```
EDAC - Error Detection And Correction

Written by Doug Thompson <dougthompson@xmission.com>
7 Dec 2005
17 Jul 2007     Updated

(c) Mauro Carvalho Chehab <mchehab@redhat.com>
05 Aug 2009     Nehalem interface

EDAC is maintained and written by:

        Doug Thompson, Dave Jiang, Dave Peterson et al,
        original author: Thayne Harbaugh,

Contact:
        website:        bluesmoke.sourceforge.net
        mailing list:   bluesmoke-devel@lists.sourceforge.net

"bluesmoke" was the name for this device driver when it was "out-of-tree"
and maintained at sourceforge.net.  When it was pushed into 2.6.16 for the
first time, it was renamed to 'EDAC'.

The bluesmoke project at sourceforge.net is now utilized as a 'staging area'
for EDAC development, before it is sent upstream to kernel.org

At the bluesmoke/EDAC project site is a series of quilt patches against
recent kernels, stored in a SVN repository. For easier downloading, there
is also a tarball snapshot available.

============================================================================
EDAC PURPOSE

The 'edac' kernel module goal is to detect and report errors that occur
within the computer system running under linux.

MEMORY

In the initial release, memory Correctable Errors (CE) and Uncorrectable
Errors (UE) are the primary errors being harvested. These types of errors
are harvested by the 'edac_mc' class of device.

Detecting CE events, then harvesting those events and reporting them,
CAN be a predictor of future UE events.  With CE events, the system can
continue to operate, but with less safety. Preventive maintenance and
proactive part replacement of memory DIMMs exhibiting CEs can reduce
the likelihood of the dreaded UE events and system 'panics'.

NON-MEMORY

A new feature for EDAC, the edac_device class of device, was added in
the 2.6.23 version of the kernel.

This new device type allows for non-memory type of ECC hardware detectors
to have their states harvested and presented to userspace via the sysfs
```

interface.

Some architectures have ECC detectors for L1, L2 and L3 caches, along with DMA engines, fabric switches, main data path switches, interconnections, and various other hardware data paths. If the hardware reports it, then a edac_device device probably can be constructed to harvest and present that to userspace.

PCI BUS SCANNING

In addition, PCI Bus Parity and SERR Errors are scanned for on PCI devices in order to determine if errors are occurring on data transfers.

The presence of PCI Parity errors must be examined with a grain of salt. There are several add-in adapters that do NOT follow the PCI specification with regards to Parity generation and reporting. The specification says the vendor should tie the parity status bits to 0 if they do not intend to generate parity. Some vendors do not do this, and thus the parity bit can "float" giving false positives.

In the kernel there is a PCI device attribute located in sysfs that is checked by the EDAC PCI scanning code. If that attribute is set, PCI parity/error scanning is skipped for that device. The attribute is:

```
	broken_parity_status
```

as is located in /sys/devices/pci<XXX>/0000:XX:YY.Z directories for PCI devices.

FUTURE HARDWARE SCANNING

EDAC will have future error detectors that will be integrated with EDAC or added to it, in the following list:

```
	MCE	Machine Check Exception
	MCA	Machine Check Architecture
	NMI	NMI notification of ECC errors
	MSRs 	Machine Specific Register error cases
	and other mechanisms.
```

These errors are usually bus errors, ECC errors, thermal throttling and the like.

===
EDAC VERSIONING

EDAC is composed of a "core" module (edac_core.ko) and several Memory Controller (MC) driver modules. On a given system, the CORE is loaded and one MC driver will be loaded. Both the CORE and the MC driver (or edac_device driver) have individual versions that reflect current release level of their respective modules.

Thus, to "report" on what version a system is running, one must report both the CORE's and the MC driver's versions.

LOADING

If 'edac' was statically linked with the kernel then no loading is necessary. If 'edac' was built as modules then simply modprobe the 'edac' pieces that you need. You should be able to modprobe hardware-specific modules and have the dependencies load the necessary core modules.

Example:

```
$> modprobe amd76x_edac
```

loads both the amd76x_edac.ko memory controller module and the edac_mc.ko core module.

===
EDAC sysfs INTERFACE

EDAC presents a 'sysfs' interface for control, reporting and attribute reporting purposes.

EDAC lives in the /sys/devices/system/edac directory.

Within this directory there currently reside 2 'edac' components:

```
        mc      memory controller(s) system
        pci     PCI control and status system
```

===
Memory Controller (mc) Model

First a background on the memory controller's model abstracted in EDAC. Each 'mc' device controls a set of DIMM memory modules. These modules are laid out in a Chip-Select Row (csrowX) and Channel table (chX). There can be multiple csrows and multiple channels.

Memory controllers allow for several csrows, with 8 csrows being a typical value.
Yet, the actual number of csrows depends on the electrical "loading" of a given motherboard, memory controller and DIMM characteristics.

Dual channels allows for 128 bit data transfers to the CPU from memory. Some newer chipsets allow for more than 2 channels, like Fully Buffered DIMMs (FB-DIMMs). The following example will assume 2 channels:

```
                Channel 0       Channel 1
        ===================================
        csrow0  | DIMM_A0       | DIMM_B0 |
        csrow1  | DIMM_A0       | DIMM_B0 |
        ===================================
```

```
        ===================================
        csrow2  | DIMM_A1         | DIMM_B1 |
        csrow3  | DIMM_A1         | DIMM_B1 |
        ===================================
```

In the above example table there are 4 physical slots on the motherboard for memory DIMMs:

```
        DIMM_A0
        DIMM_B0
        DIMM_A1
        DIMM_B1
```

Labels for these slots are usually silk screened on the motherboard. Slots labeled 'A' are channel 0 in this example. Slots labeled 'B' are channel 1. Notice that there are two csrows possible on a physical DIMM. These csrows are allocated their csrow assignment based on the slot into which the memory DIMM is placed. Thus, when 1 DIMM is placed in each Channel, the csrows cross both DIMMs.

Memory DIMMs come single or dual "ranked". A rank is a populated csrow. Thus, 2 single ranked DIMMs, placed in slots DIMM_A0 and DIMM_B0 above will have 1 csrow, csrow0. csrow1 will be empty. On the other hand, when 2 dual ranked DIMMs are similarly placed, then both csrow0 and csrow1 will be populated. The pattern repeats itself for csrow2 and csrow3.

The representation of the above is reflected in the directory tree in EDAC's sysfs interface. Starting in directory /sys/devices/system/edac/mc each memory controller will be represented by its own 'mcX' directory, where 'X" is the index of the MC.

```
        ..../edac/mc/
                   |
                   |->mc0
                   |->mc1
                   |->mc2
                   ....
```

Under each 'mcX' directory each 'csrowX' is again represented by a 'csrowX', where 'X" is the csrow index:

```
        .../mc/mc0/
                |
                |->csrow0
                |->csrow2
                |->csrow3
                ....
```

Notice that there is no csrow1, which indicates that csrow0 is composed of a single ranked DIMMs. This should also apply in both Channels, in order to have dual-channel mode be operational. Since both csrow2 and csrow3 are populated, this indicates a dual ranked set of DIMMs for channels 0 and 1.

Within each of the 'mcX' and 'csrowX' directories are several
EDAC control and attribute files.

==
'mcX' DIRECTORIES

In 'mcX' directories are EDAC control and attribute files for
this 'X" instance of the memory controllers:

Counter reset control file:

	'reset_counters'

	This write-only control file will zero all the statistical counters
	for UE and CE errors. Zeroing the counters will also reset the timer
	indicating how long since the last counter zero. This is useful
	for computing errors/time. Since the counters are always reset at
	driver initialization time, no module/kernel parameter is available.

	RUN TIME: echo "anything" >/sys/devices/system/edac/mc/mc0/counter_reset

		This resets the counters on memory controller 0

Seconds since last counter reset control file:

	'seconds_since_reset'

	This attribute file displays how many seconds have elapsed since the
	last counter reset. This can be used with the error counters to
	measure error rates.

Memory Controller name attribute file:

	'mc_name'

	This attribute file displays the type of memory controller
	that is being utilized.

Total memory managed by this memory controller attribute file:

	'size_mb'

	This attribute file displays, in count of megabytes, of memory
	that this instance of memory controller manages.

Total Uncorrectable Errors count attribute file:

'ue_count'

This attribute file displays the total count of uncorrectable errors that have occurred on this memory controller. If panic_on_ue is set this counter will not have a chance to increment, since EDAC will panic the system.

Total UE count that had no information attribute fileY:

'ue_noinfo_count'

This attribute file displays the number of UEs that have occurred with no information as to which DIMM slot is having errors.

Total Correctable Errors count attribute file:

'ce_count'

This attribute file displays the total count of correctable errors that have occurred on this memory controller. This count is very important to examine. CEs provide early indications that a DIMM is beginning to fail. This count field should be monitored for non-zero values and report such information to the system administrator.

Total Correctable Errors count attribute file:

'ce_noinfo_count'

This attribute file displays the number of CEs that have occurred wherewith no informations as to which DIMM slot is having errors. Memory is handicapped, but operational, yet no information is available to indicate which slot the failing memory is in. This count field should be also be monitored for non-zero values.

Device Symlink:

'device'

Symlink to the memory controller device.

Sdram memory scrubbing rate:

'sdram_scrub_rate'

Read/Write attribute file that controls memory scrubbing. The scrubbing rate is set by writing a minimum bandwidth in bytes/sec to the attribute file. The rate will be translated to an internal value that gives at least the specified rate.

Reading the file will return the actual scrubbing rate employed.

        If configuration fails or memory scrubbing is not implemented, the value
        of the attribute file will be -1.

==
'csrowX' DIRECTORIES

In the 'csrowX' directories are EDAC control and attribute files for
this 'X" instance of csrow:

Total Uncorrectable Errors count attribute file:

        'ue_count'

        This attribute file displays the total count of uncorrectable
        errors that have occurred on this csrow. If panic_on_ue is set
        this counter will not have a chance to increment, since EDAC
        will panic the system.

Total Correctable Errors count attribute file:

        'ce_count'

        This attribute file displays the total count of correctable
        errors that have occurred on this csrow. This
        count is very important to examine. CEs provide early
        indications that a DIMM is beginning to fail. This count
        field should be monitored for non-zero values and report
        such information to the system administrator.

Total memory managed by this csrow attribute file:

        'size_mb'

        This attribute file displays, in count of megabytes, of memory
        that this csrow contains.

Memory Type attribute file:

        'mem_type'

        This attribute file will display what type of memory is currently
        on this csrow. Normally, either buffered or unbuffered memory.
        Examples:
                Registered-DDR
                Unbuffered-DDR

EDAC Mode of operation attribute file:

```
        'edac_mode'

        This attribute file will display what type of Error detection
        and correction is being utilized.


Device type attribute file:

        'dev_type'

        This attribute file will display what type of DRAM device is
        being utilized on this DIMM.
        Examples:
                x1
                x2
                x4
                x8


Channel 0 CE Count attribute file:

        'ch0_ce_count'

        This attribute file will display the count of CEs on this
        DIMM located in channel 0.


Channel 0 UE Count attribute file:

        'ch0_ue_count'

        This attribute file will display the count of UEs on this
        DIMM located in channel 0.


Channel 0 DIMM Label control file:

        'ch0_dimm_label'

        This control file allows this DIMM to have a label assigned
        to it. With this label in the module, when errors occur
        the output can provide the DIMM label in the system log.
        This becomes vital for panic events to isolate the
        cause of the UE event.

        DIMM Labels must be assigned after booting, with information
        that correctly identifies the physical slot with its
        silk screen label. This information is currently very
        motherboard specific and determination of this information
        must occur in userland at this time.


Channel 1 CE Count attribute file:

        'ch1_ce_count'
```

```
        This attribute file will display the count of CEs on this
        DIMM located in channel 1.


Channel 1 UE Count attribute file:

        'ch1_ue_count'

        This attribute file will display the count of UEs on this
        DIMM located in channel 0.


Channel 1 DIMM Label control file:

        'ch1_dimm_label'

        This control file allows this DIMM to have a label assigned
        to it. With this label in the module, when errors occur
        the output can provide the DIMM label in the system log.
        This becomes vital for panic events to isolate the
        cause of the UE event.

        DIMM Labels must be assigned after booting, with information
        that correctly identifies the physical slot with its
        silk screen label. This information is currently very
        motherboard specific and determination of this information
        must occur in userland at this time.

=============================================================================
SYSTEM LOGGING

If logging for UEs and CEs are enabled then system logs will have
error notices indicating errors that have been detected:

EDAC MC0: CE page 0x283, offset 0xce0, grain 8, syndrome 0x6ec3, row 0,
channel 1 "DIMM_B1": amd76x_edac

EDAC MC0: CE page 0x1e5, offset 0xfb0, grain 8, syndrome 0xb741, row 0,
channel 1 "DIMM_B1": amd76x_edac


The structure of the message is:
        the memory controller                   (MC0)
        Error type                              (CE)
        memory page                             (0x283)
        offset in the page                      (0xce0)
        the byte granularity                    (grain 8)
                or resolution of the error
        the error syndrome                      (0xb741)
        memory row                              (row 0)
        memory channel                          (channel 1)
        DIMM label, if set prior                (DIMM B1
        and then an optional, driver-specific message that may
                have additional information.

Both UEs and CEs with no info will lack all but memory controller,
```

error type, a notice of "no info" and then an optional,
driver-specific error message.

===
PCI Bus Parity Detection

On Header Type 00 devices the primary status is looked at
for any parity error regardless of whether Parity is enabled on the
device. (The spec indicates parity is generated in some cases).
On Header Type 01 bridges, the secondary status register is also
looked at to see if parity occurred on the bus on the other side of
the bridge.

SYSFS CONFIGURATION

Under /sys/devices/system/edac/pci are control and attribute files as follows:

Enable/Disable PCI Parity checking control file:

	'check_pci_parity'

	This control file enables or disables the PCI Bus Parity scanning
	operation. Writing a 1 to this file enables the scanning. Writing
	a 0 to this file disables the scanning.

	Enable:
	echo "1" >/sys/devices/system/edac/pci/check_pci_parity

	Disable:
	echo "0" >/sys/devices/system/edac/pci/check_pci_parity

Parity Count:

	'pci_parity_count'

	This attribute file will display the number of parity errors that
	have been detected.

===
MODULE PARAMETERS

Panic on UE control file:

	'edac_mc_panic_on_ue'

	An uncorrectable error will cause a machine panic. This is usually
	desirable. It is a bad idea to continue when an uncorrectable error
	occurs - it is indeterminate what was uncorrected and the operating
	system context might be so mangled that continuing will lead to further

corruption. If the kernel has MCE configured, then EDAC will never
notice the UE.

LOAD TIME: module/kernel parameter: edac_mc_panic_on_ue=[0|1]

RUN TIME: echo "1" >
/sys/module/edac_core/parameters/edac_mc_panic_on_ue

Log UE control file:

'edac_mc_log_ue'

Generate kernel messages describing uncorrectable errors. These errors
are reported through the system message log system. UE statistics
will be accumulated even when UE logging is disabled.

LOAD TIME: module/kernel parameter: edac_mc_log_ue=[0|1]

RUN TIME: echo "1" > /sys/module/edac_core/parameters/edac_mc_log_ue

Log CE control file:

'edac_mc_log_ce'

Generate kernel messages describing correctable errors. These
errors are reported through the system message log system.
CE statistics will be accumulated even when CE logging is disabled.

LOAD TIME: module/kernel parameter: edac_mc_log_ce=[0|1]

RUN TIME: echo "1" > /sys/module/edac_core/parameters/edac_mc_log_ce

Polling period control file:

'edac_mc_poll_msec'

The time period, in milliseconds, for polling for error information.
Too small a value wastes resources. Too large a value might delay
necessary handling of errors and might loose valuable information for
locating the error. 1000 milliseconds (once each second) is the current
default. Systems which require all the bandwidth they can get, may
increase this.

LOAD TIME: module/kernel parameter: edac_mc_poll_msec=[0|1]

RUN TIME: echo "1000" >
/sys/module/edac_core/parameters/edac_mc_poll_msec

Panic on PCI PARITY Error:

'panic_on_pci_parity'

This control files enables or disables panicking when a parity
error has been detected.

module/kernel parameter: edac_panic_on_pci_pe=[0|1]

Enable:
echo "1" > /sys/module/edac_core/parameters/edac_panic_on_pci_pe

Disable:
echo "0" > /sys/module/edac_core/parameters/edac_panic_on_pci_pe

===

EDAC_DEVICE type of device

In the header file, edac_core.h, there is a series of edac_device structures and APIs for the EDAC_DEVICE.

User space access to an edac_device is through the sysfs interface.

At the location /sys/devices/system/edac (sysfs) new edac_device devices will appear.

There is a three level tree beneath the above 'edac' directory. For example, the 'test_device_edac' device (found at the bluesmoke.sourceforget.net website) installs itself as:

/sys/devices/systm/edac/test-instance

in this directory are various controls, a symlink and one or more 'instance' directorys.

The standard default controls are:

```
log_ce          boolean to log CE events
log_ue          boolean to log UE events
panic_on_ue     boolean to 'panic' the system if an UE is encountered
                (default off, can be set true via startup script)
poll_msec       time period between POLL cycles for events
```

The test_device_edac device adds at least one of its own custom control:

```
test_bits       which in the current test driver does nothing but
                show how it is installed. A ported driver can
                add one or more such controls and/or attributes
                for specific uses.
                One out-of-tree driver uses controls here to allow
                for ERROR INJECTION operations to hardware
                injection registers
```

The symlink points to the 'struct dev' that is registered for this edac_device.

```
INSTANCES

One or more instance directories are present. For the 'test_device_edac' case:

        test-instance0


In this directory there are two default counter attributes, which are totals of
counter in deeper subdirectories.

        ce_count        total of CE events of subdirectories
        ue_count        total of UE events of subdirectories

BLOCKS

At the lowest directory level is the 'block' directory. There can be 0, 1
or more blocks specified in each instance.

        test-block0


In this directory the default attributes are:

        ce_count        which is counter of CE events for this 'block'
                        of hardware being monitored
        ue_count        which is counter of UE events for this 'block'
                        of hardware being monitored


The 'test_device_edac' device adds 4 attributes and 1 control:

        test-block-bits-0       for every POLL cycle this counter
                                is incremented
        test-block-bits-1       every 10 cycles, this counter is bumped once,
                                and test-block-bits-0 is set to 0
        test-block-bits-2       every 100 cycles, this counter is bumped once,
                                and test-block-bits-1 is set to 0
        test-block-bits-3       every 1000 cycles, this counter is bumped once,
                                and test-block-bits-2 is set to 0


        reset-counters          writing ANY thing to this control will
                                reset all the above counters.


Use of the 'test_device_edac' driver should any others to create their own
unique drivers for their hardware systems.

The 'test_device_edac' sample driver is located at the
bluesmoke.sourceforge.net project site for EDAC.

=======================================================================
NEHALEM USAGE OF EDAC APIs

This chapter documents some EXPERIMENTAL mappings for EDAC API to handle
```

```
Nehalem EDAC driver. They will likely be changed on future versions
of the driver.

Due to the way Nehalem exports Memory Controller data, some adjustments
were done at i7core_edac driver. This chapter will cover those differences

1) On Nehalem, there are one Memory Controller per Quick Patch Interconnect
   (QPI). At the driver, the term "socket" means one QPI. This is
   associated with a physical CPU socket.

   Each MC have 3 physical read channels, 3 physical write channels and
   3 logic channels. The driver currenty sees it as just 3 channels.
   Each channel can have up to 3 DIMMs.

   The minimum known unity is DIMMs. There are no information about csrows.
   As EDAC API maps the minimum unity is csrows, the driver sequencially
   maps channel/dimm into different csrows.

   For example, suposing the following layout:
        Ch0 phy rd0, wr0 (0x063f4031): 2 ranks, UDIMMs
          dimm 0 1024 Mb offset: 0, bank: 8, rank: 1, row: 0x4000, col: 0x400
          dimm 1 1024 Mb offset: 4, bank: 8, rank: 1, row: 0x4000, col: 0x400
        Ch1 phy rd1, wr1 (0x063f4031): 2 ranks, UDIMMs
          dimm 0 1024 Mb offset: 0, bank: 8, rank: 1, row: 0x4000, col: 0x400
        Ch2 phy rd3, wr3 (0x063f4031): 2 ranks, UDIMMs
          dimm 0 1024 Mb offset: 0, bank: 8, rank: 1, row: 0x4000, col: 0x400
   The driver will map it as:
        csrow0: channel 0, dimm0
        csrow1: channel 0, dimm1
        csrow2: channel 1, dimm0
        csrow3: channel 2, dimm0

exports one
   DIMM per csrow.

   Each QPI is exported as a different memory controller.

2) Nehalem MC has the hability to generate errors. The driver implements this
   functionality via some error injection nodes:

   For injecting a memory error, there are some sysfs nodes, under
   /sys/devices/system/edac/mc/mc?/:

   inject_addrmatch/*:
      Controls the error injection mask register. It is possible to specify
      several characteristics of the address to match an error code:
         dimm = the affected dimm. Numbers are relative to a channel;
         rank = the memory rank;
         channel = the channel that will generate an error;
         bank = the affected bank;
         page = the page address;
         column (or col) = the address column.
      each of the above values can be set to "any" to match any valid value.

      At driver init, all values are set to any.
```

For example, to generate an error at rank 1 of dimm 2, for any channel,
any bank, any page, any column:

```
echo 2 >/sys/devices/system/edac/mc/mc0/inject_addrmatch/dimm
echo 1 >/sys/devices/system/edac/mc/mc0/inject_addrmatch/rank
```

To return to the default behaviour of matching any, you can do:

```
echo any >/sys/devices/system/edac/mc/mc0/inject_addrmatch/dimm
echo any >/sys/devices/system/edac/mc/mc0/inject_addrmatch/rank
```

inject_eccmask:
specifies what bits will have troubles,

inject_section:
specifies what ECC cache section will get the error:
3 for both
2 for the highest
1 for the lowest

inject_type:
specifies the type of error, being a combination of the following bits:
bit 0 - repeat
bit 1 - ecc
bit 2 - parity

inject_enable starts the error generation when something different than 0 is written.

All inject vars can be read. root permission is needed for write.

Datasheet states that the error will only be generated after a write on an address that matches inject_addrmatch. It seems, however, that reading will also produce an error.

For example, the following code will generate an error for any write access at socket 0, on any DIMM/address on channel 2:

```
echo 2 >/sys/devices/system/edac/mc/mc0/inject_addrmatch/channel
echo 2 >/sys/devices/system/edac/mc/mc0/inject_type
echo 64 >/sys/devices/system/edac/mc/mc0/inject_eccmask
echo 3 >/sys/devices/system/edac/mc/mc0/inject_section
echo 1 >/sys/devices/system/edac/mc/mc0/inject_enable
dd if=/dev/mem of=/dev/null seek=16k bs=4k count=1 >& /dev/null
```

For socket 1, it is needed to replace "mc0" by "mc1" at the above commands.

The generated error message will look like:

```
EDAC MC0: UE row 0, channel-a= 0 channel-b= 0 labels "-": NON_FATAL (addr =
0x0075b980, socket=0, Dimm=0, Channel=2, syndrome=0x00000040, count=1,
Err=8c0000400001009f:4000080482 (read error: read ECC error))
```

3) Nehalem specific Corrected Error memory counters

Nehalem have some registers to count memory errors. The driver uses those registers to report Corrected Errors on devices with Registered Dimms.

However, those counters don't work with Unregistered Dimms. As the chipset offers some counters that also work with UDIMMS (but with a worse level of granularity than the default ones), the driver exposes those registers for UDIMM memories.

They can be read by looking at the contents of all_channel_counts/

```
   $ for i in /sys/devices/system/edac/mc/mc0/all_channel_counts/*; do echo $i;
cat $i; done
        /sys/devices/system/edac/mc/mc0/all_channel_counts/udimm0
        0
        /sys/devices/system/edac/mc/mc0/all_channel_counts/udimm1
        0
        /sys/devices/system/edac/mc/mc0/all_channel_counts/udimm2
        0
```

What happens here is that errors on different csrows, but at the same dimm number will increment the same counter.
So, in this memory mapping:

```
        csrow0: channel 0, dimm0
        csrow1: channel 0, dimm1
        csrow2: channel 1, dimm0
        csrow3: channel 2, dimm0
```

The hardware will increment udimm0 for an error at the first dimm at either csrow0, csrow2 or csrow3;
The hardware will increment udimm1 for an error at the second dimm at either csrow0, csrow2 or csrow3;
The hardware will increment udimm2 for an error at the third dimm at either csrow0, csrow2 or csrow3;

4) Standard error counters

The standard error counters are generated when an mcelog error is received by the driver. Since, with udimm, this is counted by software, it is possible that some errors could be lost. With rdimm's, they displays the contents of the registers