1 Error Strategy

1.1 Purpose:

This is a rough error strategy focused at the Runway interface and how it needs to communicate between the processor, the MC (memory controller) and the IOC (I/O controller). In particular, errors detected and logged in the IOC or in Elroy without need for participation by the RBIB (Runway bus interface block) are not covered in detail here. Various Elroy documents are a better source for that information.

1.2 Philosophy:

The priorities for our error strategy are:
- Error containment
- FRU (field replaceable unit) identification once the system has crashed

If there is a hardware error in the core electronics (not I/O cards or I/O bus), we will HPMC the system as soon as possible to contain the error. An example of this philosophy is in the handling of Runway data parity errors. Even though we could distinguish data destined for an I/O card as a potentially non-critical error, we take the approach that our hardware must be very sick for us to get a Runway data parity error. As a result, we HPMC the machine in all cases of a Runway data parity error. The Runway bus wasn't designed to be error tolerant. However, if our handling of certain error conditions is deemed too severe, software can turn off the signaling aspect of each error individually. Status for the error can still be kept, but the prescribed affect on the system will not occur.

Errors are grouped together consistent with what logging resources they require. The first error in a group will use the logging resource and set a status bit, while additional errors will set overflow status bits, but will not overwrite the logging information. Runway parity errors (address or control or data) share logging resources with Runway cycles that are to memory address that are out-of-range. Correctable memory errors have their own logging resources, as do uncorrectable memory errors. The observance of a BROAD_ERROR on Runway doesn't require logging resources, nor does a memory buffer address/control parity error. All other error conditions only use IOC or Elroy logging resources.

1.3 Assumptions

1. The processor will never issue a DIR_ERROR.
2. The processor will never cause a path error (simultaneous assertion of ADDR_VALID and DATA_VALID).
3. There is no useful information in any fields when the processor does a BROAD_ERROR, except the MID. It is only the fact that a BROAD_ERROR was done (by a processor) that is meaningful to Astro.

4. Processors are timing their data returns (programmable value). This establishes a set limit on how long it can take for data to come back from memory or I/O. However, the processor only logs the address of a timed out return for reads to I/O space (in RS_LOG). A timed out return from memory space doesn't get an address logged in the processor since many of these transactions may be in flight at a given time.
5. For purposes of this error handling section, assume the IOC is configured to return a hard fail response on error, rather than returning -1.
6. The IOC is not timing its data returns - the PCI master, or more likely the driver associated with that master (software timeout), will eventually detect the error condition in a case where a return is never received.
7. Elroy never manufactures bad parity to send to Astro. A PIO read may generate a hard fail response, or -1s, but always with good parity. An inbound write that has bad parity on the PCI bus is thrown away by Elroy. What this means is that if there is ever bad parity for inbound data at the Runway output bus level, the error occurred inside Astro, inside Elroy, or on the rope between the two.

1.4 All possible errors/conditions we care about

The IOC/Elroy specific errors are glossed over big-time. Read the Elroy error handling document for more details. Below is a table summarizing the Runway/memory errors we detect, how we signal them, what we log, and what status bits we set. A more detailed description of the error conditions and how we handle them follows the summary table.

Table 1: Error Summary

Error Condition	Astro Signaling	Astro Info logged	Astro ERROR_STATUS bit
Processor or IOC reads from memory w/uncorrectable error	Assert ADDR_VALID and DATA_VALID (path error) on Runway with the incorrect data, send BROAD_ERROR and put the IOC in fatal mode.	Address, MID, TID, syndrome. The IOC and Elroy may also log some information.	**mem_uncorr_stat**
Processor or IOC reads from memory w/correctable error	Send BROAD_ERROR and put the IOC in fatal mode. Astro will normally not signal memory correctable errors. Signaling is provided for debug purposes.	Address, MID, TID, syndrome	**mem_corr_stat**
Processor or IOC reads from memory with memory buffer addr/cntrl parity error	Send BROAD_ERROR and put the IOC in fatal mode.	None. Logging of the DIMM in error can be done at the board level.	**mem_addr_par_stat**
Astro detects control parity error on Runway	Send BROAD_ERROR and put the IOC in fatal mode.	State of Runway ADDR_DATA bus and control signals during control parity error assertion	**run_ctrl_par_err_stat**

Error Condition	Astro Signaling	Astro Info logged	Astro ERROR_STATUS bit
Astro detects address parity error on Runway	Send BROAD_ERROR and put the IOC in fatal mode.	State of Runway ADDR_DATA bus and control signals during address parity error assertion	**run_addr_par_err_stat**
Astro detects data parity error on Runway (All Runway data cycles are assumed to be for Astro.)	Send BROAD_ERROR and put the IOC in fatal mode. ECC is poisoned if the data is written to memory.	State of Runway ADDR_DATA bus and control signals during data parity error assertion	**run_data_par_err_stat**
Observe BROAD_ERROR transaction on Runway	Put the IOC in fatal mode.	None	**run_broad_err_stat**
Observe a Runway path error	Send BROAD_ERROR and put the IOC in fatal mode.	State of Runway ADDR_DATA bus and control signals during path error assertion	**run_path_err_stat**
Processor (or IOC for a transaction that didn't go through the TLB) reads from or writes to memory space that is out-of-range	Send BROAD_ERROR and put the IOC in fatal mode.	State of Runway ADDR_DATA bus and control signals during ADDR_VALID assertion	**run_mem_range_err_stat**
Processor reads from PA IO space in Astro, things don't go well and the IOC is setup to return hard fail response (rather than –1)	None – processor will timeout. See detail writeup on when a rope goes fatal, Elroy goes fatal, or neither	The IOC and/or Elroy is responsible for logging	None
Proc. writes to PA IO space in Astro and things don't go well	None – data is bit-bucketed. See detail writeup on when Elroy goes fatal, or nothing happens	The IOC and/or Elroy is responsible for logging	None

1.4.1 Errors detected by the memory subsystem

1) A processor or the IOC reads from memory and we have an uncorrectable error. (If the data is supplied via a cache-to-cache copy and there is an uncorrectable error this falls under the Runway data parity error case).
   - Assert both ADDR_VALID and DATA_VALID on Runway for the return cycles that have bad data.
   - Send a BROAD_ERROR transaction on Runway.
   - Put the IOC in fatal mode.
   - Set **mem_uncorr_stat** in the ERROR_STATUS register.
   - Log the address, MID, and TID in MEM_ADDR, and the syndrome in MEM_SYND.
   - The IOC rope interface and Elroy will log some information.
   - The processor will do an HPMC as a result of the BROAD_ERROR.

2) A processor or the IOC reads from memory and we have a correctable error.
   - Set **mem_corr_stat** in the Astro ERROR_STATUS register.
   - Log the address, MID, and TID in MEM_ADDR_CORR, and the syndrome in MEM_SYND_CORR. Note that the syndrome allows the bit in error to be uniquely identified for correctable errors.

- The memory controller will correct the error. Hardware scrubbing of the error is **NOT** done.
- No other signaling will normally occur. However, the generation of a BROAD_ERROR and placing the IOC in fatal mode can be enabled for debug purposes and/or for firmware testing.

3) A processor or the IOC reads from memory and we have a memory buffer address/control parity error.
   - Send a BROAD_ERROR transaction.
   - Put the IOC in fatal mode.
   - Set **mem_addr_par_stat** in the Astro ERROR_STATUS register.
   - Nothing is logged in Astro. A board-level implementation can be done to identify which DIMM had the buffer that detected the address/control parity error

1.4.2 Errors detected by the Runway bus interface

1) Detect a control parity error on the Runway bus (transaction could have been initiated by a processor, or by the IOC).
   - Send a BROAD_ERROR transaction.
   - Put the IOC in fatal mode.
   - Set **run_ctrl_par_err_stat** in ERROR_STATUS.
   - Log the state of the ADDR_DATA bus in the RUN_ADDR or RUN_DATA register (as appropriate) and the control state in the RUN_CTRL register.
   - Complete the transaction internally normally.
   - The processor will do an HPMC as a result of the BROAD_ERROR.

2) Detect an address parity error on the Runway bus.
   - Send a BROAD_ERROR transaction.
   - Put the IOC in fatal mode.
   - Set **run_addr_par_err_stat** in ERROR_STATUS.
   - Log the state of the ADDR_DATA bus in the RUN_ADDR register and the control state in the RUN_CTRL register.
   - Ignore the transaction in all respects except that we will still participate in snooping to keep the coherency response FIFOs in sync. The transaction will go into the RBIB's coherency response FIFO and will go into the read coherency map as a NOP.

3) Detect a data parity error on the Runway bus. (We assume that we are always a sink for data.)
   - Send a BROAD_ERROR transaction.
   - Put the IOC in fatal mode.
   - Set **run_data_par_err_stat** in ERROR_STATUS.
   - Log the state of the ADDR_DATA bus in the RUN_DATA register and the control state in the RUN_CTRL register. In some cases, the address may still be in RUN_ADDR as well.
   - Complete the transaction internally normally.
   - If main memory is the destination we will write bad ECC into the DRAMs.

4) Observe a BROAD_ERROR transaction on Runway generated by a processor or by ourselves.

- Put the IOC in fatal mode.
- Don't log anything except the fact that we saw a BROAD_ERROR (set **run_broad_err** in ERROR_STATUS).

5) Detect a path error on the Runway bus.
   - Send a BROAD_ERROR transaction.
   - Put the IOC in fatal mode.
   - Set **run_path_err_stat** in ERROR_STATUS.
   - Log the state of the ADDR_DATA bus in the RUN_DATA register and the control state in the RUN_CTRL register.
   - Complete the transaction internally normally.
6) A processor reads (or does any other coherent transaction) from memory space that is out-of-range, or the IOC sends a similar bad address out on Runway. This may be due to a bad driver for an I/O card that didn't use the TLB, an incorrectly setup TLB entry, or some hardware failure.
   - Send a BROAD_ERROR transaction.
   - Put the IOC in fatal mode.
   - Set **run_mem_range_err_stat** in ERROR_STATUS.
   - Log the state of the ADDR_DATA bus in the RUN_ADDR register and the control state in the RUN_CTRL register.
   - Ignore the transaction in all respects except that we will still participate in snooping to keep the coherency response FIFOs in sync. The transaction will go into the RBIB's coherency response FIFO and will go into the read coherency map as a NOP.
   - The processor will do an HPMC as a result of the BROAD_ERROR.

7) A processor writes (or does any other non-coherent transaction) to memory space that is out-of-range, or the IOC sends a similar bad address out on Runway. This may be due to a bad driver for an I/O card that didn't use the TLB, an incorrectly setup TLB entry, or some hardware failure.
   - Send a BROAD_ERROR transaction.
   - Put the IOC in fatal mode.
   - Set **run_mem_range_err_stat** in ERROR_STATUS.
   - Log the state of the ADDR_DATA bus in the RUN_ADD register and the control state in the RUN_CTRL register.
   - The write data will go into the bit bucket.
   - The processor will do an HPMC as a result of the BROAD_ERROR.

1.4.3 Errors detected by the IOC or Elroy

1) A processor reads from IO space in Astro and things don't go well or the IOC is already in fatal mode. (Assume the IOC is set to generate a hard fail response rather than return -1).
   - If the only problem is that data gets corrupted inside Astro:
     - The transaction completes normally, but with bad parity propagated to the Runway bus.
     - The processor will detect the parity error and will send a BROAD_ERROR and will HPMC.
   - For all other cases:

- The Elroy or Astro (depending on where things didn't go well) will return a hard fail response.
- The Elroy or the rope may go fatal depending on what bad things happened on the PCI bus, or to the data as it traveled toward the processor. The whole smart/dumb PCI card thing may factor into the choice of fatal mode as well.
- The IOC or Elroy are responsible for logging in these cases.
- The processor will timeout since it didn't get a return. This will cause the processor to do a BROAD_ERROR, and HPMC.
- The processor will have logged the address in RS_LOG.

2) A processor writes to IO space in Astro and things don't go well or the IOC is already in fatal mode.
   - The write will be discarded at the point in Elroy that something is observed as corrupt if address (on the rope) or data (parity kept along the entire path) get corrupted before they reach the PCI bus. The intent will be to never issue the write on PCI if the corruption is noticed soon enough.
   - The IOC or Elroy are responsible for logging.
   - Certain failing conditions will cause the Elroy to go into PCI fatal mode.

3) A processor writes to a card that does infinite retries or a non-existent Elroy or an Elroy that has died.
   - The write transactions will stack up in the command/data FIFO in the IOC rope interface and can begin to flow control the processors.
   - The IOC has a forward progress timer that will eventually trip and drain (throw away) all the transactions for the stalled rope.
   - The rope is put in fatal mode.

1.5 Undetected errors

1) If the IOC receives an unexpected response (a return for a transaction it didn't issue) we will not detect this. With control parity giving us some good protection here, we don't feel like this is an area for concern.

2) We don't flag transactions that we don't support, we just ignore them (but still track coherency for coherent transactions). The belief is that these transactions can't be generated by the hardware (control parity protects us here as well), or we don't need to worry about them.

1.6 Error Registers

1.6.1 Error status and control registers

There are a number of memory and Runway error conditions. Each condition has a bit to represent it. The control, enable, and status registers associated with these conditions are defined below. The error-clearing scheme in ERROR_CONTROL may seem overly complex, but it is intended to provide a bomb-proof way for error information to be logged without ever being lost.

Most other schemes for clearing error status and logging registers have holes for when errors can come in at the "wrong" time and be missed. The algorithm described can probably be implemented in hardware such that there are still holes, so we're assuming the hardware implementation will be done carefully such that no holes exist.

1.6.1.1 Error Control Register

MSB	ERROR_CONTROL Register (address: 0xFF_FED0_8010)	LSB

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
reserved																															
Power On Initialization																															
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																										CE	CL	reserved			
Power On Initialization																															
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0	0

Size: 64-bit only		
Field	**Access**	**Description**
CL	W	Clear logging information.
CE	R/W	Clear enable.

Register 1: ERROR_CONTROL Register

The ERROR_CONTROL register controls the clearing of "stat" and "over" bits in ERROR_STATUS, as well as clearing (or re-enabling the updating of) error logging registers. See the error logging register section to see which registers get cleared versus which ones just get re-enabled for updating. There are two bits in this register, **CE** (clear enable) and **CL** (clear log). **CE** (bit 5 of the register) is a read/write bit. **CL** (bit 4 of the register) is a write-only bit that always returns "0" on reads. The behavior of writing to these bits is only defined for the patterns "1,0" and "0,1". Writing "1's" to both bits, or "0's" to both bits, is unwise. When software desires to read log information (for instance an HPMC occurred and a monarch processor was selected), the following steps are taken:

- Software should set the **CE** bit by writing a "1" to it. After this has been done, hardware will sneak in and clear **CE** if any enabled errors (log_en is set) come in. This ensures no errors are missed.
- Software should read ERROR_STATUS and then all of the logging registers that would be meaningful. Reads of these registers are non-destructive, so they may be read multiple times if desired.
- Software should attempt to clear the ERROR_STATUS register and clear or re-enable for updating the error logging registers by writing a "1" to the **CL** bit. If the hardware has cleared the **CE** bit since software set it, the write to the **CL** bit won't cause any error info to be cleared. Hardware looks at the current state of the **CE** bit as the write to the **CL** bit occurs to make this determination.
- Software should re-read the ERROR_STATUS to see if the clear was successful. If not, software must start over with setting the **CE** bit, reading ERROR_STATUS and log registers,

etc. If the write of “1” to the **CL** bit did work, all of the status and logging information was current and valid, and life is wonderful. It is left as an exercise for the reader to figure out what “the clear was successful” really means.

- Software is now finished.

1.6.1.2 Error Enable Register

MSB — ERROR_ENABLE Register (address: 0xFF_FED0_8018) — LSB

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
reserved																															
Power On Initialization																															
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved							run_path_err_sig_en	mem_addr_par_sig_en	mem_corr_sig_en	mem_uncorr_sig_en	run_broad_err_sig_en	run_mem_range_err_sig_en	run_data_par_err_sig_en	run_addr_par_err_sig_en	run_ctrl_par_err_sig_en	reserved							run_path_err_log_en	mem_addr_par_log_en	mem_corr_log_en	mem_uncorr_log_en	run_broad_err_log_en	run_mem_range_err_log_en	run_data_par_err_log_en	run_addr_par_err_log_en	run_ctrl_par_err_log_en
Power On Initialization																															
X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0	X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0

Size: 64-bit only		
Field	**Access**	**Description**
run_path_err_sig_en	R/W	Enable signaling for Runway path errors. The signaling for this error is generating a BROAD_ERROR transaction on Runway and placing the IOC in fatal mode. The containment of corrupt data **is** tied to signaling being enabled.
mem_addr_par_sig_en	R/W	Enable signaling for memory buffer address/control parity errors. The signaling for this error is generating a BROAD_ERROR transaction on Runway and placing the IOC in fatal mode.
mem_corr_sig_en	R/W	Enable signaling for correctable memory errors. The signaling for this error is generating a BROAD_ERROR transaction on Runway and placing the IOC in fatal mode. This is expected to only be used for debug or firmware test purposes.

Size: 64-bit only		
Field	**Access**	**Description**
mem_uncorr_sig_en	R/W	Enable signaling for uncorrectable memory errors. The signaling for this error is generating a BROAD_ERROR transaction on Runway and placing the IOC in fatal mode. Astro will also signal a path error (simultaneous assertion of ADDR_VALID and DATA_VALID) on RUNWAY for the data cycles in error. The containment of corrupt data **is** tied to signaling being enabled.
run_broad_err_sig_en	R/W	Enable signaling for the observance of a BROAD_ERROR on the Runway bus. The signaling for this error is placing the IOC in fatal mode.
run_mem_range_err_sig_en	R/W	Enable signaling for Runway memory address out-of-range errors. The signaling for this error is generating a BROAD_ERROR transaction on Runway and placing the IOC in fatal mode.
run_data_par_err_sig_en	R/W	Enable signaling for Runway data parity errors. The signaling for this error is generating a BROAD_ERROR transaction on Runway and placing the IOC in fatal mode. In addition, bad ECC will be written to memory for memory writes. The containment of corrupt data **is** tied to signaling being enabled.
run_addr_par_err_sig_en	R/W	Enable signaling for Runway address parity errors. The signaling for this error is generating a BROAD_ERROR transaction on Runway and placing the IOC in fatal mode.
run_ctrl_par_err_sig_en	R/W	Enable signaling for Runway control parity errors. The signaling for this error is generating a BROAD_ERROR transaction on Runway and placing the IOC in fatal mode.
run_path_err_log_en	R/W	Enable logging for Runway path errors. The information that is logged for this error is the state of the Runway ADDR_DATA bus in the RUN_DATA register and the Runway control state in the RUN_CTRL register.
mem_addr_par_log_en	R/W	Enable logging for memory buffer address/control parity errors. Only the ERROR_STATUS bit gets set, no other logging is performed.
mem_corr_log_en	R/W	Enable logging for correctable memory errors. The information that is logged is the address, MID, and TID in MEM_ADDR_CORR, and the syndrome in MEM_SYND_CORR. In addition, this bit is the enable for single-bit detection and correction. Error detection and correction are not performed unless **mem_corr_log_en** is set.
mem_uncorr_log_en	R/W	Enable logging for uncorrectable memory errors. The information that is logged is the address, MID, and TID in MEM_ADDR, and the syndrome in MEM_SYND.
run_broad_err_log_en	R/W	Enable the logging of the observance of a BROAD_ERROR on Runway. Only the ERROR_STATUS bit gets set, no other logging is performed.

Size: 64-bit only		
Field	**Access**	**Description**
run_mem_range_err_log_en	R/W	Enable logging for Runway access to memory that is out-of-range errors. The information that is logged for this error is the state of the Runway ADDR_DATA bus in the RUN_ADDR register and the Runway control state in the RUN_CTRL register.
run_data_par_err_log_en	R/W	Enable logging for Runway data parity errors. The information that is logged for this error is the state of the Runway ADDR_DATA bus in the RUN_DATA register and the Runway control state in the RUN_CTRL register.
run_addr_par_err_log_en	R/W	Enable logging for Runway address parity errors. The information that is logged for this error is the state of the Runway ADDR_DATA bus in the RUN_ADDR register and the Runway control state in the RUN_CTRL register.
run_ctrl_par_err_log_en	R/W	Enable logging for Runway control parity errors. The information that is logged for this error is the state of the Runway ADDR_DATA bus in the RUN_ADDR or RUN_DATA register (depending on when the control parity error occurs) and the Runway control state in the RUN_CTRL register.

Register 2: ERROR_ENABLE Register

The ERROR_ENABLE register independently controls the logging enable and signaling enable of an error condition. An error condition's log_en bit must be set for that error to reflect in the ERROR_STATUS register. What this also implies is that if signaling of an error is desired, logging should be enabled as well. The hardware may make some assumptions about this, so the behavior with signaling enabled but logging disabled is undefined. It should also be noted that disabling or enabling error conditions during normal operation could result in the loss of some error information. This enabling/disabling is assumed to be done by PDH at system initialization time.

1.6.1.3 Error Status Register

ERROR_STATUS Register (address: 0xFF_FED0_8020) (MSB … LSB)

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
reserved																															
Power On Initialization																															
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved							run_path_err_over	mem_addr_par_over	mem_corr_over	mem_uncorr_over	run_broad_err_over	run_mem_range_err_over	run_data_par_err_over	run_addr_par_err_over	run_ctrl_par_err_over	reserved							run_path_err_stat	mem_addr_par_stat	mem_corr_stat	mem_uncorr_stat	run_broad_err_stat	run_mem_range_err_stat	run_data_par_err_stat	run_addr_par_err_stat	run_ctrl_par_err_stat
Power On Initialization																															
X	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0	X	X	X	X	X	X	X	0	0	0	0	0	0	0	0	0

Size: 64-bit only		
Field	**Access**	**Description**
run_path_err_over	R	Detects overflow for Runway path errors.
mem_addr_par_over	R	Detects overflow for memory buffer address/control parity errors.
mem_corr_over	R	Detects overflow for correctable memory errors.
mem_uncorr_over	R	Detects overflow for uncorrectable memory errors.
run_broad_err_over	R	Detects overflow for the observance of a BROAD_ERROR on the Runway bus.
run_mem_range_err_over	R	Detects overflow for Runway memory address out-of-range errors.
run_data_par_err_over	R	Detects overflow for Runway data parity errors.
run_addr_par_err_over	R	Detects overflow for Runway address parity errors.
run_ctrl_par_err_over	R	Detects overflow for Runway control parity errors.
run_path_err_stat	R	Status of Runway path errors.
mem_addr_par_stat	R	Status of memory buffer address/control parity errors.
mem_corr_stat	R	Status of correctable memory errors.
mem_uncorr_stat	R	Status of uncorrectable memory errors
run_broad_err_stat	R	Status of the observance of a BROAD_ERROR on Runway.
run_mem_range_err_stat	R	Status of Runway access to memory that is out-of-range errors.
run_data_par_err_stat	R	Status of Runway data parity errors.
run_addr_par_err_stat	R	Status of Runway address parity errors.
run_ctrl_par_err_stat	R	Status of Runway control parity errors

Register 3: ERROR_STATUS Register

The ERROR_STATUS register contains error status (**stat**) and error overflow (**over**) bits. These bits can be thought of as being in four groups corresponding to the three sets of logging resources inside Astro, plus run_broad_err and mem_addr_par (which don't use any logging resources). The first error in a group sets its "stat" bit, the second error in a group sets its "over" bit. If simultaneous errors were to occur, multiple "stat" bits could get set. How these status and overflow bits are cleared is described in the ERROR_CONTROL register description. The four error groups are:

1. run_*_err errors, excluding run_broad_err. These all use RUN_ADDR, RUN_DATA, and RUN_CTRL. The first occurrence of one of these errors will set the appropriate "stat" bit (i.e. **run_addr_par_err_stat**). The second occurrence of any of these errors will set the appropriate "over" bit (i.e. **run_data_par_err_over**). In this manner, it can always be determined which error the log information corresponds to in a multi-error scenario. Note that simultaneous errors could set multiple "stat" bits, but in such a case, the log information applies to both errors.
2. run_broad_err or mem_addr_par. Since nothing is logged, the first occurrence sets **run_broad_err_stat** or **mem_addr_par_stat**, a second occurrence sets **run_broad_err_over** or **mem_addr_par_over**.
3. mem_uncorr. Since there are dedicated logging resources (MEM_ADDR, MEM_SYND), the first occurrence sets **mem_uncorr_stat**, a second occurrence sets **mem_uncorr_over**.
4. mem_corr. Since there are dedicated logging resources (MEM_ADDR_CORR, MEM_SYND_CORR), the first occurrence sets **mem_corr_stat**, a second occurrence sets **mem_corr_over**.

1.6.2 Error logging registers

1.6.2.1 Runway Control Register

RUN_CTRL Register (address: 0xFF_FED0_8038) (MSB … LSB)

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
reserved																		a_par [3:0]				a_ctl_par	a_mid[2:0]			a_tid[5:0]					
Power On Initialization																															
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved																		d_par[3:0]				d_ctl_par	d_mid[2:0]			d_tid[5:0]					
Power On Initialization																															
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Size: 64-bit only		
Field	**Access**	**Description**
a_par	R	Address parity.
a_ctl_par	R	Control parity for address
a_mid	R	Master ID for address.
a_tid	R	Transaction ID for address.
d_par	R	Data parity.
d_ctl_par	R	Control parity for data
d_mid	R	Master ID for data.
d_tid	R	Transaction ID for data.

Register 4: RUN_CTRL Register

The RUN_CTRL register saves the state of the Runway control signals:

1) TRANS_ID
2) MASTER_ID
3) CTL_PAR
4) AD_PAR

One half of the register saves the state of these signals on address cycles, the other half of the register saves the state of these signals on data cycles. This register updates with every non-idle cycle until an error is detected, then it freezes. Interesting facts about the MASTER_ID are that a MASTER_ID of 0x7 is used by Astro for BROAD_ERRORs it generates. A MASTER_ID of 0x6 is used by the Astro IOC for any transactions it masters. For these IOC mastered transactions, the two most significant bits of the TRANS_ID contain additional information. A pattern of 00 indicates a transaction initiated by the cache (fetch or flush), a pattern of 10 indicates a TLB request, a pattern of 01 indicates a peer-to-peer read transaction, and a pattern of 11 indicates a non-memory targeted write transaction (peer-to-peer, interrupt, etc.).

1.6.2.2 Runway Address Register

MSB	RUN_ADDR Register (address: 0xFF_FED0_8040)																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
run_addr[63:32]																															
Power On Initialization																															
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
run_addr[31:0]																															
Power On Initialization																															
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Size: 64-bit only		
Field	**Access**	**Description**
run_addr	R	Runway address.

Register 5: RUN_ADDR Register

The RUN_ADDR register saves the state of the ADDR_DATA bus on Runway whenever an address cycle is on the bus. This register updates until an error is detected, then it freezes.

1.6.2.3 Runway Data High Register

MSB	RUN_DATA_HIGH Register (address: 0xFF_FED0_8048)																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
run_data[127:96]																															
Power On Initialization																															
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
run_data[95:64]																															
Power On Initialization																															
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Size: 64-bit only		
Field	**Access**	**Description**
run_data	R	Runway data.

Register 6: RUN_DATA_HIGH Register

The RUN_DATA_HIGH register saves the state of the upper 64-bits of turbo-mode data on the ADDR_DATA bus on Runway whenever a data cycle is on the bus. This register updates until an error is detected, then it freezes.

1.6.2.4 Runway Data Low Register

MSB	RUN_DATA_LOW Register (address: 0xFF_FED0_8050)																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
run_data[63:32]																															
Power On Initialization																															
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
run_data[31:0]																															
Power On Initialization																															
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

Size: 64-bit only		
Field	**Access**	**Description**
run_data	R	Runway data.

Register 7: RUN_DATA_LOW Register

The RUN_DATA_LOW register saves the state of the lower 64-bits of turbo-mode data on the ADDR_DATA bus on Runway whenever a data cycle is on the bus. This register updates until an error is detected, then it freezes.

1.6.2.5 Memory Address Register

MEM_ADDR Register (address: 0xFF_FED0_C008)

MSB … LSB

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
reserved																							mid[2:0]			tid[5:0]					
Power On Initialization																															
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved		addr[35:6]																													
Power On Initialization																															
X	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Size: 64-bit only		
Field	**Access**	**Description**
mid	R	Master ID of uncorrectable read.
tid	R	Transaction ID of uncorrectable read.
addr	R	Address of uncorrectable read.

Register 8: MEM_ADDR Register

The MEM_ADDR register stores the MID, TID, and address of an uncorrectable memory error. All implemented bits of the **addr** field reset to 1's (an impossible memory address). This register only updates when an error is detected. The syndrome bits can be used to improve address resolution to 8-bytes.

1.6.2.6 Memory Syndrome Register

MSB	MEM_SYND Register (address: 0xFF_FED1_1440)																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
syndrome7[7:0]								syndrome6[7:0]								syndrome5[7:0]								syndrome4[7:0]							
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
syndrome3[7:0]								syndrome2[7:0]								syndrome1[7:0]								syndrome0[7:0]							
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Size: 64-bit only		
Field	**Access**	**Description**
syndrome7	R	Syndrome byte 7.
syndrome6	R	Syndrome byte 6.
syndrome5	R	Syndrome byte 5.
syndrome4	R	Syndrome byte 4.
syndrome3	R	Syndrome byte 3.
syndrome2	R	Syndrome byte 2.
syndrome1	R	Syndrome byte 1.
syndrome0	R	Syndrome byte 0.

Register 9: MEM_SYND Register

The MEM_SYND register saves the syndrome of a memory read that has an uncorrectable error. There are 8 bits of syndrome per 64-bit quantity, so a whole cache-line is represented. This register only updates when an error is detected. Its reset value is all 0's (the syndrome for no error). The **syndromeN** field is the syndrome from the Nth 64-bit word of a cache-line (**syndrome0** is for the cache line at the lowest address).

1.6.2.7 Memory Address Correctable Register

MSB	MEM_ADDR_CORR Register (address: 0xFF_FED0_C010)																														LSB
63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
reserved																							mid[2:0]			tid[5:0]					
Power On Initialization																															
X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X	X
31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
reserved		addr[35:6]																													
Power On Initialization																															
X	X	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1	1

Size: 64-bit only		
Field	**Access**	**Description**
mid	R	Master ID of correctable read.
tid	R	Transaction ID of correctable read.
addr	R	Address of correctable read.

Register 10: MEM_ADDR_CORR Register

The MEM_ADDR_CORR register stores the MID, TID, and address of a correctable memory error. All implemented bits of the **addr** field reset to 1's (an impossible memory address). This register only updates when an error is detected. The syndrome bits can be used to improve address resolution to 8-bytes.

1.6.2.8 Memory Syndrome Correctable Register

MSB **MEM_SYND_CORR Register (address: 0xFF_FED1_1448)** LSB

63	62	61	60	59	58	57	56	55	54	53	52	51	50	49	48	47	46	45	44	43	42	41	40	39	38	37	36	35	34	33	32
syndrome7[7:0]								syndrome6[7:0]								syndrome5[7:0]								syndrome4[7:0]							
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

31	30	29	28	27	26	25	24	23	22	21	20	19	18	17	16	15	14	13	12	11	10	9	8	7	6	5	4	3	2	1	0
syndrome3[7:0]								syndrome2[7:0]								syndrome1[7:0]								syndrome0[7:0]							
Power On Initialization																															
0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0	0

Size: 64-bit only		
Field	**Access**	**Description**
syndrome7	R	Syndrome byte 7.
syndrome6	R	Syndrome byte 6.
syndrome5	R	Syndrome byte 5.
syndrome4	R	Syndrome byte 4.
syndrome3	R	Syndrome byte 3.
syndrome2	R	Syndrome byte 2.
syndrome1	R	Syndrome byte 1.
syndrome0	R	Syndrome byte 0.

Register 11: MEM_SYND_CORR Register

The MEM_SYND_CORR register saves the syndrome of a memory read that has a correctable error. There are 8 bits of syndrome per 64-bit quantity, so a whole cache-line is represented. This register only updates when an error is detected. Its reset value is all 0’s (the syndrome for no error). The **syndromeN** field is the syndrome from the Nth 64-bit word of a cache-line (**syndrome0** is for the cache line at the lowest address).

1.7 Corner cases

What should happen when a read from memory has an uncorrectable error but the data isn't actually used due to a cache-to-cache copy satisfying the read request? A path error can’t be signaled to a processor. In this case, a BROAD_ERROR will be sent instead. Even though the data wasn’t used, the memory subsystem is very sick, and the entire system should be halted as quickly as possible to prevent further errors from propagating bad data.

1.8 Things I thought I'd mention

- Astro never issues a DIR_ERROR.

- There isn't a one-to-one correspondence between error occurrences that cause us to issue a BROAD_ERROR, and BROAD_ERRORs being issued on Runway. The first error condition will trigger Astro to send a BROAD_ERROR to Runway. Astro will not send any more BROAD_ERRORs to Runway until the error is cleared by the ERROR_CONTROL register.