BKK19-116: RAS on ARM64
Reliability, Availability, and Serviceability (RAS) on ARM64 introduction and status update
Fu Wei <wefu@redhat.com>
AGENDA

1. Brief introduction
   ○ Architecture, RAS Extension, APEI, SDEI
   ○ New features & proposals

2. Prototype
   ○ Firmware First Error Handling
     ■ Overview
     ■ APEI protocol (boot time)
     ■ CperLib (run time)
   ○ BERT(TODO)
   ○ EINJ(some ideas)

3. Status & Plans
Brief introduction
Architecture, RAS Extension, APEI, SDEI
New features & proposals
RAS Extension: Gather HW error info for FW

ESB instruction  
Help to **locate** Error

RAS Extension registers
- Provide the error **info** to FW
- Control the **Interrupt** by FW

ARMv8-A RAS extensions standardize **the interface** between HW and FW

- a mandatory extension to ARMv8.2, **an optional extension to the Armv8.0 and Armv8.1**
APEI (ACPI Platform Error Interfaces)

For last crash (critical error)
- BERT
- HEST

For Storage
- ERST

For Testing
- EINJ

For runtime

Provides a standard way to convey error info from Firmware to OS

For Error info format

CPER
UEFI
SDEI usage in RAS

Software Delegated Exception Interface: An interface between FW & OS, for registering, notifying and servicing system events using SMC/HVC. [SDEI Specification (ARM DEN0054A)]
RAS Extension: New features in v1.1 (optional)

RAS Timestamp Extension
- Provides a standard mechanism for timestamping error records.
- It is implemented if \( \text{ERR}<n>\text{FR.TS} \neq 0b00 \).
- The value of \( \text{ERR}<n>\text{FR.TS} \) defines which timebase is used.

Common Fault Injection Model Extension
- Fakes the detection of an error at a node.
- It is implemented if \( \text{ERR}<n>\text{FR.INJ} \neq 0b00 \).
- The registers are:
  - \( \text{ERR}<n>\text{PFGF} \) (Pseudo-fault Generation Feature)
  - \( \text{ERR}<n>\text{PFGCTL} \) (… Control)
  - \( \text{ERR}<n>\text{PFGCDN} \) (… Countdown)
ARMv8.4 architectural features for RAS

ARMv8.4-RAS

● ARMv8.4-DFE
● RAS System Architecture v1.1
  ○ Simplifications to ERR<n>STATUS
  ○ Additional ERR<n>MISC[2,3] registers
  ○ The optional RAS Common Fault Injection Model Extension.

ARMv8.4-DFE

● DFE: Double Fault Extension
● ARMv8.4-DFE provides two controls:
  ○ SCR_EL3.EASE: External aborts to SError interrupt vector
  ○ SCR_EL3.NMEA: Non-maskable External Aborts
Arm® Reliability, Availability, and Serviceability (RAS) Specification Documentations

The latest version for now is:
DDI 0587C.a

Arm® Reliability, Availability, and Serviceability (RAS)
Specification
Armv8, for the Armv8-A architecture profile

Copyright © 2017, 2018 Arm Limited or its affiliates. All rights reserved.
Document number: DDI 0587C.a

The latest version for now is:
DDI 0487D.a

ARM Architecture Reference Manual
Armv8, for ARMv8-A architecture profile

Copyright © 2012-2018 Arm Limited or its affiliates. All rights reserved.
ARM DDI 0487D.a (201808)

Linaro connect
Barcelona 2019
This document describes ACPI extensions that enable kernel first RAS handling for systems that employ the Arm RAS extensions.

For PEs, this specification covers:

1. Armv8 RAS extension
2. Armv8.4 RAS extension.

It also covers the RAS system architecture v1.0 and v1.1.
ACPI for the Armv8 RAS Extensions 1.0 (proposal-1)

**AEST**: Arm Error Source Table

**AEST node** are composed of the following parts:

- A **header**
- A set of **common** fields for all error nodes (Node generic data)
- A component **specific** section
- A section that describes the **interfaces** associated with an error node.
- A section describing associated **interrupts** with the error node.
Integrating AEST into HEST

Q: Do we have to add a brand new table for the extension?

An alternative way is to integrate the “AEST” nodes into the HEST directly by creating new error source structure, Type: **12-ARMv8 RAS Extension error node**
ACPI for the Armv8 RAS Extensions 1.0 (proposal-3)

The section gathers the content of a RAS extension node’s registers without any specific information that can associate it with components in the SoC, because it is used alongside other error sections already defined for CPER, such as processor sections, generic and Arm, or Memory sections.
Prototype

- Firmware First Error Handling
- BERT
Firmware First Error Handling Overview
MM Secure Partition & Secure Partition Manager

- **MM Secure Partition** implements management functions, runs in S-EL0
- Leverages existing firmware code based on EDK2: *StandaloneMm*
- Minimise code in EL3

- **Secure Partition Manager** in BL31 exports standard ABI to
  - Initialize the partition
  - Delegate SMC requests to the partition
  - For RAS, delegate **RAS error handling** to a MM Secure Partition of RAS
APEI protocol: Data path of updating APEI table
CperLib: Data path of generating the CPER blob

- **EL0**
  - OS: Linux (without KVM)
  - HEST driver
  - SEA handler
  - SEI handler
  - Error IRQ handlers

- **EL1**
  - Firmware
  - Non-secure mem for firmware
  - CperInit

- **EL2**
  - Firmware
  - ApelTest.efi
  - BL32 boot script
  - CperLib Test signal
  - Hardware
    - RAS Extensions
      - Error Record Register
      - ERROR Interrupt
      - GICv3
      - SEA/SEI

- **EL3**
  - Secure RAM (on chip)
  - SDEI dispatcher
  - MM Event Handler

- **S-EL0**
  - Secure Partition Image
  - MM Foundation/Core
  - MM Error handler

- **S-EL1**
  - Firmware
  - BL2 Secure RAM (on chip)
BERT: When catastrophic errors occur (TODO)

- If the whole system has crashed down, BMC or other system controller should generate the error blob and save it.
BERT: After emergency reboot (to be improved)
EINJ: Error Injection with Common Fault Injection Model Extension (some ideas)
Status & Plans

Ongoing development
TODO for Reference Solution
Ongoing development

- **QEMU**
  - sec-uart patch for StandaloneMm debug (will upstream soon)
  - SBSA QEMU

- **ARM-TF**
  - upstream qemu support for StandaloneMm

- **EDK2**
  - Improve and upstream APEI protocol
  - Improve and upstream CperLib code
TODO for Reference Solution

- **QEMU**
  - Test on SBSA QEMU

- **Hardware**
  - Test on a real hardware (ARMv8.2+)

- **Firmware**
  - EDK2:
    - prototype solution of BERT Support in SPI(StandaloneMm) and edk2 BL33 image
    - EINJ implementation

- **Linux**
  - Improve HEST & BERT driver
Thank you

Join Linaro to accelerate deployment of your Arm-based solutions through collaboration

contactus@linaro.org
FYI

More detail for RAS on AArch64
What is RAS? -- Definition

**Reliability**
Continuity, Computation needs be **correct** and **reliable**.

**Availability**
Readiness, System needs to remain available as long as possible.

**Serviceability**
Ability to undergo modifications and repairs, system should provide information to administrator to aid in system servicing.

The RAS architecture primarily cares about the **ERRORs produced from HARDWARE**.
Why do we need RAS? -- Reality

**Inevitability**
Although faults are rare, enterprise systems can be very large. So failures are inevitable.

**Impacts**
Enterprise systems often provide mission-critical services. Any system failure or data corruption impacts the customer’s business and reputation.
Why do we need RAS? -- Importance

So we **inevitably have to maintain** system very well. For reducing the expense for maintenance, we should:

- **Replace only failed parts**
- **Scheduled maintenance** cheaper than unscheduled service outages
History

ECC in memory controllers and I/O RAMs

Machine Check Architecture (MCA)
- A mechanism in which the CPU reports hardware errors to the OS
  - model-specific registers (MSRs)
    - set up machine checking
    - record detected errors
    - the info they contain is CPU specific
- Machine Check Exception (MCE)
  - signals the detection of an uncorrected machine-check error
  - handler collect information about error from MSRs
- Utility: mcelog

PCI-E: Advanced Error Reporting (AER)

Linux kernel
- EDAC (Error Detection and Correction)
  - designed to report and possibly act on hardware errors
  - inspect the hardware directly (system-specific handling and decoding.)
  - only support memory controller and PCI/AGP errors

Firmware (first)
- APEI
- UEFI
Taxonomy of Error in Diagram

Producer

- `Corrupted data`
- Detected by Hardware?
  - Yes
    - Error Detection Code (EDC)
  - No
    - Uncorrected
      - UnContainable (UC)
      - Silent Data Corruptions (SDC)

Corrected by Hardware?
- Yes
  - Error Detection Code (EDC)
  - Corrected Error (CE)
- No
  - Passed to the Consumer
    - Deferred (DE)
    - Signaled
      - Correct Operation can Continue?
        - Yes
          - Latent
        - No
          - Unrecoverable (UEU)
      - Propagated?
        - Yes
          - Unrecoverable (UEU)
        - No
          - Corrected Error (CE)

Consumer

- `Corrupted data`
- Detected by Hardware?
  - Yes
    - Error Detection Code (EDC)
    - Error Correcting Code (ECC)
    - Recorded, try to recover
      - Recoverable (UER)
    - Recorded, try to save context
      - Unrecoverable (UEU)
  - No
    - Uncorrected
      - UnContainable (UC)
      - Silent Data Corruptions (SDC)

Can be recovered?
- Yes
  - Detected uncorrected error (DUE) or (UE)
    - Error Detection Code (EDC)
    - Error Correcting Code (ECC)
    - Recoverable (UER)
- No
  - Corrected Error (CE)
RAS Architecture

Firmware First error handling requires standard interfaces between multiple SW components.
Hardware support for RAS

RAS Extension
- ESB (Error Synchronization Barrier) instructions
- RAS Extension registers
- Corrupted data poisoning

CPU
- ARMv8-A architecture (a mandatory extension to ARMv8.2)
- EL2, EL3, or both
- Virtualization extension or Security extensions or both

GICv3
- Interrupt routing modes
- Private and shared interrupts (PPI/SPI)
- Ability to set an interrupt pending event signaling and delegation
  Interrupt groups/priority
RAS Extension: ESB instruction

- ESB (Error Synchronization Barrier) can be used to synchronize Unrecoverable errors (containable errors).
- Software can determine that:
  - The error was reported as Unrecoverable.
  - The preferred return address of the SEI is an ESB instruction.
  - The software between that ESB and the previous ESB can be isolated.
  - ESB might update:
    - DISR_EL1 / DISR (Deferred Interrupt Status Register)
    - VDISR_EL2 / VDISR (Virtual Deferred Interrupt Status Register)
## RAS Extension: Registers

### System register views

*(in Arm® Architecture Reference Manual)*

- Processor Feature Register
- Memory Model Feature Register
- Error Record Register
  - Error Record ID Register
  - Error Record Select Register
  - [Selected Error Record] Feature Register
  - [...] Control Register
  - [...] Primary Status Register
  - [...] Address Register
  - [...] Miscellaneous Register *[0-3]*
- Deferred Interrupt Status Register
- Hypervisor Configuration Register
- Virtual SError Exception Syndrome Register
- Virtual Deferred Interrupt Status Register
- Secure Configuration Register

### Memory-mapped view

- Error Record Feature Register
- Error Record Control Register
- Error Record Primary Status Register
- Error Record Address Register
- Error Record Miscellaneous Register*[0-3]*
- Pseudo-fault Generation Feature register
- Pseudo-fault Generation Control register
- Pseudo-fault Generation Countdown register
- Error Group Status Register
- Implementation Identification Register
- Error Interrupt Configuration Register*[FHI, ERI, CRI][0-2]*
- Error Interrupt Status Register
- Device Affinity Register
- Device Architecture Register
- Device Configuration Register
- Peripheral Identification Register*[0-4]*
- Component Identification Register*[0-3]*
APEI tables

BERT: Boot Error Record Table
Record fatal errors, then report it in the second boot

CPER (in the Appendix of UEFI spec)
Common Platform Error Record
APEI tables

**HEST: Hardware Error Source Table**
- Key info:
  - **HOW** to get trigger
  - **WHERE** are the error records
  - **HOW** to release records' mem

- For ARM64: **GHES v2**
  - **HOW** to get trigger: Notification Structure
  - **WHERE** are the error records: Error Status Address (GAS: Generic Address Structure)
  - **HOW** to release records' mem: Read Ack Register

For IA-32:
- MCE
- CMC
- NMI

For PCI: AER
- Root Port
- Endpoint
- Bridge

For generic hardware:
- GHES V1/V2 (Generic Hardware Error Source)
APEI tables

**ERTS**: Error Record Serialization Table

**Operation abstract**, provides details necessary to communicate with on-board persistent storage for error recording.

**EINJ**: Error Injection Table

**Operation abstract**, provides a generic interface which OSPM can inject hardware errors to the platform without requiring platform specific software.
1. System boot: BootROM-->BL2-->BL3x
   a. **BL31** initializes SPM, SDEI dispatcher and BL32(MM dispatcher)
   b. **UEFI** (BL33), DXE, UEFI Platform Driver:
      i. query MM partitions by **APEI protocol** for error source info
      ii. MM partitions return error source info back to **UEFI**
      iii. **UEFI** map in and mark error record region as Runtime Services Data Region
      iv. Update/add error source info in **HEST**

2. OS starts running: **HEST driver** scan HEST table and register error handlers by **SDEI**

3. UE occurred, the event will be routed to EL3 (SPM)
4. SPM routes the event to RAS error handler in S-EL0 (MM partitions)
5. MM Foundation (**CperLib**) creates the CPER blobs by the info from RAS Extension
6. SPM notifies **SDEI** to call the corresponding OS registered handler
7. OS gets the CPER blobs by Error Status Address block, process the error, try to recovery.
8. report the error event by **RAS event**
9. **rasdaemon** log error info from RAS event to recorder
APEI protocol: API

Main
- Apei.c
  - **UpdateApei**: updates error source information to the specified APEI(HEST/BERT) Table
- ApeiCommon.c
  - GetApeiTable
  - AcpiTableChecksum

For HEST table:
- ApeiHest.c
  - **UpdateHest**
    - GetErrorSources
    - UpdateGhes

For BERT table (TBD):
- ApeiBert.c
  - **UpdateBert**
    - GetErrorInfo
    - FillBert

```
typedef EFI_STATUS (EFIAPI *EFI_APEI_UPDATE) (IN CONST EFI_APEI_PROTOCOL *This, IN UINT32 Signature);
```
CperLib: API

- **CperInit:**
  parse the error source info, and return the error record address info

  ```c
  EFI_STATUS
  EFIAPI
  CperInit (  
    IN EFI_APEI_ERROR_SOURCE  *ErrorSource,
    IN OUT UINTN *ErrorRecordAddress  
  );
  ```

  The Dynamic Tables module passes the ErrorSource to help CperLib get the memory region info for other APIs, like CperWrite

- **CperWrite:**
  creates a CPER blob at the defined memory region from Section info data

  ```c
  EFI_STATUS
  EFIAPI
  CperWrite (  
    IN SECTIONS_INFO *SectionInfo,
    IN UINTN ErrorRecordAddress  
  );
  ```

  Wrap the given section data into CPER blob and put it into the specific memory region