor also poisons a cache line if it receives a hard-error response on the system bus. This gives platform designers additional flexibility in signaling errors to the processor. A hard-error response typically indicates platform errors such as parity errors in the PCI bus.

Data poisoning takes place at a cache line boundary by forcing a double-bit ECC error in the entire cache line. Poisoned data propagates among processors and memory agents. The rules for handling a poisoned cache line within a processor are as follows:

1. A load or fetch from a poisoned cache line signals a local MCA.
2. A store to a poisoned cache line is ignored. The line in the cache remains unchanged and poisoned.
3. A snoop or eviction causes the processor to write back the cache line with the double-bit ECC error to the system bus and signals a corrected machine check interrupt.[3]

Implementing data poisoning adds significant hardware complexity. The processor implements three simplifications. First, the cache line granularity data poisoning simplifies the hardware poisoning logic. On a cache line fill or eviction, data in a cache line are moved between the bus queues and the L2 cache on a subcache line basis, taking multiple cycles. However, the result of the double-bit ECC error detection isn't known until the data are already stored in the cache line fill buffer (in a fill) or the bus write-back queue (in an eviction). Poisoning at a cache line boundary means that on error detection the logic doesn't need to recall the errant data chunk(s) and only needs to blindly set the data poisoning bit for the entire cache line.

Next, the L1I, L1D, and the L3 caches don't participate in data poisoning. On the way into the processor, the data are returned from the bus queues to the processor core on data loads. If a double-bit ECC error is detected, the L3 cache controller won't write back the line to the L3 cache but will invalidate it. For the L1I and L1D caches, the local MCA signaled by the processor, as required by Rule 1, lets the PAL error handler invalidate the errant line. On the way out of the processor, modified data in the L3 cache are evicted to the bus queues. If a line has a double-bit ECC error, a hardware bus reset is initiated. This is acceptable because the probability of a double-bit ECC error is negligible in the L3 cache.

Finally, no special double-bit ECC error pattern is used to indicate the poisoned data inside the L2 cache, in the bus queues, and on the system bus. This enables passing, as is, a cache line with a double-bit ECC error, without inserting a special double-bit ECC error pattern.

Data poisoning requires platform cooperation. Platforms that support data poisoning must not initiate a hardware bus reset when detecting double-bit ECC errors on processor-initiated memory read and write transactions. The errant line should be returned to the processor or written into the memory with the double-bit ECC errors.

Error promotion

A challenge to the processor MCA implementation is the diverse types of platforms and operating systems that the processor must run. These platforms and operating systems have different processor behavior expectations on error detection. As an example, most platforms want to maximize system reliability by letting hardware correct detected single-bit ECC errors. However, in fault-tolerant applications, where two processors are running in lockstep, hardware correction will cause a problem because the normal data return path and the single-bit ECC correction path have different timings. This will eventually cause the processors to fall out of lockstep. For these reasons, it's actually preferable to promote a hardware-correctable error to a global MCA. External logic residing in the platform, observing the global MCA pin, may then take early corrective action.

As another example, certain software atomic events (such as accessing a platform error log or configuration registers) are noninterruptible even by an MCA. An MCA that occurs during one of these atomic events may cause loss of error containment because the processor or platform may be in an inconsistent state. For these reasons, during these events it's important to temporarily promote a local MCA into a global MCA or a hardware bus reset.

These examples suggest that a certain level of processor configurability is required. On the Itanium processor, less severe errors may be promoted to more severe ones via a configuration model-specific register, manipulated via a defined PAL interface.[3] However, to simplify the hardware, only the following promotion options are implemented:

- A hardware-correctable error may be promoted to take a local MCA.
- A local MCA (including a promoted one from a hardware-correctable error) may be promoted to signal a global MCA.
- A global MCA (including a promoted one from a local MCA) may be promoted to take a hardware bus reset.

Availability features

The Itanium processor's emphasis on system availability is evidenced by its error containment strategy. Additional availability features are also implemented on the processor.

Watchdog timer

The processor implements a 64-bit watchdog timer, the size of which roughly corresponds to a time interval of eight seconds. This timer measures the time elapsed between two instructions that are retired and is reset on every instruction retirement. On time-out, the processor initiates a hardware bus reset, breaking a potential system hang condition where the system is not executing.

Two factors determine timer size. One is the longest time an instruction may take to retire. On an Itanium-processor-based platform, this instruction is an uncacheable load to a slow device on the ISA bus, which can take milliseconds to complete. The other determining factor is the highest frequency that the processor and its proliferation may run. A 64-bit timer provides a good margin in both cases.

Error logging

The processor provides an extensive error information logging capability. Each error log is implemented as a model-specific register, accessible via an internal processor bus by the PAL error handler during the error-handling process. Each error log consists of the following fields:

- A signaling field indicating whether a corrected machine check interrupt, a local MCA, a global MCA, or a hardware bus reset is pending.
- A status field containing a valid bit, an overflow bit, the physical address of the offending operation (if physical address logging is available), its status (hits or misses in the cache), an error indicator (error in the cache data or tag array), and other related information.
- A control field controlling whether an error is promoted.

The signaling and status fields are sticky, set by the hardware and cleared only by the PAL error handler. Either the system abstraction layer (SAL) firmware[6] or the operating system may set the control field (via a PAL call), as described previously. The hardware ensures that the error log content is correct at all times, under all pipeline conditions and microarchitectural corner cases. This has proven difficult to implement. On many occasions, to simplify the hardware, the design team had to settle for less information, resulting in fewer errors that can be corrected.

As a further simplification, the Itanium processor implements a log-first-error policy, rather than logging the most severe error. Later errors cause the processor to set an overflow bit if there's a chance of losing critical information. This causes the PAL error handler to return a nonrecoverable error indicator to the SAL and the operating system.

Corrected errors notification

An effective way of improving system availability is to provide important error information early to the system maintenance staff. Early notification enables corrective action before a correctable error becomes uncorrectable. As an example, if the processor is experiencing an unusually high number of correctable errors, it may be prudent to preemptively replace the processor, or the entire processor board, if the number of errors exceeds an empirical threshold. For this reason, the processor and the PAL error handler send a corrected machine check interrupt on all hardware- or PAL-corrected errors. This relieves the operating system of the burden of polling the error logs periodically, slowing the system response time. The corrected machine check interrupt is an IA-

64 architecture interrupt and is typically given a low priority.

Processor MCA architecture

Perhaps the most notable enhancement to the IA-64 MCA architecture is handling errors in the PAL error handler, invoked via an MCA. The Itanium processor implements all MCA events as asynchronous fault events so that traps that happen at the same time as the MCAs are not lost. An MCA may be signaled either locally or globally, depending on the severity of the errors.

In a local MCA, only the local processor detecting the error enters the PAL error handler. The remote processors aren't affected. No external pins are asserted and no external special cycle issued on the system bus. The PAL error handler, residing in an uncacheable space that is defined in the architecture, runs in physical mode with address translation turned off. Running from an uncacheable space in physical mode with address translation turned off ensures that the TLBs and caches are not used during the PAL error handling stage (SAL or operating system error handling may turn on translation and run in cacheable space). Because it is defined in the architecture, the uncacheable space ensures that the PAL error handler is immediately available after reset.

In a global MCA, all processors on the same system bus (and other system buses, depending on the system configuration) enter the PAL error handler. The local processor asserts the global MCA pin. All other processors on the same system bus (or other system buses) will enter the PAL error handler.

A hardware bus reset causes a similar system response as when a global MCA occurs; all processors in the system will enter the PAL error handler. Additionally, assertion of the hardware bus reset pin clears the following states in the processors:

- certain relevant microarchitectural states; however, all architectural states are preserved including the general and floating-point register files, the caches, and the TLBs;
- all outstanding transactions in the memory and bus queues, including the state tracking logic in the bus unit; and
- all internal state machines, including the lock state machines in the bus unit.

Clearing these states ensures forward progress so that after the assertion of the hardware bus reset pin, the first processor instruction fetch will be issued to the PAL error handler.

Error handling flow

Figure 2 (next page) shows a simplified hardware and PAL error-handling flow on the Itanium processor. Hardware-correctable errors are corrected silently and the offending process runs uninterrupted. An MCA is only signaled for errors that require PAL error handler intervention.

Upon entry, the PAL error handler performs a queue-draining operation (by allowing all outstanding transactions in the memory and bus queues to complete) to quiet the system before attempting any error correction. The handler then performs an initial error analysis and corrects as many errors as possible. SAL or operating system code is responsible for a more detailed analysis, correcting any platform errors and terminating the offending process as needed.

Before exit, the PAL error handler checks if another MCA has occurred during the error-handling process.

Multiple or nested MCAs

Handling multiple MCAs is always a challenge for any MCA architecture. Multiple MCAs are a result of either the same error detected multiple times or different errors detected at the same time. The Itanium processor uses a simple overflow bit in the error log to handle these cases. This bit is set whenever critical error information needed for error handling (usually the physical address of the offending instruction) is lost. All errors that occur before the queue-draining operation are handled serially if the overflow bit is clear. If the overflow bit is set, the PAL error handler returns a nonrecoverable indicator to the SAL or operating system. This causes a cluster or systemwide reset, depending on system configuration and the capability of the operating system.

An error that occurs after the queue-draining operation is considered a nested MCA. To handle this case, the PAL error handler clears all the MCA pending bits in the processor upon entry. If another MCA is detected after

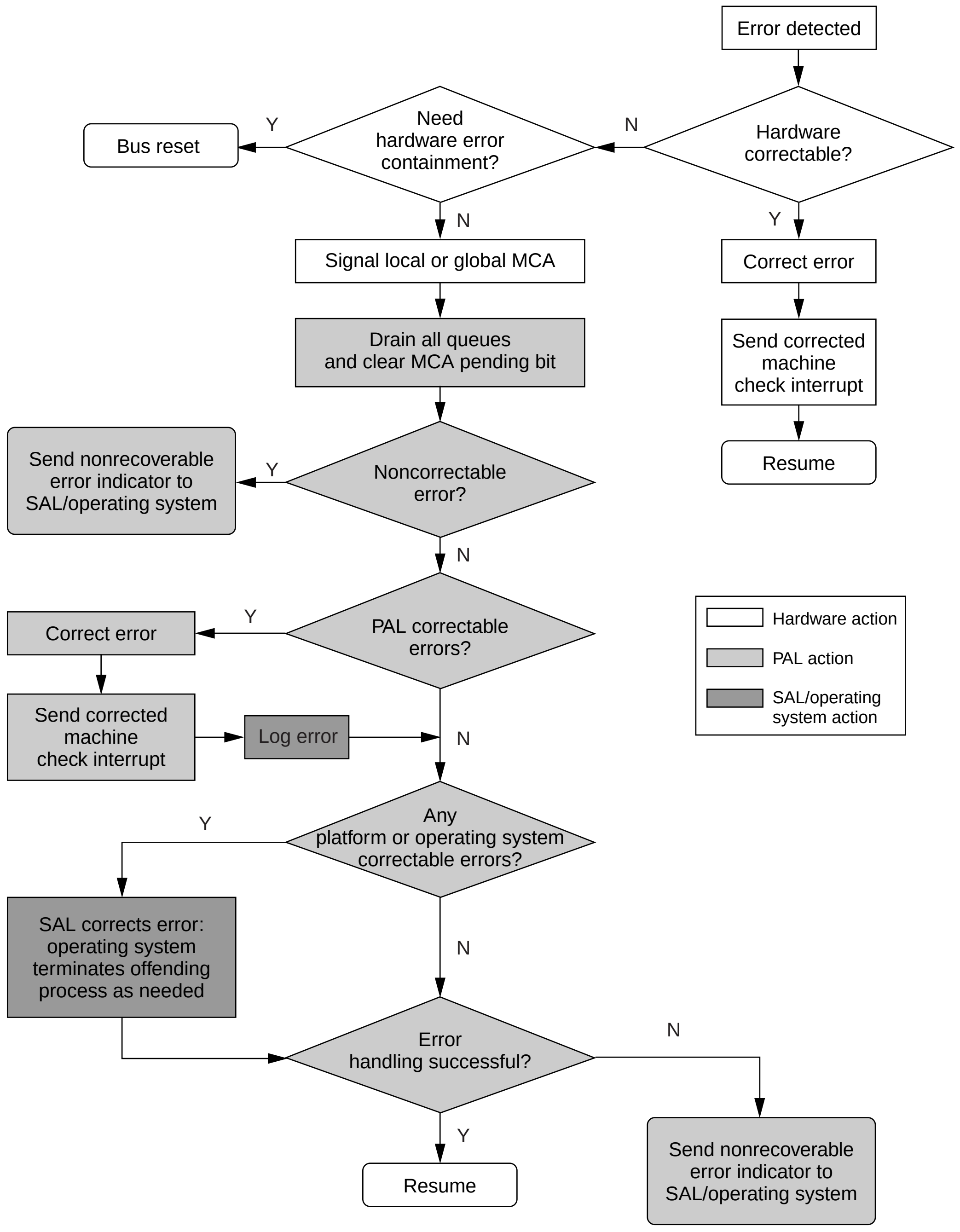
Error detected
Hardware correctable?
N
Y
Need hardware error containment?
Y
Bus reset
N
Signal local or global MCA
Drain all queues and clear MCA pending bit
Correct error
Send corrected machine check interrupt
Resume
Noncorrectable error?
Y
Send nonrecoverable error indicator to SAL/operating system
N
PAL correctable errors?
Y
Correct error
Send corrected machine check interrupt
Log error
N
Hardware action
PAL action
SAL/operating system action
Any platform or operating system correctable errors?
Y
SAL corrects error: operating system terminates offending process as needed
N
Error handling successful?
Y
Resume
N
Send nonrecoverable error indicator to SAL/operating system

Figure 2. Simplified hardware and PAL error-handling flow.

this point, the processor error-logging logic will set the MCA pending bit. This MCA won't be handled immediately since MCAs are masked during the error-handling process.

It's an operating system policy to determine whether to handle nested MCAs. A less-capable operating system won't enable MCAs during the error-handling process. If a nested MCA occurs, the MCA pending bit is set and remains set on exit from the handler. This is considered a nonrecoverable error that will be reported once MCAs are enabled again.

A more-capable operating system that handles nested MCAs may enable an MCA after it has logged the error information in the platform and cleared the error logs in the processor, as

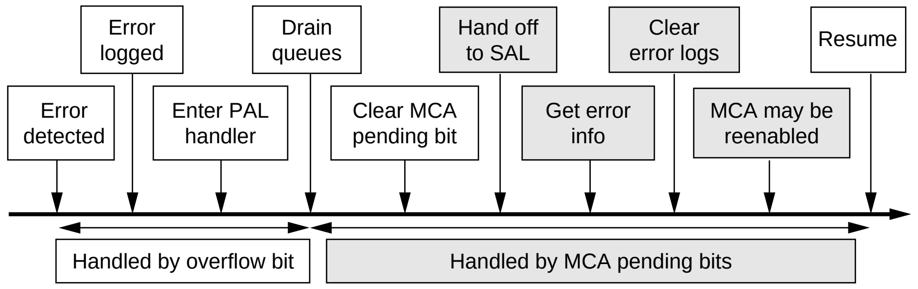

Figure 3. Handling multiple and nested MCAs using overflow and MCA pending bits.

shown in Figure 3. In this way, nested MCAs are also handled serially, and all MCA pending bits will stay cleared on exiting from the handler. The processor MCA architecture enables this handling in a transparent and elegant manner.

Handling platform errors

The Itanium processor MCA architecture works seamlessly with the platform-error handling firmware in the SAL layer. Platform errors may be signaled to the processor via the global MCA pin, the hardware bus reset pin, a double-bit ECC data error for read operations, or a hard-error response for read and write operations. In all cases, the processor signals an MCA and transfers control to the PAL error handler, which, in turn, hands off control to the SAL error handler via an architectural interface.[3] In general, the SAL error-handling flow is platform specific and therefore not covered in this article.

High availability and reliability are easily two of the most important system attributes in the high-end server space. With its extensive fault coverage and hardware and firmware support, the Itanium processor provides an excellent basic set of high availability and reliability features for future IA-64 processors to build upon. Work is already well under way to collect data and measure the effectiveness of these features on the Itanium processor. MICRO

Acknowledgments

Many people have contributed to the concept, implementation, and validation of the Itanium processor architecture and the PAL error handler. In particular, I would like to thank the following individuals: Valentine Anders and Jim Hays from Hewlett-Packard and Gary Hammond, John Crawford, Rad Mahalikudi, Kiran Desai, Amy O'Donnell, Susan Mar, Suresh Marisetty, and Mani Ayyar from Intel.

References

1. *Intel IA-64 Architecture Software Developer's Manual*, Vol. 1, Intel, Santa Clara, Calif., Order No. 245317-001, Jan. 2000.
2. P. Liden et al., "On Latching Probability of Particle Induced Transients in Combinational Networks," *Proc. Symp. Fault Tolerant Computing (FTCS-24)*, IEEE Press, Piscataway, N.J., 1994, pp. 340-349.
3. *Intel IA-64 Architecture Software Developer's Manual*, Vol. 2, Intel, Santa Clara, Order No. 245318-001, Jan. 2000.
4. C.L. Chen and M.Y. Hsiao, "Error Correcting Codes for Semiconductor Memory Applications: A State-of-the-Art Review," *IBM J. Research and Development*, Vol. 28, No. 2, Mar. 1984.
5. *Intel 450NX PCIset Datasheet*, Rev. 1.3, Intel, Mar. 1999.
6. *IA-64 System Abstract Layer Specification*, Intel, Order No. 245359-001, Jan. 2000.

Nhon Quach is the manager of the IA-64 system architecture at Intel. Quach received a PhD in computer architecture from Stanford University, an MS degree in silicon processing and device physics from MIT, and a BS degree in computer engineering from the University of Texas at Austin. His technical interests include computer system architecture, fault-tolerant computing, and advanced microarchitecture techniques. He has written more than 20 publications and holds more than 20 patents.

Direct comments and questions about this article to Nhon Quach, Intel, 2200 Mission College Blvd., Santa Clara, CA 95054; nhon.quach@intel.com.