ACPI for the Armv8 RAS Extensions 1.0

Platform Design Document

Non-confidential

arm

Document number: DEN0085

Contents

Release information 3

Non-Confidential Proprietary Notice 4

1 About this document 5
1.1 Terms and abbreviations 5
1.2 References 5
1.3 Feedback 5

2 ACPI description for Armv8 RAS error nodes 6

3 Arm Error Source Table 7
3.1 Component types 8
3.1.1 Processor structures 9
3.1.2 Memory controller structures 9
3.1.3 Vendor defined structures 10
3.2 Interfaces 10
3.3 Interrupts 11

4 Integration into APEI 12
4.1 Integrating AEST into HEST 12
4.2 Representing RAS error nodes in the BERT 12
4.2.1 CPER Armv8 RAS extension section: Introduction and usage 12

Release information

Date	Version	Changes
2019/Mar/01	1.0	• Alpha release

Non-Confidential Proprietary Notice

1 About this document

1.1 Terms and abbreviations

RAS: Reliability, Availability, Serviceability (see [1])

Term	Meaning
ACPI	The Advanced Configuration and Power Interface specification. This defines a standard for device configuration and power management by an OS
PE	Processing element (see [2]

1.2 References

This section lists publications by Arm and by third parties.

See Arm Infocenter (http://infocenter.arm.com) for access to Arm documentation.

[1] *DDI 0587 Arm® Reliability, Availability, and Serviceability (RAS) Specification Armv8, for the Armv8-A architecture profile*. ARM Ltd.

[2] *DDI 0487 Arm® Architecture Reference Manual ARMv8, for the ARMv8-A architecture profile*. ARM Ltd.

[3] *Advanced Configuration and Power Interface Specification. UEFI Forum*. See http://uefi.org/specifications

[4] *DEN 0049 IO Remapping Table*. ARM Ltd.

[5] *Unified Extensible Firmware Interface Specification. UEFI Forum*. See http://uefi.org/specifications

1.3 Feedback

Arm welcomes feedback on its documentation.

If you have comments on the content of this manual, send an e-mail to errata@arm.com. Give:

- The title (ACPI for the Armv8 RAS Extensions).
- The document ID and version (DEN0085 1.0).
- The page numbers to which your comments apply.
- A concise explanation of your comments.

Arm also welcomes general suggestions for additions and improvements.

2 ACPI description for Armv8 RAS error nodes

This document describes ACPI extensions that enable kernel first RAS handling for systems that employ the Arm RAS extensions. For PEs, this specification covers the Armv8 RAS extension and the Armv8.4 RAS extensions. This specification also covers the RAS system architecture, versions 1.0 and 1.1. See [2] for further details.

3 Arm Error Source Table

Provisional

Opens:

This document proposes we create an Arm specific RAS table, the *A*rm *E*rror *S*ource *T*able (AEST), to represent Arm architected RAS error sources. Some partners have indicated that they would rather see this implemented in APEI, chapter 18 if the ACPI specification [3]. If that were required another approach would be to create error source structures in the HEST table that use the same kind of structures discussed below. This idea is covered in Section 4.1.

Armv8.2 and above RAS error sources are represented in the *A*rm *E*rror *S*ource *T*able (AEST), which is described in Table 3.

Table 3: AEST format

Field	Byte length	Byte offset	Description
Header			Standard ACPI format for header.
Signature	4	0	'AEST' Arm error source table
Length	4	4	Length of table in bytes.
Revision	1	8	For this revision this must be 0.
Checksum	1	9	The entire table must sum to zero.
OEM ID	6	10	OEM ID.
OEM Table ID	8	16	For AEST, the table ID is the manufacture model ID
OEM Revision	4	24	OEM revision of the AEST table for the supplied OEM Table ID.
Creator ID	4	28	The vendor ID of the utility that created the table.
Creator Revision	4	32	The revision of the utility that created the table.
Body			
Array of AEST node	–	36	Array of AEST node structures see Table 4.

The AEST consists of a header, and an array of AEST node entries. The format for the node entries is described in Table 4.

Table 4: AEST node format

Field	Byte Length	Byte Offset	Description
Header			
Type	1	0	Node type: • 0 processor error node • 1 memory error node • 2 vendor defined error node
Length	2	1	Length of structure in bytes
Reserved	1	3	Must be zero
Revision	4	4	0.
Offset to Node specific data	4	8	Offset from the start of the node entry (Type field),to node specific data

Field	Byte Length	Byte Offset	Description
Offset to Node Interface Array	4	12	Offset from the start of the node entry (Type field) to node interface array
Node Interface Array size	4	16	Number of entries in the interface array
Offset to Node Interrupt Array	4	20	Offset from the start of the node entry (Type field) to node interrupt array
Node Interrupt Array size	4	24	Number of entries in the interrupt array
Node generic data			
Timestamp Rate	8	28	If the timestamp extension is implemented, and does not use the timebase of the generic counter, as indicated by ERRFR.TS == 0b10, this field indicates the timestamp frequency in HZ of the counter. Otherwise this field MBZ and the OS must ignore its content.
Timestamp Start Value	8	36	If the timestamp extension is implemented, and does not use the timebase of the generic counter, as indicated by ERRFR.TS == 0b10, this field indicates the initial value of the timestamp at system power on. Otherwise this field MBZ and the OS must ignore its content.
Error injection countdown rate	8	44	If Common Fault Injection Model Extension is supported, as indicated by ERRFR.INJ != 0b00, this field provides the rate at which the Error Generation Counter decrements in HZ. Otherwise this field MBZ and the OS must ignore its content.
Node specific data	–	Offset for node specific data	
Node Interface Array	–	Offset for node interface array	
Node Interrupt Array	–	Offset for node interrupt array	

AEST error node are composed of the following parts:

- A header described in Table 4
- A set of common fields for all error nodes described in Table 4
- A component specific section that associates the error node to a component in the system.
- A section that describes the interfaces associated with an error node.
- A section describing associated interrupts with the error node.

3.1 Component types

Each node entry is associated with a component in the system. The node specific data provides the OS with the information needed to identify this association. Three types of component are supported:

- Processor structures
- Memory controller structures

- Vendor define structures

The tables described in these sections, provide the structure for the Node specific data entry of an AEST node.

Provisional

SMMU or GIC related components are not described in this version. Arm would welcome feedback on this issue.

3.1.1 Processor structures

Processor structures are described in Table 5.

Table 5: processor structure

Field	Byte Length	Byte offset	Description
ACPI processor ID	4	0	Processor ID of node
Cache level	4	8	Level of cache from perspective of chosen processor
Cache type	4	12	Cache type : • 0x0 Data • 0x1 Instruction • 0x2 unified

3.1.2 Memory controller structures

Memory structures are described in Table 6.

Table 6: memory structure format

Field	Byte Length	Byte Offset	Description
Proximity domain	4	0	SRAT proximity domain

Provisional

Proximity domain might be a bit coarse in trying to convey information as to which areas of the physical address space a memory controller covers. Particularly if interleaving is used. Another approach could be a list of memory regions, where each region is described by following:

- physical base address
- size
- stride

An additional data item could be a FRU that identifies any DIMM serviced by the impacted controller.

3.1.3 Vendor defined structures

Vendor defined structures are described in Table 7. An OSPM might just log these, or offer them to vendor specific drivers where appropriate.

Table 7: vendor defined structure format

Field	Byte Length	Byte Offset	Description
Vendor ID	4	0	This identifies the node vendor using the vendor ACPI ID as described in the ACPI ID registry is available at http://www.uefi.org/acpi_id_list
Vendor specific data	4	4	Vendor specific data e.g. to identify this error source.

3.2 Interfaces

Nodes can have a system register or a memory mapped interface. Therefore a node in the AEST might present one or two interface entries in its interface array. The array can be found through the Interface Array offset defined in the node header, see Table 4.

Table 8 describes the format of the interface entries.

Provisional

Error node MISC registers could be logged, or logged and cleared. The latter might be useful if, for example, a MISC register contains a counter that needs to be cleared to stop the error from being signaled again. We could add to these tables some bits to indicate whether for a given record in the node, a MISCx register should be logged, or logged and cleared.

More complex management is hard to represent in an ACPI table, and therefore for cases where this is required, one approach could be that this specification recommends such nodes should be handled firmware first, and therefore should not be exposed in this table.

Arm welcomes feedback on both of these points.

Table 8: AEST node format

Field	Byte Length	Byte Offset	Description
Interface type	1	0	Interface type: • 0x0 – System Register • 0x1 – Memory mapped All other values are reserved
Reserved	3	1	MBZ
Flags	4	4	Bit[0] – Node interface is shared. Note that for processor cache nodes, the sharing is restricted to the processors that share the indicated cache – Bits [31:1] – reserved MBZ
Base Address	8	8	Base address of error group that contains the error node. This address is only valid if interface type is == 0x1
Start error record index	2	16	Zero based index of first standard error record that belongs to this node. Value must lie in the range 0-(N-1) where N is value read from ERRIRDR_EL1 register

Field	Byte Length	Byte Offset	Description
Number of error records	2	22	Number of error records between this & next node. This includes both implemented and unimplemented error records.

3.3 Interrupts

RAS architecture [1] nodes can have three kinds of interrupts associated with them:

- *E*rror *R*ecovery *I*nterrupt (ERI)
- *F*ault *H*andling *I*nterrupt (FHI)
- *C*ritical Error *I*nterrupt (CI)

Of these only the first two types are represented in ACPI. This is because critical error interrupts are meant to be handled by system controllers, rather than application processors under the control of the OS.

Table 9 describes the interrupt structures used to represent node interrupts to the OS. These structures form the entries of the node interrupt array. The array can be found through the Interrupt Array offset defined in the node header, see Table 4.

Table 9: AEST node format

Field	Byte Length	Byte Offset	Description
Interrupt type	1	0	Interrupt type: • 0x0 – Fault Handling Interrupt • 0x1 – Error Recovery Interrupt All other values reserved
Reserved	2	1	MBZ
interrupt Flags	1	3	Bits [31:1]: Must be zero Bit 0: • 0 – Interrupt is edge-triggered • 1 – Interrupt is level-triggered
Interrupt GSIV	4	4	GSIV of Fault handling interrupt, if interrupt is SPI or PPI. Zero if it is not implemented.
ID Node	20	8	IORT ID node if interrupt if MSI based. See [4] for the format of IORT ID nodes. MBZ if interrupt is wire based.

4 Integration into APEI

Two aspects of integration into APEI are briefly considered below:

- Integrating AEST into the HEST.
- Representing the Armv8.2 and beyond RAS extensions in CPER records.

4.1 Integrating AEST into HEST

Provisional

This section is provisional and contingent on deciding whether error nodes can be represented in the AEST, or whether they should be folded into the ACPI core spec, in the APEI chapter.

An alternative to creating the AEST table is to integrate the nodes listed above into the HEST directly. We can do by creating new error source structure, described below in Table 10.

Table 10: HEST error source sub-table

Field	Byte Length	Byte offset	Description
Type	2	0	12 - Armv8 RAS Extension error node
Source Id	2	2	This value serves to uniquely identify this error source against other error sources reported by the platform.
Reserved	4	4	Reserved must be zero. Note in other RAS structures in the HEST these offsets typically have: • A Reserved field • A flags field, that indicates different levels of firmware interaction (FW_FIRST and GHES_ASSIST) • An enabled field Since these sources are meant to be interacted with Kernel first, the fields above seem unnecessary. A possible exception is the enabled field. Arm welcomes feedback on this question.
Node entry	8	–	As described in Table 4.

4.2 Representing RAS error nodes in the BERT

Representing RAS error node contents in BERT will require first expressing them in the CPER. To this end, below we describe proposed extensions to the CPER specification (see Appendix N of [5]).

4.2.1 CPER Armv8 RAS extension section: Introduction and usage

CPER records are comprised of a header, and a set of sections. Each section describes information relevant to error. There is at least one section, but there can be more. The CPER specification provides sections that are fairly generic, such as the generic processor error section, and sections that are more architecture specific, such as the Arm or I64 error sections.

This extension introduces and Arm8 RAS section. The section gathers the content of a RAS error node's registers. The section does not carry any specific information that can associate it with components in the SoC. Instead, it is expected that the section is used alongside other error sections already defined for CPER, such as processor sections, generic and Arm, or Memory sections.

The Armv8 RAS extension section is described by the following GUID:

```
Type: {0xBF32D4D5, 0xB427, 0x4025, {0x84, 0x95, 0x8A, 0x9E, 0x5D, 0x40, 0x30, 0
    xE4}}
```

The contents of the Armv8 RAS Extension section are described in Table 11.

Table 11: Armv8 RAS error record entry

Field	Byte Length	Byte offset	Description
Revision	4	0	0 **Note** This field might be unnecessary as the version filed in the section descriptor might be usable instead. It is not clear in the UEFI specification whether that version field is for the section header format or to be used for the section body format. Arm welcomes feedback on this question.
Number of records	4	4	Number of error records captured in the section
Error Record Array	8	–	Array of Armv8 RAS Error section entries described in Table 12

The format for individual error record entries is described in Table 12.

Table 12: Armv8 RAS error record entry

Field	Byte Length	Byte offset	Description
Error record number	1	0	RAS error record number
RAS extension revision	1	1	Describes the revision of the Arm RAS architecture used for this node. For system component, this takes the following format: • bits[7:4] REVISION field of the ERRDEVARCH register in the RAS specification [1]. • bits[3:0] ARCHVER field of the ERRDEVARCH register in the RAS specification [1]. Note that registers ERR<n>MISC2 and ERR<n>MISC3 are only valid if this value is non zero.
Reserved	7	1	Reserved must be zero

Field	Byte Length	Byte offset	Description
ERR<n>FR	8	8	Content of the Error Record Feature Register
ERR<n>CTLR	8	16	Content of the Error Record Control Register
ERR<n>STATUS	8	24	Content of the Error Record Primary Status Register
ERR<n>ADDR	8	32	Content of the Error Record Address Register
ERR<n>MISC0	8	40	Content of the Content Error Record Miscellaneous Register 0
ERR<n>MISC1	8	48	Content of the Content Error Record Miscellaneous Register 1
ERR<n>MISC2	8	56	Content of the Content Error Record Miscellaneous Register 2
ERR<n>MISC3	8	64	Content of the Content Error Record Miscellaneous Register 3