Xen's Machine Check Architecture implementation for Intel® Processors

Written 6-14-12

Updated 0-0-0

This work is to document Xen's implementation of Intel's Machine Check Architecture (MCA) and surrounding features. To this end, let's consider what MCA is and where it fits into the system services.

Intel's Machine Check Architecture

MCA is a hardware based processor centric error detection and reporting mechanism. Figure 1, Error classification, decomposes system errors into the classes of interest. The two main classes are detected and undetected errors. The undetected errors, while they may be important, are, by their very nature, not handled by the MCA. Hence, Xen's implementation covered in this document does not address the issue of undetected errors either. Detected errors, on the other hand, are addressed and are the reason for the Machine Check Architecture's existence.

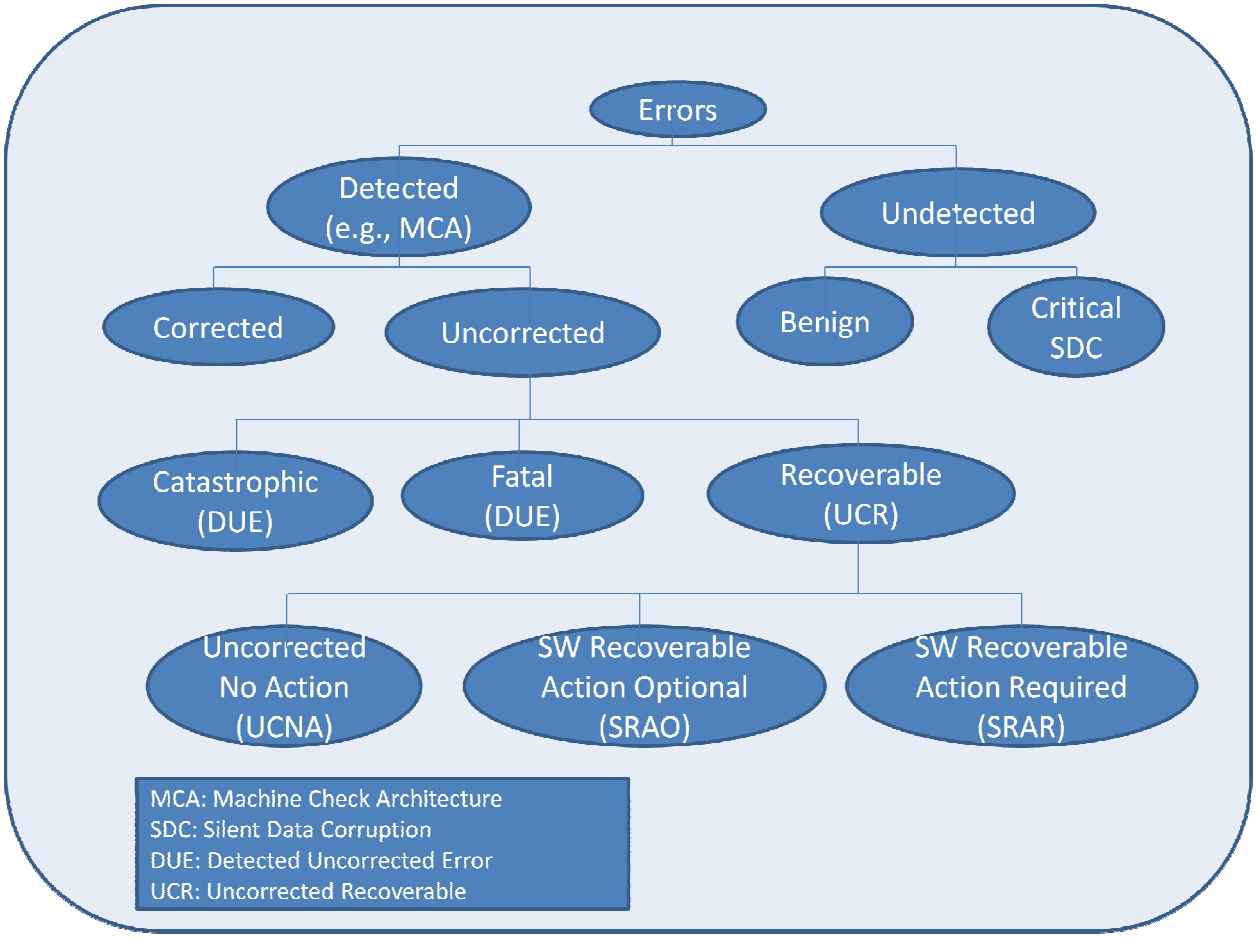

Figure 1, Error classification

Detected errors are split into corrected and uncorrected errors. Corrected errors pose no threat to the system once they have been corrected. Despite their harmless nature they, along with the uncorrected errors, are still made visible to software for tracking purposes. Some errors are harbingers of problems to come so it is important to be able to monitor them. This monitoring allows failure analysis to be done so parts that are likely to fail can be replaced proactively rather than under sudden failure conditions.

Uncorrected errors decompose into three classes: Catastrophic, Fatal and Recoverable. Both Catastrophic and Fatal cause the system to be reset in all cases. Uncorrected Recoverable errors (UCR) have the potential for the system to continue to function but not in all cases. In particular, the Software Recoverable Action Required class requires that software take action to correct the problem. If software fails to correct the problem the system must still be reset.

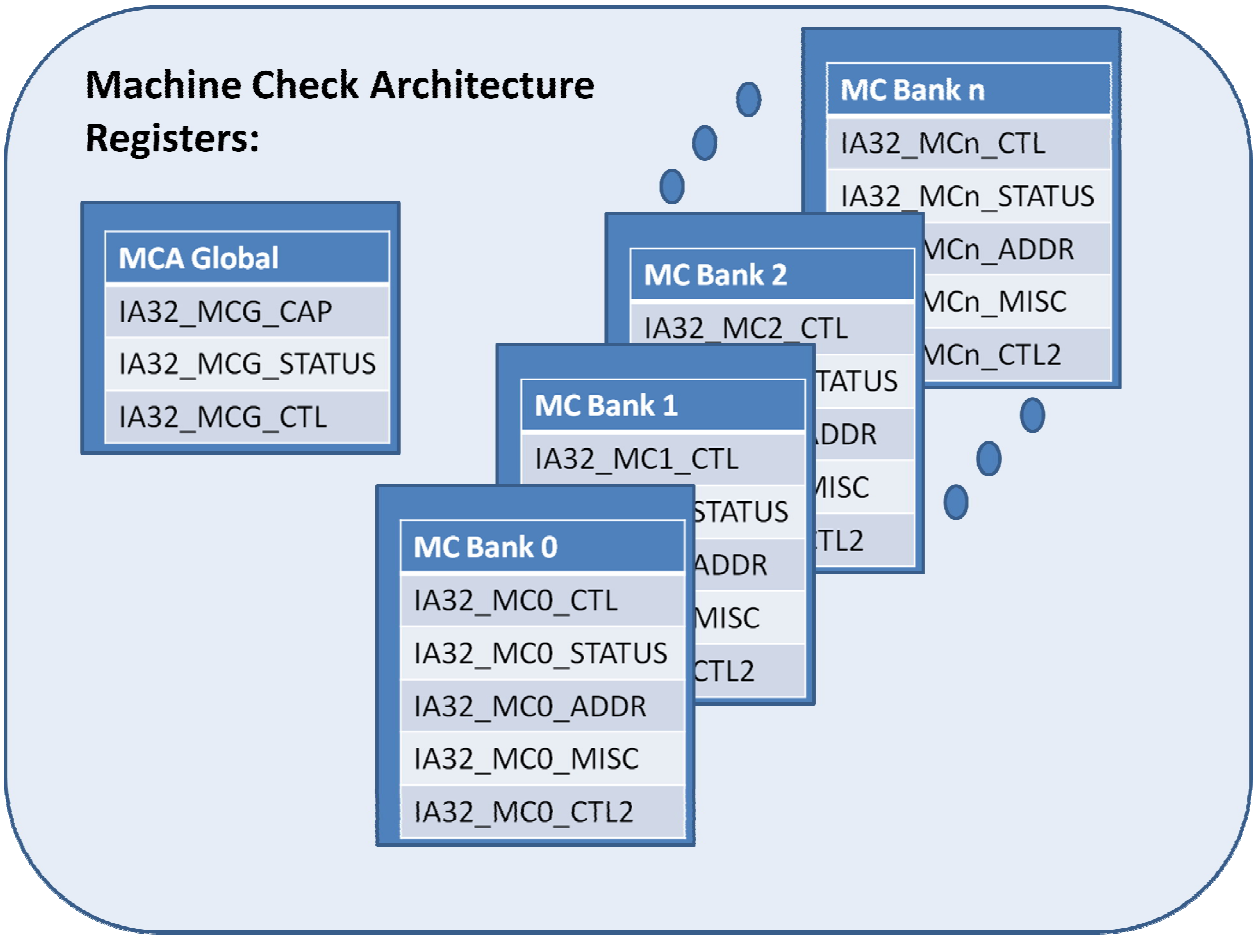

Figure 2, Machine Check Architecture Global and banked registers

Given these error classes detected by hardware, mechanisms have been defined for notifying software (BIOS, Firmware, Operating System and / or Hypervisors) of the detected errors and the actions needed to be preformed. These mechanisms have two flavors, logging and signaling. Logging is done by writing information into the Machine Check (MC) registers (Figure 2, Machine Check Architecture Global and banked registers) where it can be read by system software and transferred to system logs. The MC

registers include a set of global registers and a number of banked sets of registers. Signaling uses a few different signals, CMCI and MCE.

CMCI (Corrected Machine Check Interrupt) is used for signaling Corrected Errors (CE) and UnCorrected Recoverable (UCR) like UnCorrected No Action (UCNA) recoverable errors. MCE (Machine Check Exception) is used for uncorrected recoverable like Software Recoverable Action Optional (SRAO) and Software Recoverable Action Required (SRAR) as well as fatal errors that require the system be reset. In all these cases information about the error event is written to the MC banks prior to signaling. CMCI signal is sent to only those hardware threads in the affected socket while MCE signals are sent to all hardware threads in the system.

The MCA Global Capabilities register, IA32_MCG_CAP, defines the capabilities offered by the system. Its interpretation is defined in Figure 3, IA32_MCG_CAP coding.

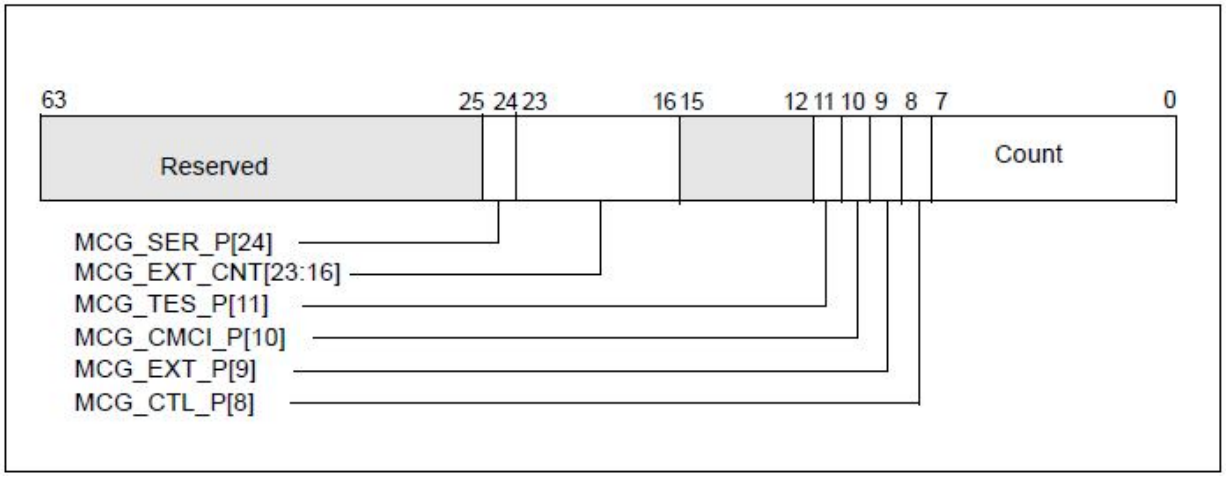

Figure 3, IA32_MCG_CAP coding

As shown, bits [7:0] hold the number of banks supported by the system. Each of the fields ending with an "_P" (MCG_SER_P, MCG_TES_P, MCG_CMCI_P, MCG_EXT_P and MCG_CTL_P) indicate the existence of a feature or register and should always be checked before utilizing the resource they represent.

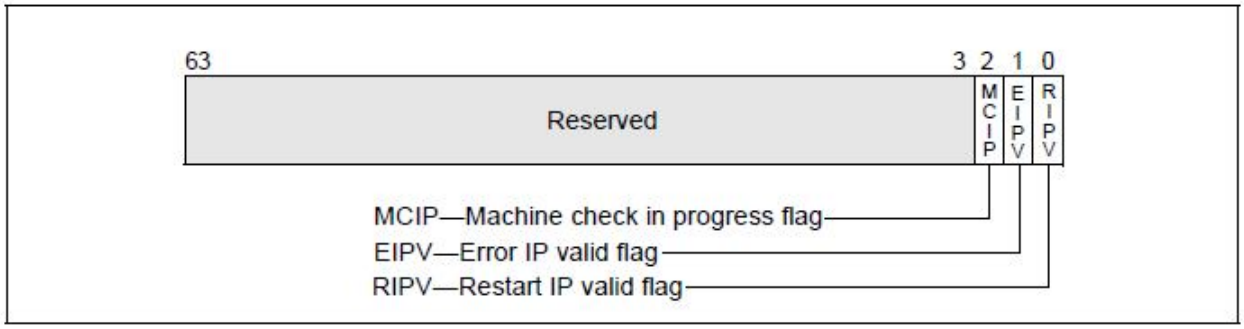

Figure 4, IA32_MCG_STATUS coding

The coding for information in the global status register, IA32_MCG_STATUS, is given in Figure 4, IA32_MCG_STATUS coding. The rest of the registers are banked with each bank having its own control register, IA32_MCi_CTL, and status register, IA32_MCi_STATUS. The bank control register, IA32_MCi_CTL, should be initialized with all 1s or all 0s to enable or disable MCE signaling respectively. MC bank status register coding is given in Figure 5, IA32_MCi_STATUS coding. The rest of the potential registers for each bank are only valid when indicated by their respective valid bits and should not be accessed unless their valid bit is asserted. Both the MC bank addr and misc registers (IA32_MCi_ADDR and IA32_MCi_MISC) have their respective valid bits in IA32_MCi_STATUS while IA32_MCi_CTL2's valid indicator is located in IA32_MCG_CAP[10].

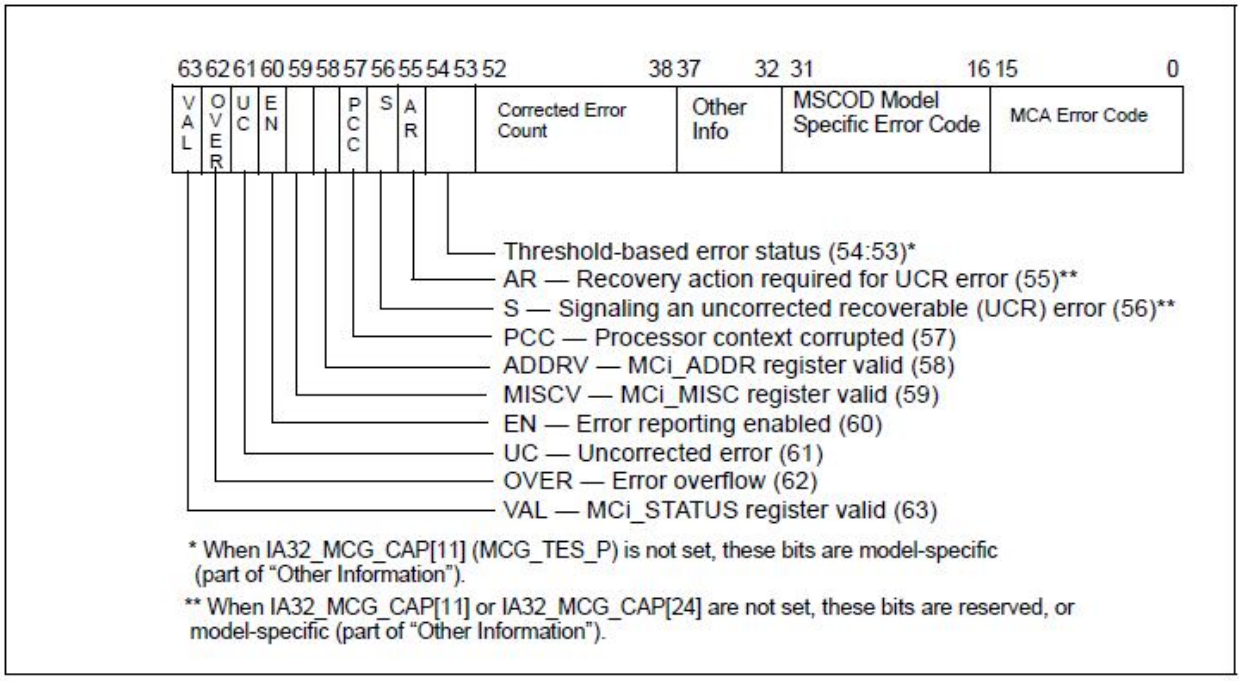

Figure 5, IA32_MCi_STATUS coding

A brief summary of errors that are reported using the Machine Check Architecture is given in Figure 6, Machine Check Error Classification. Standard UnCorrected errors (UC) require a reset of the system and are signaled with MCE. The special case of UnCorrected errors with No Action (UCNA) required and Corrected errors both use the Corrected Machine Check Interrupt (CMCI) signal to notify software of their existence but this is for logging purposes only. The last two are Software Recoverable Action Optional (SRAO) and Software Recoverable Action Required (SRAR) and use MCE to notify software. As their names imply they are for error cases that are recoverable and for which software may be able to take action to ensure system integrity.

Type of Error[1]	UC	PCC	S	AR	Signaling	Software Action	Example
Uncorrected Error (UC)	1	1	x	x	MCE	Reset the system	
SRAR	1	0	1	1	MCE	For known MCACOD, take specific recovery action; For unknown MCACOD, must bugcheck	Cache to processor load error
SRAO	1	0	1	0	MCE	For known MCACOD, take specific recovery action; For unknown MCACOD, OK to keep the system running	Patrol scrub and explicit writeback poison errors
UCNA	1	0	0	0	CMC	Log the error and Ok to keep the system running	Poison detection error
Corrected Error (CE)	0	0	x	x	CMC	Log the error and no corrective action required	ECC in caches and memory

NOTES:

1. VAL=1, EN=1 for UC=1 errors; OVER=0 for UC=1 and PCC=0 errors SRAR, SRAO and UCNA errors are supported by the processor only when IA32_MCG_CAP[24] (MCG_SER_P) is set.

Figure 6, Machine Check Error Classification

A more complete description of the Machine Check Architecture is documented in: Intel® 64 and IA-32 Architectures Software Developer's Manual, Volume 3B.

Xen's Implementation

With this, the above, background we are ready to discuss Xen's implementation to support Intel's® Machine Check Architecture. Xen's implementation supports both logging and signaling aspects of MCA. Specifically as Xen provides a virtualized environment for its guest operating systems, it could choose to expose all, some or none of the underlying MCA capabilities to its guests. Each guest is called a domain. Domain 0, or Dom0 is a privileged guest that is used to manage system resources and the other user domains, or the other DomUs.

Xen follows two main imperatives:

- Never allow a corrupted system to continue, and
- Without risking the above point keep the system up and running as long as possible

If the first of these two is ever at risk, the system must be reset. However, whenever possible, it is better to kill a process in a guest or even a guest itself while retaining the integrity of the rest of the system. In

any case, it is important to provide information on what caused the issue if possible. Xen's MCA related code is located in:

XEN_SOURCE/arch/x86/cpu/mcheck.

This includes the common code as well as the Intel® and AMD architecture specific code.

Xen Hypervisor

As we have seen, hardware provides for both logging and signaling of error information. Software must, however, initialize the system for MCA to be active. Essentially this entails verifying that the system supports MCA, determining the number of MC banks and initializing their values. In cases where the system has been reset there may also be error information in these registers that is needed for analysis of the previous run. Xen's initialization routine is, intel_mcheck_init(), and is located in mce_intel.c.

MCA related error information (i.e., ignoring IO facilities) logging is written into the MCA registers where software can access them. Many of these errors are corrected by hardware before their information is written to the MC banks but without a signal to indicate their presence. For logging purposes and predictive analysis Xen implements a polling mechanism which polls the MCA registers every 30 seconds to discover and log the Corrected Errors (CE) that are never signaled. The polling time is halved each time multiple errors are found and returned to its original value with some hysteresis to prevent vacillation. The main routine, mce_checkregs(), for this polling mechanism is located in non-fatal.c.

Mce_checkregs() simply walks the MC banks using mcheck_mca_logout() from mce.c checking IA32_MCi_STATUS[63] for asserted bits. Each bank that has this bit set will have error information that will be logged and any needed actions taken. The IA32_MCi_STATUS register will be cleared if the error is recoverable. If not recoverable, the information in the MC bank register values will be left intact for analysis after reset during boot. Once the local logging and actions have been completed, Xen informs Dom0 of any errors it found through the use of VIRQ.

Corrected Machine Check Interrupt (CMCI) handler, smp_cmci_interrupt(), uses mcheck_mca_logout() as its work horse because the actions needed are the same as those needed while polling the MCA banks. smp_cmci_interrupt() is located in mce_intel.c. The main difference between this mechanism and the polling mechanism is that each hardware thread in the socket is signaled so, in many cases, there will be more than one thread involved. Thread synchronization is used to choose a "Monarch" thread while all other threads wait for the monarch to complete the work.

Xen's Machine Check Exception (MCE) handler, intel_machine_check(), is located at mce_intel.c and also uses mcheck_mca_logout(). However, in addition to dealing with multiple threads as does CMCI, the MCE handler needs to verify that at least one MCE class error was found because MCE is not allowed to have spurious interrupts. All spurious interrupts result in system reset.

Thus far the above is mostly the same for most kernels enabling MCA with the exception of the VIRQ Xen sends to Dom0. The information available to Dom0 through VIRQ extends beyond what is available in the MCA registers. Along with the MCA information the physical cpu number, TSC (Time Stamp

Counter) when the event occurred and additional FRU[1] (Field Replaceable Unit) information may also be included.

In addition to the vIRQ, Xen maintains a set of virtual MCA registers for each guest, Dom0 and DomU alike.[2] Every event seen by Xen through the MCA registers is translated into values Dom0 will understand prior the VIRQ notification. Guests (Dom0 and DomU) have two MC banks virtualized.[3] A single bank (Bank 1) is used for all errors that are reflected to the guests.[4]

The virtualized MCA registers for DomU guests has a strictly define set of capabilities identified in its vIA32_MCG_CAP register as follows:

DomU vIA32_MCG_CAP capabilities	Description	Defined DomU Values
Count field[7:0]	MC bank count	0x2
MCG_CTL_P[8]	Disable MCG_CTL	0
MCG_EXT_P[9]	No extended regs	0
MCG_CMCI_P[10]	Enable CMCI signaling	1
MCG_TES_P[11]	IA32_MCi_STATUS[56:53] are architectural	1
MCG_EXT_CNT[23:16]	Meaningful only if MCG_EXT_P=1	0000,0000
MCG_SER_P[24]	Support s/w error recovery, must have MCG_TES_P	1
Reserved[63:25]	Reserved bits	N/A

Figure 7, vIA32_MCG_CAP field values defined for DomU

[1] Firmware may assist in determining FRU information.
[2] The MSR read / write operations for the MCA registers is implemented in the general protection (GP) fault handler for para-virtualized (PV) guests and in the VMExit handler for hardware virtual machine (HVM) guests.
[3] The sole reason for two banks is that Linux ignores bank 0 on certain AMD processors.
[4] Current plan is to only deliver a single MCA related event at a time to a guest. Later, we plan to extend the capability to include multiple events which may be easier to handle with more than one MC bank.

The vIA32_MCG_CAP register will be read only and its field values have been defined to enable only those capabilities needed to enable DomU guest to attend to errors that have the potential for improving it situation. The vIA32_MCi_CTL registers will be set to all '1's while vIA32_MCi_STATUS, vIA32_MCi_MISC, vIA32_MCi_ADDR and the vIA32_MCi_CTL2 will all be cleared initially.

As given in Figure 7, vIA32_MCG_CAP field values defined for DomU, two MC Banks will be virtualized. No IA32_MCG_CTL or extended registers will be virtualized. CMCI signaling will be enabled but at present no CMCI signals will be delivered. This is to limit the amount of work done by guests that monitor this category of error and might otherwise poll the MC banks for error information. Support for software error recovery is enabled and as such MCG_TES_P is also enabled.

The values written to the vMCA registers are translated into values that the guest will understand and be able to take action on. This includes translating all system addresses into guest addresses and assigning 0x0 to the MSCOD field of vIA32_MCi_STATUS.

Other than the vIRQ signals that are sent to Dom0, the only signals injected into guests by Xen includes those signals that effect the guest directly and for which there may be some action the guest can take to improve the situation. With this in mind, the only signals currently injected into guests are for Software Recoverable Action Optional (SRAO) and Software Recoverable Action Required (SRAR) type errors. This is to allow the guest a chance to take action on these errors.

While CMCI signals are intended to be limited to those hardware threads on the effected socket the vCPU's don't have a socket abstraction so this signal, if injected into a guest, needs to be sent to all vCPU in the guest. MCE are intended to go to every hardware thread in the system so they will also be sent to every vCPU within a guest as well.

Live Migration

As the issues that brought about a rethinking of the old MCA design were caused specifically by the way the MCA registers where virtualized and how they interacted with live migration, it is appropriate that we discuss how this design will work with live migration.

The choice of MCA registers and configuration settings defined for guests remain the same across migration. Specifically the vIA32_MCG_CAP register will be read only and its field values are well defined. However, to allow for the possibility of future changes that may be made, this register needs to be migrated to ensure it remains unchanged to the guest. The single bank vIA32_MCi_CTL register will be set to all '1's so it is also well defined. vIA32_MCi_STATUS should be cleared when no outstanding errors are present and vIA32_MCi_MISC and vIA32_MCi_ADDR are undefined in this case so they can also be cleared. However, vIA32_MCi_CTL2 is not well defined as it is written by the OS and needs to maintain its value across migration so it must be migrated. Specifically the exception here is vIA32_MCi_CTL2 which is initially cleared for new VM. OS's that choose to enable CMCI signaling for corrected errors will write vIA32_MCi_CTL2[30] to enable CMCI and write a threshold value into vIA32_MCi_CTL2[15:0]. The OS will expect these values to remain the same across migration and so

both vIA32_MCG_CAP and vIA32_MCi_CTL2 will be migrated while the other vMSR will just be initialized at the receiving end.

Notice that there is an open case in the above paragraph; vIA32_MCi_STATUS should be cleared when there are no outstanding errors. We must ensure this is the case if we don't plan to migrate this MSR's value.

Xen will check to see if there are any unaddressed error signals prior to beginning live migration and will abort any in progress live migrations for which an error signal is delivered to the guest.

Dom0 guest

As Dom0 is responsible for managing the Xen system it has a special responsibility to handle system error logging and notification. This responsibility is address by implementing a VIRQ handler to receive signals from the Xen hypervisor on errors and use its resources to log these errors and notify system administrators. The main facility used in Linux systems is the mcelog service while it is FMA (Fault Management Architecture) service in Solaris. Dom0 should use the information it gets through the vIRQ interface to feed its local services rather than its own native MCA routines to ensure it reports accurate information.

Xen supports a hypercall for Dom0 guest to gather information about errors in addition to the vMCA registers. This hypercall, HYPERVISOR_mca(), is the preferred path for Dom0 to get its information. HYPERVISOR_mca() is located in LINUX_KERNEL_SOURCE/linux/drivers/xen/mcelog.c.

As with the Xen hypervisor and the DomUs, Dom0 must also take any needed action needed to keep the system alive.

DomU guest

No special requirements are placed on DomU guests. If the guest implements MCA aware services it will be able to access what appear to be the systems MCA capabilities but will really be those presented to the guest by Xen. This gives the guest a chance to correct some error cases that might otherwise cause the system to be reset. DomU will, on the other hand, not see all of the system MCA information as most of this information is platform specific and has little or nothing to do with a virtualized guest.