AMD Reliability Availability Serviceability (RAS) UEFI Architecture Specification

THIS DOCUMENT CONTAINS CONFIDENTIAL INFORMATION THAT COULD BE SUBSTANTIALLY DETRIMENTAL TO THE INTEREST OF AMD THROUGH UNAUTHORIZED USE OR DISCLOSURE.

"Copyright 2013, AMD. All rights reserved. The material in this document constitutes an unpublished work created in 2012. The use of this copyright notice is intended to provide notice that AMD owns a copyright in this unpublished work. The copyright notice is not an admission that publication has occurred. This work contains confidential, proprietary to the reader information and trade secrets of AMD. No part of this document may be used, reproduced, or transmitted in any form or by any means without the prior written permission of AMD."

Advanced Micro Devices

Table of Contents

© AMD Confidential. Reference Copyright Notice on Cover Page ©

© AMD Confidential. Reference Copyright Notice on Cover Page ©

Revision History

Revision	Revision History	Author	Date
0.1	Initial Revision	SJK	04/03/2012
1.0	Production Release	SJK	03/27/2013
1.1	Added new RAS Callout IDs and Setup Config. Values.	SJK	05/09/2013

© AMD Confidential. Reference Copyright Notice on Cover Page ©

Overview

Document Purpose / Scope

The purpose of this document is to outline AMD's Reliability Availability Serviceability (RAS) UEFI Architecture and standardize a series of firmware interfaces between the platform BIOS and the AGESA RAS reference code. These firmware interfaces initialize, configure, and handle RAS (Reliability Availability Serviceability) features supported by the processor and supporting chipset. Adding these RAS interfaces to AGESA increases the physical layer abstraction in the platform BIOS and provide a framework for handling runtime errors.

This specification describes how the AGESA reference code RAS interfaces:

- Describe the overall UEFI RAS architecture and its components
- Enable and configure correctable and uncorrectable errors
- Outline a runtime firmware error handling framework via a SMI, SCI, and APIC interrupt.
- Provide APEI error architecture tables

References

- BIOS and Kernel Developer's Guide (BKDG) for AMD Processors for specific CPU family register settings and operation
- SB900 ABUMI/GPP AER & RAS BIOS Programming Guide
- AMD64 Architecture Programmer's Manual Volume 2: System Programming
- ACPI V5.0 specification
- System Management BIOS Reference Specification V2.7.0
- UEFI Specification V2.3.1
- WHEA Platform Design Guide V2.2

Requirements

Software Components

- AMD Generic Encapsulated Software Architecture (AGESA)

© AMD Confidential. Reference Copyright Notice on Cover Page ©

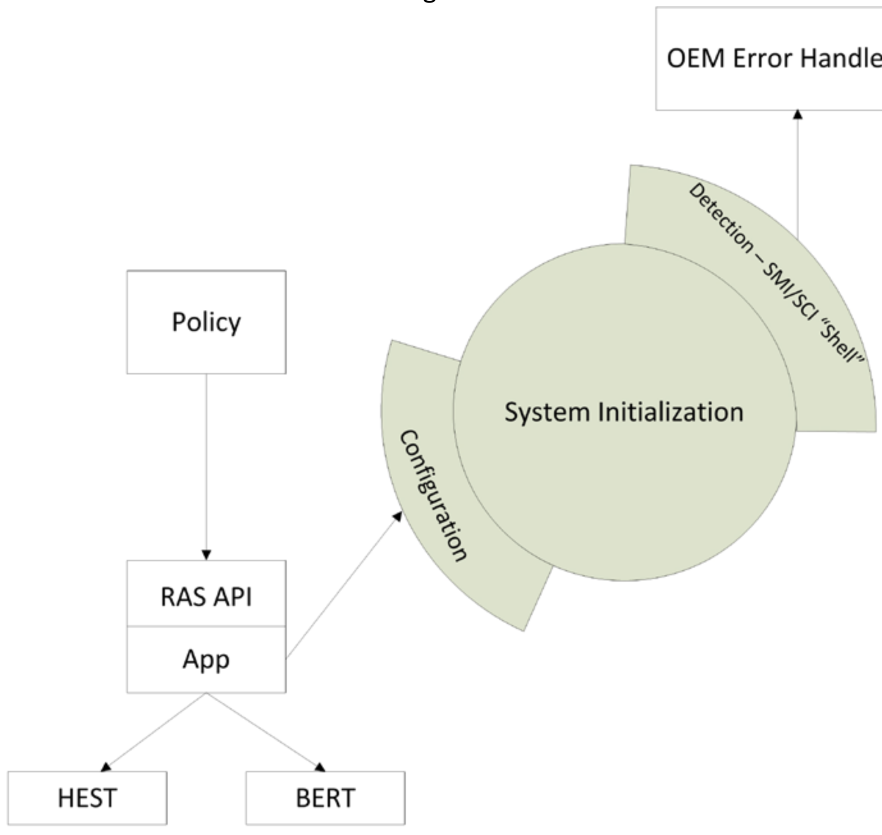
Firmware RAS Architecture

1.0 Firmware RAS Architecture Overview

AMD's firmware RAS architecture consists of several components which reside in both the Pre-EFI (PEI) and Driver Execution Environment (DXE) phases of UEFI POST. Some of these modules/drivers are standalone while others are linked to a partner driver for a complete handling of the error. Each of these driver design types are explained in their respective details sections to follow.

As shown in Figure 1.1, the RAS interface relationship to the overall system initialization body.

Figure 1.1

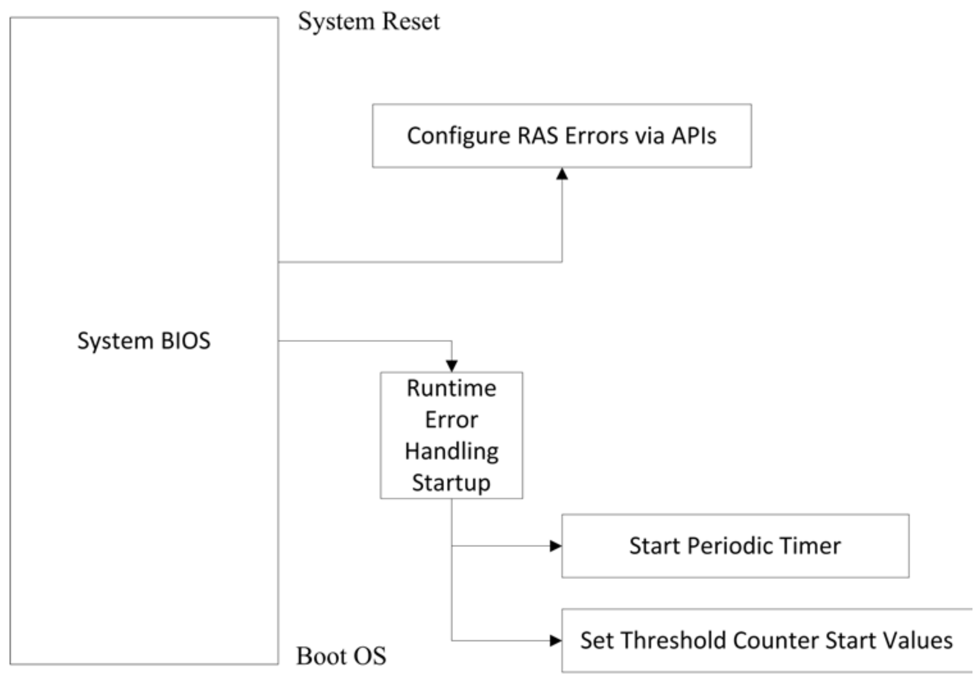


Overall RAS Design

© AMD Confidential. Reference Copyright Notice on Cover Page ©

As shown in Figure 1.2, the RAS error interface initialization is two-fold. First, the RAS error protocol interfaces shall be called by the platform BIOS to enable and configure the respective RAS features in conjunction with selecting the desired RAS event handlers. Second, the platform BIOS begins the runtime error handling startup process which encompasses the launching of SMI handlers and then initializing counter start values. A recommended feature to be implemented by the platform BIOS owner would be to add a respective periodic timer (usually SMI-based) to reset threshold counters when their timeout period has expired.

Figure 1-2


System Reset
Configure RAS Errors via APIs
System BIOS
Runtime Error Handling Startup
Start Periodic Timer
Set Threshold Counter Start Values
Boot OS

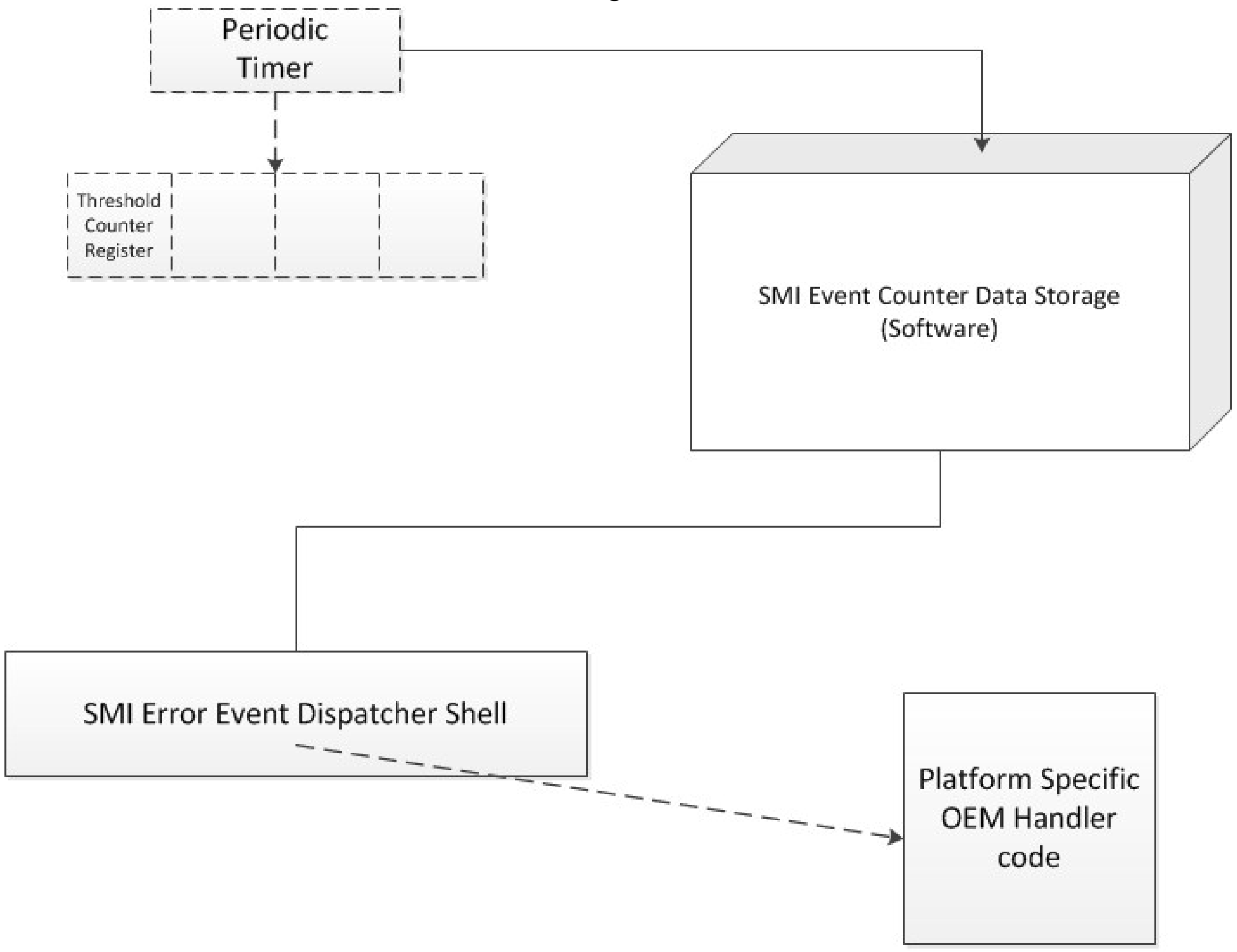
RAS Error Interface Initialization

1.2 Runtime Interface Handler

As shown in Figure 1-3 below, the platform BIOS owner can add an optional runtime interface. The interaction profile which includes a periodic timer present for clearing of threshold counter registers in either silicon or software-based event thresholding. In addition, the figure below outlines a link for creating software thresholding by saving off event counter data in a SMI save space region.

© AMD Confidential. Reference Copyright Notice on Cover Page ©

Figure 1-3


Periodic Timer
Threshold Counter Register
SMI Event Counter Data Storage (Software)
SMI Error Event Dispatcher Shell
Platform Specific OEM Handler code

SMI Interface Architecture

UEFI RAS Architecture

2.0 UEFI RAS Architecture Overview

The UEFI RAS Architecture consists of five UEFI modules. There is one modules belonging in the PEI phase:

- AmdRasResetPeim

© AMD Confidential. Reference Copyright Notice on Cover Page ©

There are four modules /drivers installed in the DXE phase:

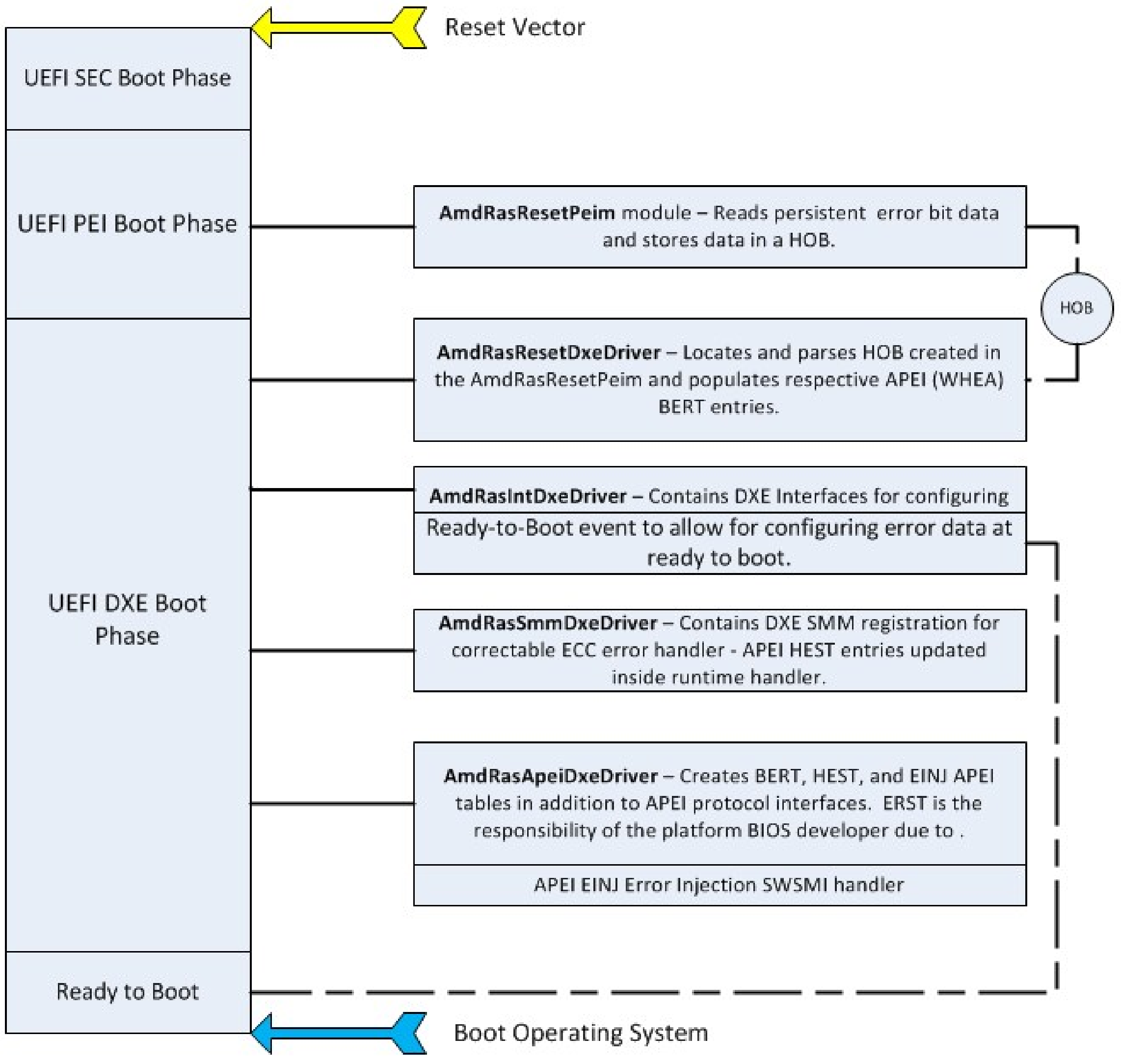
- AmdRasResetDxeDriver
- AmdRasIntDxeDriver
- AmdRasSmmDxeDriver
- AmdRasApeiDxeDriver

Please note that these UEFI modules are not directly part of the AMD Reference (AGESA) core code; however, they will be included as subset of the complete suite of AGESA deliverables under the AGESA UEFI Addendum section. In addition, each module is currently designed to use the EDK I library functions to maintain code compatibility with all BIOS vendors using EDK I libraries. An EDK II version of this UEFI RAS architecture code is planned and will release with a future processor family.

© AMD Confidential. Reference Copyright Notice on Cover Page ©

Diagram 2-1 outlines the various modules designed for AMD's complete RAS UEFI solution and where the individual modules fit in the overall POST process. Bold dashed lines linking modules indicate that a rigid dependency exists between them. A light dashed line indicates that there is no rigid dependency but they have similar meanings in their respective locations. No dashed lines indicate that no dependencies exist with any other modules.

Figure 2-1


Reset Vector
UEFI SEC Boot Phase
UEFI PEI Boot Phase
UEFI DXE Boot Phase
Ready to Boot
Boot Operating System
AmdRasResetPeim module – Reads persistent error bit data and stores data in a HOB.
HOB
AmdRasResetDxeDriver – Locates and parses HOB created in the AmdRasResetPeim and populates respective APEI (WHEA) BERT entries.
AmdRasIntDxeDriver – Contains DXE Interfaces for configuring
Ready-to-Boot event to allow for configuring error data at ready to boot.
AmdRasSmmDxeDriver – Contains DXE SMM registration for correctable ECC error handler - APEI HEST entries updated inside runtime handler.
AmdRasApeiDxeDriver – Creates BERT, HEST, and EINJ APEI tables in addition to APEI protocol interfaces. ERST is the responsibility of the platform BIOS developer due to .
APEI EINJ Error Injection SWSMI handler

AMD RAS UEFI Driver Model

© AMD Confidential. Reference Copyright Notice on Cover Page ©

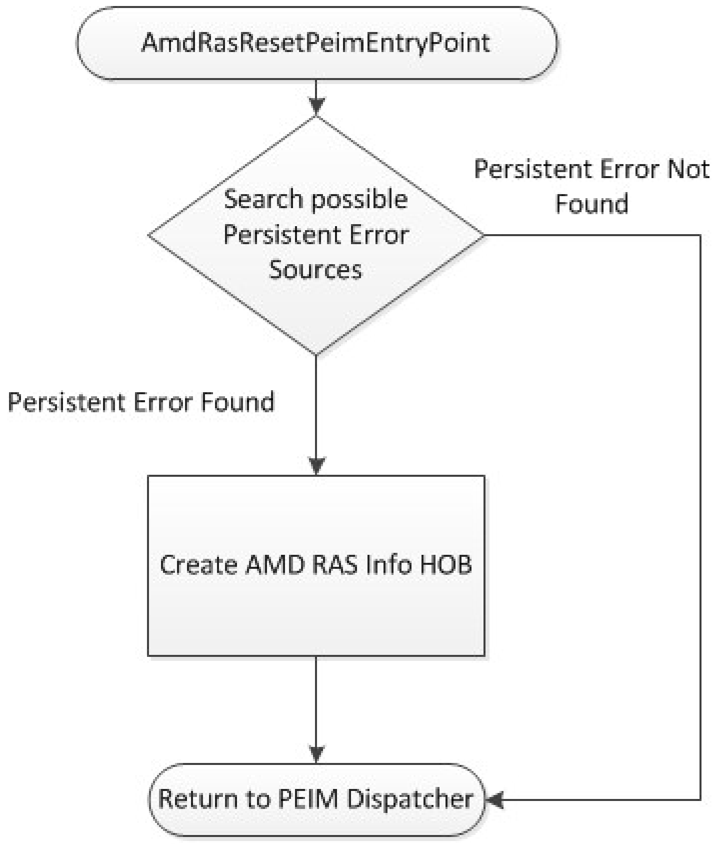
3.0 AmdRasResetPeim Details

The *AmdRasResetPeim* module is a PEI Module (PEIM) that acquires error data from persistent error status bits (sticky bits) in the event that a critical error has occurred and a sync-flood reset condition has taken place. Upon the subsequent reboot, this module will scan sources of persistent error bits and determine if further interrogation is necessary. Examples of RAS errors that would be sourced in this module would be any error that is capable of generating a sync-flood/reset event, such as multi-bit uncorrectable DRAM errors or watchdog timer error. Currently, this module only retrieves node-based DRAM persistent errors; however, it can easily be expanded to cover other persistent error sources. A nested for (..) loop could also be included to support core-based persistent error sourcing.

In the event that an error condition is discovered that requires further processing, then the required error status registers will be stored in a Hand-Off-Block (HOB) to await further processing in the counterpart of this module, *AmdRasResetDxeDriver,* located in the DXE phase when advanced error logging capabilities are available, such as ACPI APEI tables, SMBIOS, and writing to NVRAM is permitted.

© AMD Confidential. Reference Copyright Notice on Cover Page ©

Figure 4-1


AmdRasResetPeimEntryPoint
Search possible Persistent Error Sources
Persistent Error Not Found
Persistent Error Found
Create AMD RAS Info HOB
Return to PEIM Dispatcher

AmdRasResetPeim

This *AmdRasResetPeim* module contains a AmdRasResetPeimEntryPoint function, which is called by the UEFI core PEIM dispatcher on each boot. No other public interfaces are exposed in this PEI module.

4.0 AmdRasResetDxeDriver Details

The *AmdRasReseDxeDriver* module is a DXE driver that is dependent on its PEIM counterpart, the *AmdRasResetPeim* module, for acquiring persistent error data following a sync-flood/reset event. As mentioned in the *AmdRasResetPeim* module details section, a HOB is created and that is where the *AmdRasReseDxeDriver* module takes over. The *AmdRasReseDxeDriver* module first acquires access to the HOB list via the UEFI defined HOB_LIST_GUID. Once the HOB list is found, then the list is searched for the HOB created in the *AmdRasReseDxeDriver* module (AMD_RAS_INFO_HOB) using the AMD_RAS_INFO_HOB_GUID as the locating descriptor.

© AMD Confidential. Reference Copyright Notice on Cover Page ©

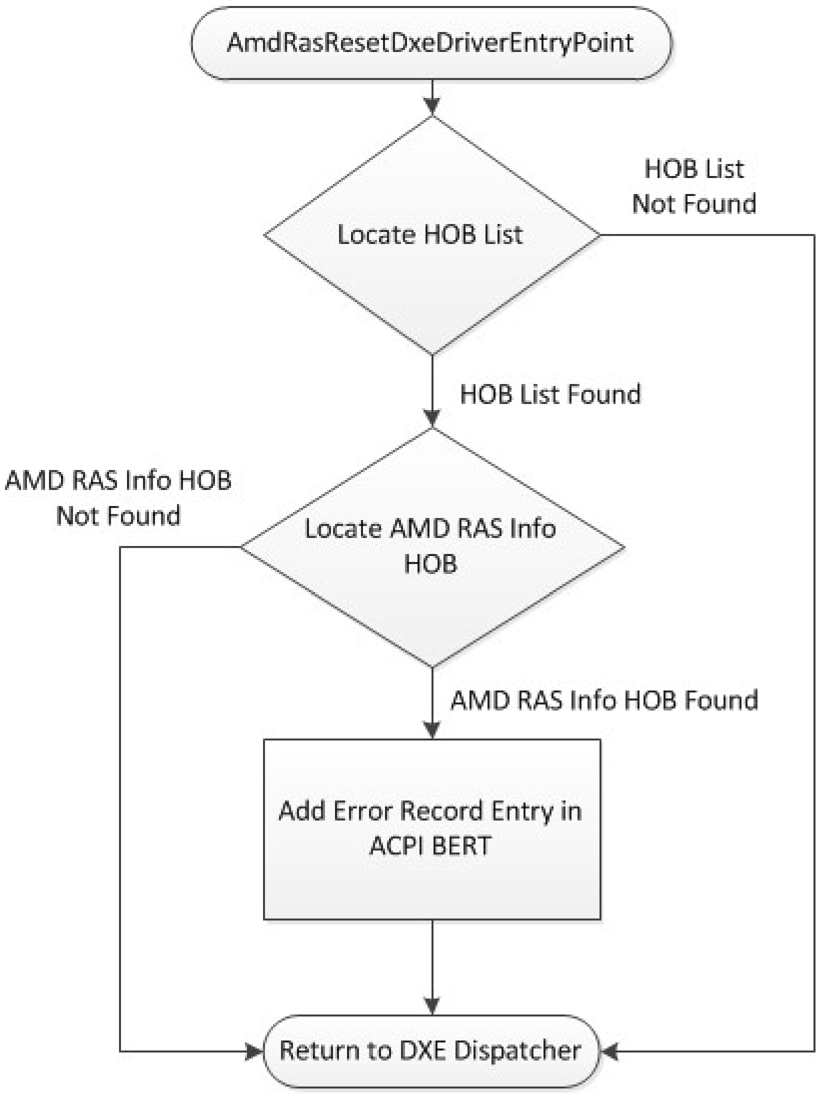
If a valid AMD_RAS_INFO_HOB is found, then the *ErrorPresent* flag is checked to verify if valid error data exists in the HOB data. If so, the HOB data is parsed and the error data is converted to a new ACPI Boot Error Record Table (BERT) entry which, in the case of Windows Server 2008, will log this error in Windows System Event Viewer diagnostic tool with *WHEA-logger* defined as the source.

Examples of these RAS error types that could be sourced in this module are any error that is capable of generating a sync-flood/reset event, such as multi-bit uncorrectable DRAM errors or watchdog timeout error. This module is currently configured to only parse and log DRAM error events. Additional error event types can be incorporated into the existing *switch (RasInfoHob->ErrorCode)* statement. Several empty case statements of common persistent error types have already been listed.

This *AmdRasResetDxeDriver* module contains an AmdRasResetDxeDriverEntryPoint function, which is called by the UEFI core DXE dispatcher on each boot. No other public interfaces are exposed in this DXE driver. The AMD_RAS_INFO_HOB structure is defined in the following source file*: \AGESA\UEFI\Addendum\AmdPlatformRas\Guid\AmRasInfoHob\AmdRasInfoHob.h.*

© AMD Confidential. Reference Copyright Notice on Cover Page ©

Figure 5-1



AmdRasResetDxeDriver

© AMD Confidential. Reference Copyright Notice on Cover Page ©

5.0 AmdRasIntDxeDriver Details

The *AmdRasIntDxeDriver* module is a DXE driver that installs protocol interfaces for configuring RAS error devices that can wait to be configured in the DXE phase. Examples of these RAS errors requiring early configuration are Single-bit and Multi-bit ECC memory errors, Advanced Error Reporting (AER), and CPU core Machine Check Architecture errors. This module does not have any rigid dependencies with any other modules.

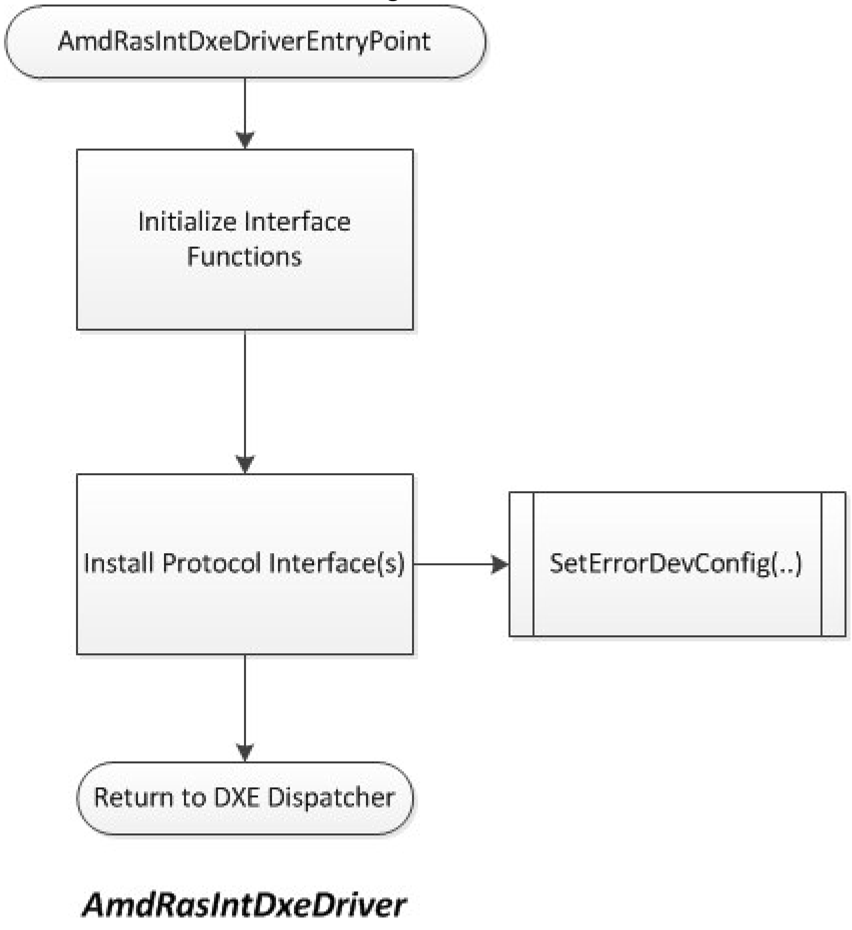
This *AmdRasIntDxeDriver* module contains a AmdRasIntDxeDriverEntryPoint function, which is called by the UEFI core DXE dispatcher on each boot and the primary function will be to initialize function pointers to the following interface(s):

- AmdSetErrorConfigDevice
- AmdSetErrorConfigAll
- AmdGetErrorConfigDevice
- AmdGetErrorConfigAll

Definitions for installed protocol interfaces are provided in Section 5.1 AMD_RAS_DXE_INTERFACE_PROTOCOL.

© AMD Confidential. Reference Copyright Notice on Cover Page ©

Figure 5-1


AmdRasIntDxeDriverEntryPoint
Initialize Interface Functions
Install Protocol Interface(s)
SetErrorDevConfig(..)
Return to DXE Dispatcher
AmdRasIntDxeDriver

5.1 AMD_RAS_DXE_INTERFACE_PROTOCOL

Summary

This protocol is used to configure RAS devices.

GUID

```
#define AMD_RAS_DXE_INTERFACE_PROTOCOL_GUID \
{0xa732cd62, 0x894f, 0x460f, 0x96, 0x12, 0x8b, 0xd, 0xac, \
0x42, 0x17, 0x65}
```

Protocol Interface Structure

```
Typedef struct {
   AMD_GET_ERROR_CONFIG_ALL     AmdGetErrorConfigAll;
```

© AMD Confidential. Reference Copyright Notice on Cover Page ©

```
    AMD_GET_ERROR_CONFIG_DEVICE  AmdGetErrorConfigDevice;
    AMD_SET_ERROR_CONFIG_ALL     AmdSetErrorConfigAll;
    AMD_SET_ERROR_CONFIG_DEVICE  AmdSetErrorConfigDevice;
  } AMD_RAS_DXE_INTERFACE_PROTOCOL;
```

Parameters

AmdGetErrorConfigAll	Retrieves the error configuration for all DXE RAS devices defined. These error devices include those incorporated into the CPU, UNB, and Southbridge. **Note:** The current implementation of this protocol is only a stub and is left up to the platform BIOS vendor to develop this item.
AmdGetErrorConfigDevice	Retrieves the error configuration for a defined DXE RAS device. These error devices include but are not limited to the CPU, UNB, and Southbridge. **Note:** The current implementation of this protocol is only a stub and is left up to the platform BIOS vendor to develop this item.
AmdSetErrorConfigAll	Retrieves the error configuration for all DXE RAS devices defined. These error devices include those incorporated into the CPU, UNB, and Southbridge. **Note:** The current implementation of this protocol is only a stub and is left up to the platform BIOS vendor to develop this item.
AmdSetErrorConfigDevice	Sets the error configuration for a defined DXE RAS device. These error devices include but are not limited to the CPU, UNB, and Southbridge.

5.1.1 AMD_RAS_DXE_INTERFACE_PROTOCOL.AmdGetErrorConfigAll()

Summary

This interface retrieves the error configuration for all defined devices and their accompanying device configuration structure. **Note:** The current implementation of this protocol is only a stub and is left to the platform BIOS vendor to develop this protocol if implementation is required.

© AMD Confidential. Reference Copyright Notice on Cover Page ©

Prototype

```
typedef
EFI_STATUS
(EFIAPI   * AmdGetErrorConfigAll) (
   IN          AMD_RAS_DXE_INTERFACE_PROTOCOL   *This,
   IN OUT      (VOID*)                          *ConfigBuffer
   IN          UINTN                            SourceSize
);
```

Parameters

This	A pointer to the AMD_RAS_INTERFACE_PROTOCOL instance.
ConfigBuffer	The buffer containing the RAS error device configuration table which contains all of the RAS device configurations defined in the table. The configuration table will be processed by this protocol interface all at one time.
SourceSize	The size of the *ConfigBuffer* parameter.

EFI_SUCCESS	The RAS device table was successfully retrieved.
EFI_INVALID_PARAMETER	An invalid parameter was supplied to this protocol.

5.1.2 AMD_RAS_DXE_INTERFACE_PROTOCOL.AmdGetErrorConfigDevice()

Summary

This interface retrieves the error configuration for the provided device ID and accompanying device configuration structure. **Note:** The current implementation of this protocol is only a stub and is left to the platform BIOS vendor to develop this protocol if implementation is required.

Prototype

```
typedef
EFI_STATUS
(EFIAPI   * AmdGetErrorConfigDevice) (
   IN          AMD_RAS_DXE_INTERFACE_PROTOCOL   *This,
   IN          UINTN                            DeviceID,
   IN OUT      (VOID*)                          *ConfigBuffer
);
```

© AMD Confidential. Reference Copyright Notice on Cover Page ©

Parameters

This	A pointer to the AMD_RAS_INTERFACE_PROTOCOL instance.
DeviceID	A unique device ID assigned to the RAS error configuration being processed in the ConfigBuffer parameter.
ConfigBuffer	The buffer containing the RAS error configuration structure which correlates with the DeviceID (param2), and will be processed by this protocol interface.

Status Codes Returned

EFI_SUCCESS	The RAS device was successfully retrieved.
EFI_INVALID_PARAMETER	An invalid parameter was supplied to this protocol.

5.1.3 AMD_RAS_DXE_INTERFACE_PROTOCOL.AmdSetErrorConfigAll()

Summary

This interface executes the error configuration for all defined devices and their accompanying device configuration structures in the device table. **Note:** The current implementation of this protocol is only a stub and is left to the platform BIOS vendor to develop this protocol if implementation is required.

Prototype

```
typedef
EFI_STATUS
(EFIAPI   * AmdSetErrorConfigAll) (
   IN          AMD_RAS_DXE_INTERFACE_PROTOCOL   *This,
   IN OUT      (VOID*)                          *ConfigBuffer
   IN          UINTN                            SourceSize
);
```

Parameters

This	A pointer to the AMD_RAS_INTERFACE_PROTOCOL instance.
ConfigBuffer	The buffer containing the RAS error device configuration table which contains all of the RAS device configurations defined in the table. The configuration

© AMD Confidential. Reference Copyright Notice on Cover Page ©

	table will be processed by this protocol interface all at one time.
SourceSize	The size of the *ConfigBuffer* parameter.

Status Codes Returned

EFI_SUCCESS	The RAS device table was successfully configured.
EFI_INVALID_PARAMETER	An invalid parameter was supplied to this protocol.

5.1.4 AMD_RAS_DXE_INTERFACE_PROTOCOL.AmdSetErrorConfigDevice()

Summary

This interface executes the error configuration for the provided device ID and accompanying device configuration structure.

Prototype

```
typedef
EFI_STATUS
(EFIAPI   * AmdSetErrorConfigDevice) (
   IN          AMD_RAS_DXE_INTERFACE_PROTOCOL   *This,
   IN          UINTN                            DeviceID,
   IN OUT      (VOID*)                          *ConfigBuffer
);
```

Parameters

This	A pointer to the AMD_RAS_INTERFACE_PROTOCOL instance.
DeviceID	A unique device ID assigned to the RAS error configuration being processed in the ConfigBuffer parameter.
ConfigBuffer	The buffer containing the RAS error configuration structure which correlates with the DeviceID (param2), and will be processed by this protocol interface.

Related Definitions

RAS Device Common Structure

```
typedef struct {
   AMD_RAS_DEVICE_ID        DeviceID;
   BOOLEAN                  Enable;
```

© AMD Confidential. Reference Copyright Notice on Cover Page ©

```
    ERROR_MONITOR_TYPE          ErrorMonitorType;
    UINT32                      Size;
} RAS_DEVICE_COMMON;
```

Correctable Memory Error Configuration Structure

```
typedef struct {
    RAS_DEVICE_COMMON           DevCommon;
    BOOLEAN                     EnThreshCounter;
    BOOLEAN                     SetLockedBit;
    UINT32                      ThresshLimit;
    ECC_SYMBOL_SIZE             SymbolSize;
    BOOLEAN                     EnMemParity;
    BOOLEAN                     EnCorrEcc;
} RAS_NB_CORR_MEM_CONFIG;
```

Uncorrectable Memory Error Configuration Structure

```
typedef struct {
    RAS_DEVICE_COMMON           DevCommon;
    BOOLEAN                     SetLockedBit;
    BOOLEAN                     EnMemParity;
    BOOLEAN                     EnUnCorrECC;
    ECC_SYMBOL_SIZE             SymbolSize;
} RAS_NB_UNCORR_MEM_CONFIG;
```

RAS Device ID Enum

```
typedef enum {
    RAS_DXE_ID_INVALID=0,
    CPU_CORE,
    NB_CORR_MEM,
    NB_UNCORR_MEM,
    NB_ONLINE_SPARE,
    NB_CORR_L3_CACHE,
    NB_UNCORR_L3_CACHE,
    NB_CORR_LINKS,
    NB_UNCORR_LINKS,
    UNB_PCIE_AER,
    UNB_ONION,
    UNB_PARITY,
    SB_PCIE_AER,
    SB_PARITY
} AMD_RAS_DEVICE_ID;
```

© AMD Confidential. Reference Copyright Notice on Cover Page ©

Error Monitor Type

```
typedef enum {
   POLLING=0,
   APIC,
   SMI,
   NMI,
   SCI,
   SYNC_FLOOD,
   PCIE_LINK_DISABLE
} ERROR_MONITOR_TYPE;
```

Description

The AmdSetErrorConfigDevice () function is used to configure assigned RAS devices located on the CPU (core and Northbridge), Unified NorthBridge (UNB), and Southbridge (SB).

Status Codes Returned

EFI_SUCCESS	The RAS device was successfully configured.
EFI_OUT_OF_RESOURCES	Not enough storage is available to hold the RAS Interface Protocol structure.
EFI_PROTOCOL_ERROR	There was an error while attempting to install the requested protocol in the list of available protocols.

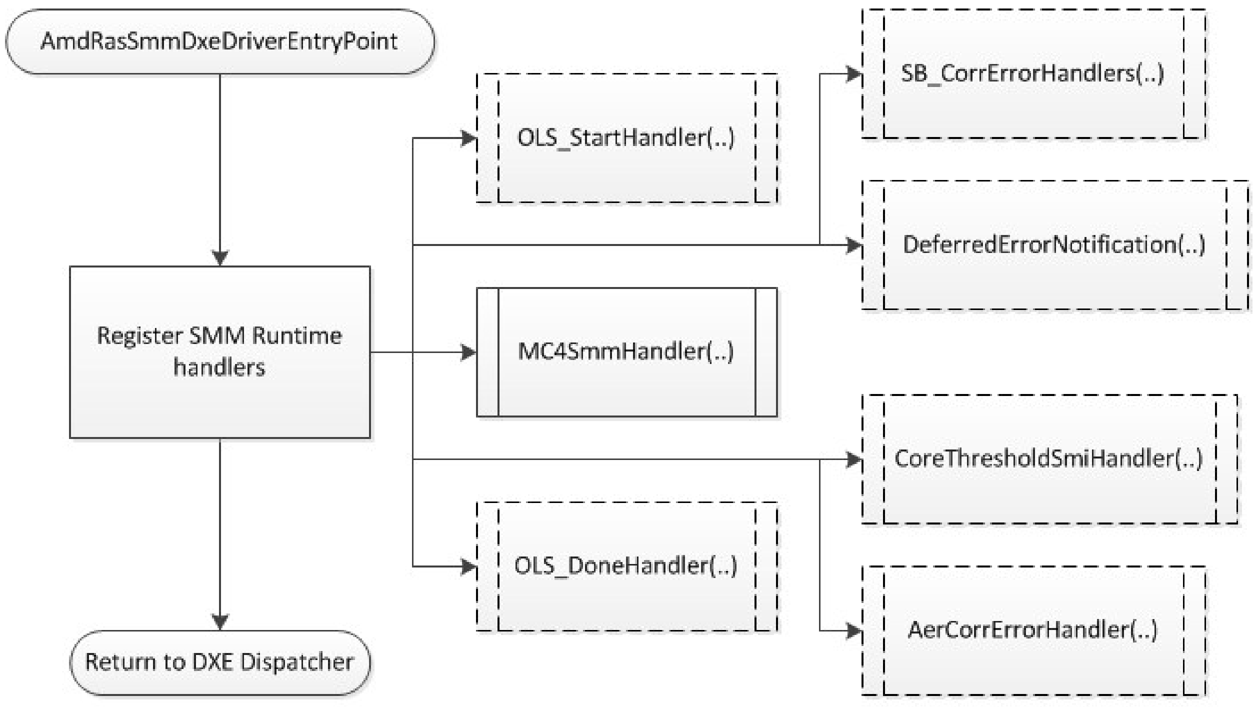
6.0 AmdRasSmmDxeDriver Details

The *AmdRasSmmDxeDriver* module is a UEFI DXE driver that registers SMM runtime handlers to support SMI-based interrupt events. These SMI events originate from multiple sources, most notable being error thresholding. Non-threshold related SMM handlers which can be included in this DXE driver are: Online Spare Memory ECC Counter limit SMM handler, Online Spare Memory swap is complete handler, SouthBridge SMI error handlers, and GNB related SMI error handlers. A software SMI error handler for assigning the proper ACPI EINJ table will be integrated into ACPI APEI DXE driver and not included in *AmdRasSmmDxeDriver*.

This *AmdRasSmmDxeDriver* module contains an AmdRasApeiDxeDriverEntryPoint function, which is called by the UEFI core DXE dispatcher on each boot and no other interface protocols are present.

© AMD Confidential. Reference Copyright Notice on Cover Page ©

Figure 6-1



AmdRasSmmDxeDriver

6.1 AmdRasSmmDxeDriver AmdRasSmmCallout()

The *AmdRasSmmDxeDriver* module contains a callout routine to allow end-users to configure a series of exposed BIOS configuration/Setup options. The callout routines provide a mechanism for the end-user to implement platform specific code to modify predefined code flow paths within the SMM handler registration process and to allow for end-users to add their platform-specific logging and error notification code.

Summary

This callout allows a BIOS vendor to implement platform-specific implementation of BIOS setup configuration; primarily to configure SMI handler capabilities. Due to the platform-specific nature of a portion of the source code in this file, compilation errors may result. Therefore, some code modifications will need to be done by the BIOS vendor in order to integrate the functionality contained within this file to allow for successful compilation and code execution.

© AMD Confidential. Reference Copyright Notice on Cover Page ©

Prototype

```
typedef
EFI_STATUS
(AmdRasSmmCallout) (
   IN          CALLOUT_ID CalloutID,
   IN OUT      (VOID*)    * pConfigBuffer
);
```

Parameters

`CalloutID`	A unique device ID assigned to the RAS SMM callout configuration being processed in the pConfigBuffer parameter.
`pConfigBuffer`	The buffer containing the RAS SMM callout configuration structure which correlates with the DeviceID (param1) and will be processed by this callout.

Related Definitions

```
typedef enum {
   CALLOUT_ID_INVALID=0,
   ECC_HNDLR_CALLOUT_ID,
   FF_CALLOUT_ID,
   ECC_THRESH_COUNTER_CALLOUT_ID,
   ECC_HANDLE_CALLOUT_ID
} CALLOUT_ID;


typedef struct {
   BOOLEAN                  EccHandlerEnable;
   BOOLEAN                  FirmwareFirst;
   UINT16                   DramErrorThreshold;
   UINT16                   RasHandleType;
} SMM_SETUP_CONFIG;
```

7.0 RasLibrary

The RAS library directory provides a number of helper routines that permit the implementation of the AMD UEFI RAS framework without the need to link to proprietary BIOS vendor library functions. These library routines are necessary to complete the RAS framework functionality; however, they are not included in the EDK Library so in an effort to support a universal BIOS vendor design, a series of library routines are provided to fulfill this goal.

© AMD Confidential. Reference Copyright Notice on Cover Page ©

7.1 RasLibrary.AddDevicePath()

Summary

Returns a pointer to a new device path that contains the original device path and the appended device path.

Prototype

```
typedef
EFI_DEVICE_PATH_PROTOCOL*
(AddDevicePath) (
   IN    EFI_DEVICE_PATH_PROTOCOL    *DevicePathLoadedImage,
   IN    EFI_DEVICE_PATH_PROTOCOL    *DevicePathToAdd
);
```

Parameters

DevicePathLoadedImage	A pointer to the loaded image device path.
DevicePathToAdd	A pointer to the device path to add to loaded image device path.

Related Definitions

None

7.2 RasLibrary.RasSmmReadMem8()

Summary

This library routine returns a byte value of the data which resides at the MMIO address requested while in SMM.

Prototype

```
typedef
UINT8
(RasSmmReadMem8) (
   IN    UINT64     Address
);
```

Parameters

`Address`	64-bit MMIO address of the device to be read .

Related Definitions

`None`

7.3 RasLibrary.RasSmmWriteMem8()

Summary

This library routine writes data of a single byte size to the MMIO address provided while in SMM.

Prototype

```
typedef
VOID
(RasSmmWriteMem8) (
   IN    UINT64    Address,
   IN    UINT8     Data
);
```

Parameters

`Address`	64-bit MMIO address of the device to be written.
`Data`	Data byte to be written out to the MMIO address defined in the *Address* parameter.

Related Definitions

`None`

7.4 RasLibrary.RasSmmReadMem32()

Summary

This library routine returns a 32-bit value of the data which resides at the MMIO address requested while in SMM.

© AMD Confidential. Reference Copyright Notice on Cover Page ©

Prototype

```
typedef
UINT32
(RasSmmReadMem32) (
   IN    UINT64     Address
);
```

Parameters

`Address`	64-bit MMIO address of the device to be read .

Related Definitions

None

7.5 RasLibrary.RasSmmWriteMem32()

Summary

This library routine writes data of a 32-bit size to the MMIO address provided while in SMM.

Prototype

```
typedef
VOID
(RasSmmWriteMem32) (
   IN    UINT64     Address,
   IN    UINT32     Data
);
```

Parameters

`Address`	64-bit MMIO address of the device to be written.
`Data`	32-bit data to be written out to the MMIO address defined in the *Address* parameter.

Related Definitions

None

© AMD Confidential. Reference Copyright Notice on Cover Page ©

7.6 RasLibrary.RasSmmReadMem64()

Summary

This library routine returns a 64-bit value of the data which resides at the MMIO address requested while in SMM.

Prototype

```
typedef
UINT64
(RasSmmReadMem64) (
   IN    UINT64    Address
);
```

Parameters

`Address`	64-bit MMIO address of the device to be read .

Related Definitions

None

7.7 RasLibrary.RasSmmWriteMem64()

Summary

This library routine writes data of a 64-bit size to the MMIO address provided while in SMM.

Prototype

```
typedef
VOID
(RasSmmWriteMem64) (
   IN    UINT64    Address,
   IN    UINT64    Data
);
```

Parameters

`Address`	64-bit MMIO address of the device to be written.

© AMD Confidential. Reference Copyright Notice on Cover Page ©

`Data`	64-bit data to be written out to the MMIO address defined in the *Address* parameter.

Related Definitions

`None`

7.8 RasLibrary.RasSmmReadIo8()

Summary

This library routine returns a single byte value of the data which resides at the IO address requested while in SMM.

Prototype

```
typedef
UINT64
(RasSmmReadIo8) (
   IN    UINT64     Address
);
```

Parameters

`Address`	64-bit IO address of the device to be read.

Related Definitions

`None`

7.9 RasLibrary.RasSmmWriteIo8()

Summary

This library routine writes data of a single byte size to the IO address provided while in SMM.

Prototype

```
typedef
VOID
```

© AMD Confidential. Reference Copyright Notice on Cover Page ©

```
(RasSmmWriteIo8) (
   IN     UINT64     Address,
   IN     UINT8      Data
);
```

Parameters

`Address`	64-bit MMIO address of the device to be written.
`Data`	8-bit data to be written out to the IO address defined in the *Address* parameter.

Related Definitions

None

8.0 AmdRasApeiDxeDriver Details

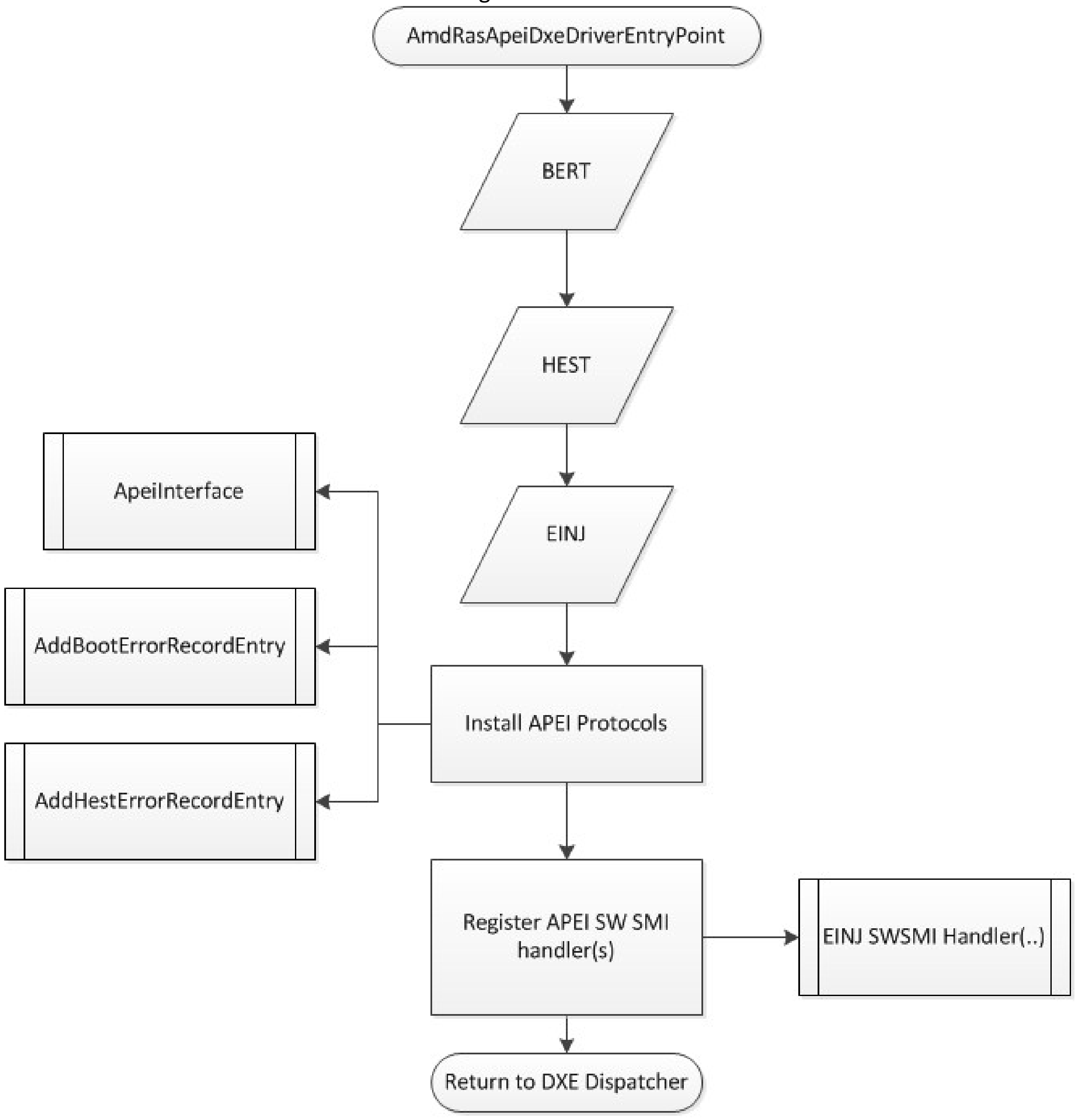
The *AmdRasApeiDxeDriver* module is a DXE driver that builds complete ACPI Platform Error Interface (APEI) tables for the BERT – Boot Error Record Table, HEST – Hardware Error Serialization Table, and EINJ – Error Injection Table and a stubbed out version of the ERST – Error Record Serialization Table and places them in reserved system memory from which the OS can publish. The Error Record Serialization Table (ERST) is only a provided in a stubbed out version because this table is highly platform specific and; therefore, would make a completed AMD-provided version of this table superfluous. In addition, please note that the BIOS vendor using these APEI tables will need to update the OEM ID, OEM Table ID, and OEM Revision prior to completing POST as these parameters are supplied with AMD-defined values. In addition, this DXE module will also install any interface protocols outlined below as well as registering a Software SMI handler to support the EINJ error injection mechanism from a test application. Software SMI events are an integral part of the ACPI EINJ table operation. A test application generates a software SMI with a value that corresponds to one assigned to the error injection procedure. The software SMI handler updates the error injection register values to perform the respective error injection operation.

This *AmdRasApeiDxeDriver* module will contain a AmdRasSmmDxeDriverEntryPoint function, which is called by the UEFI core DXE dispatcher on each boot and the primary function will be to register a Software SMI handler for proper selection of the EINJ table, and initialize function pointers to following interface(s):

- AmdApeiInterface
- AddBootErrorRecordEntry
- AddHestErrorRecordEntry

© AMD Confidential. Reference Copyright Notice on Cover Page ©

Figure 2-8



AmdRasApeiDxeDriver

© AMD Confidential. Reference Copyright Notice on Cover Page ©

8.1 AMD_RAS_APEI PROTOCOL

Summary

This protocol is used to accessing the RAS APEI interfaces.

GUID

```
#define AMD_RAS_APEI_PROTOCOL_GUID \
{0xe9dbcc60, 0x8f93, 0x47ed, 0x84, 0x78, 0x46, 0x78, 0xf1, \
0x9f, 0x73, 0x4a}
```

Protocol Interface Structure

```
Typedef struct {
   AMD_APEI_INTERFACE                AmdApeiInterface;
   AMD_ADD_BOOT_ERROR_RECORD_ENTRY   AddBootErrorRecordEntry;
   AMD_ADD_HEST_ERROR_RECORD_ENTRY   AddHestErrorSourceEntry;
} AMD_RAS_APEI_PROTOCOL;
```

Parameters

AmdApeiInterface	Gets a pointer to the three supplied APEI tables (BERT, EINJ, and HEST).
AddBootErrorRecordEntry	Adds a new record entry to the BERT.
AddHestErrorSourceEntry	Adds a new record entry to the HEST.

8.2 AMD_RAS_APEI_PROTOCOL.AmdApeiInterface

Summary

This interface maintains pointers to the APEI tables loaded into Boot Services reserved memory region.

Prototype

```
typedef
EFI_STATUS
(EFIAPI   * AmdApeiInterface) (
```

© AMD Confidential. Reference Copyright Notice on Cover Page ©

```
    IN OUT      APEI_DRIVER_PRIVATE_DATA    *ApeiPrivData
    );
```

Parameters

ApeiPrivData	A pointer to the ACPI APEI table structure.

Related Definitions

APEI Table Interface Structure

```
typedef struct {
   VOID                        *ErrorLogAddressRange;
   VOID                        *PersistentStoreBuffer;
   APEI_BERT_ACPI_TABLE        *ApeiBertTbl;
   APEI_HEST_ACPI_TABLE        *ApeiHestTbl;
   APEI_EINJ_ACPI_TABLE        *ApeiEinjTbl;
   APEI_ERST_ACPI_TABLE        *ApeiErstTbl;
} APEI_DRIVER_PRIVATE_DATA;
```

Description

The AmdApeiInterface structure is used to retrieve the ACPI APEI tables via the APEI_DRIVER_PRIVATE_DATA structure. This structure contains pointers to each of the four ACPI APEI tables that are published via the APEI DXE driver.

Status Codes Returned

EFI_SUCCESS	The APEI Tables Interface structure was located successfully.
EFI_NOT_FOUND	Could not locate the requested APEI Interface Protocol structure.

8.3 AMD_RAS_APEI_PROTOCOL.AddBootErrorRecordEntry()

Summary

This interface adds a new record entry to the Boot Error Record Table.

Prototype

© AMD Confidential. Reference Copyright Notice on Cover Page ©

```
typedef
EFI_STATUS
(EFIAPI   * AddBootErrorRecordEntry) (
   IN          UINT8      *ErrorRecord,
   IN          UINTN      RecordLen,
   IN          UINT8      ErrorType,
   IN          UINT8      SeverityType
);
```

Parameters

ErrorRecord	A pointer to the error data to be added to the BERT. This error data can be either of type Raw or Generic.
RecordLen	The size in bytes of the ***ErrorRecood*** data structure (param 1).
ErrorType	The error type to be logged in the BERT entry, Raw or Generic.
SeverityType	The error severity level used in setting the block status bitmap in the error region table.

Related Definitions

ErrorType Defines

```
#define Error_Type_Raw                 1
#define Error_Type_Generic             2
```

Error SeverityType Defines

```
#define ERR_SEVERITY_FATAL             1
#define ERR_SEVERITY_CORRECTED         2
```

```
typedef struct {
  GEN_ERR_DATA_ENTRY          GenErrorDataEntry;
  PLATFORM_MEM_ERR_SEC        MemErrorSection;
} BERT_GENERIC_MEM_ERR_ENTRY;
```

Generic Error Status Block - WHEA Platform Design Guide V2.2 Table 4-13

```
typedef struct {
  UINT32                      BlockStatus;
  UINT32                      RawDataOffset;
  UINT32                      RawDataLen;
  UINT32                      DataLen;
  UINT32                      ErrSeverity;
} GENERIC_ERR_STS_BLK;
```

© AMD Confidential. Reference Copyright Notice on Cover Page ©

Generic Error Data Entry – WHEA Platform Design Guide V2.2 Table 4-14

```
typedef struct {
    ERR_DATA_TYPE_DEF       SectionType;
    UINT32                  ErrorSeverity;
    UINT16                  Revision;
    UINT8                   ValidBits;
    UINT8                   Flags;
    UINT32                  ErrDataLen;
    UINT8                   FRU_ID[16];
    UINT8                   FRU_Text[20];
} GEN_ERR_DATA_ENTRY;
```

Error Data Type (GUID) for Boot Error Region Table

```
typedef struct {
    UINT32                  ErrTypeGUIDDword0;
    UINT16                  ErrTypeGUIDWord0;
    UINT16                  ErrTypeGUIDWord1;
    UINT8                   ErrTypeGUIDByte0;
    UINT8                   ErrTypeGUIDByte1;
    UINT8                   ErrTypeGUIDByte2;
    UINT8                   ErrTypeGUIDByte3;
    UINT8                   ErrTypeGUIDByte4;
    UINT8                   ErrTypeGUIDByte5;
    UINT8                   ErrTypeGUIDByte6;
    UINT8                   ErrTypeGUIDByte7;
} ERR_DATA_TYPE_DEF;
```

UEFI 2.3.1 Spec, Appendix N, Table 245, Platform Memory Error Section

```
typedef struct {
    UINT64                  ValidBits;
    UINT64                  ErrStatus;
    UINT64                  PhyAddr;
    UINT64                  PhyAddrMask;
    UINT16                  Node;
    UINT16                  Card;
    UINT16                  Module;
    UINT16                  Bank;
    UINT16                  Device;
    UINT16                  Row;
    UINT16                  Column;
    UINT16                  BitPosition;
```

© AMD Confidential. Reference Copyright Notice on Cover Page ©

```
        UINT64                  RequestorID;
        UINT64                  ResponderID;
        UINT64                  TargetID;
        UINT8                   MemErrType;
        UINT8                   Reserved;
        UIN16                   RankNum;
        UINT16                  CardHandle;
        UINT16                  ModuleHandle;
    } PLATFORM_MEM_ERR_SEC;
```

Description

The AddBootErrorRecordEntry () function is used to add a new record to an existing APEI Boot Error Record Table (BERT). The error type declared for the added BERT table entry can be either defined as Raw or Generic. Generic error data follows the format outlined in Table 4-14 of the WHEA Platform Design Guide V2.2.

Status Codes Returned

EFI_SUCCESS	The BERT entry was completed successfully.
EFI_OUT_OF_RESOURCES	Not enough memory is available to hold the new BERT entry.

8.4 AMD_RAS_APEI_PROTOCOL.AddHestErrorRecordEntry()

Summary

This interface adds a new record entry to the Hardware Error Source Table.

Prototype

```
typedef
EFI_STATUS
(EFIAPI   * AddHestErrorRecordEntry) (
   IN          UINT8      *ErrorRecord,
   IN          UINT32     RecordLen
);
```

Parameters

ErrorRecord	A pointer to the error source data to be added to the HEST table. This error data can be either of type Raw or Generic.
RecordLen	The size in bytes of the ***ErrorRecood*** data structure (param 1).

Related Definitions

```
typedef struct {
   GEN_HW_ERR_SOURCE          UncorrGenErrSource;
   GEN_HW_ERR_SOURCE          CorrGenErrSource;
} ERROR_SOURCE_STRUCT;
```

WHEA Platform Design Guide V2.2 Table 4-12 Generic Hardware Error Source

```
typedef struct {
   UINT16                     GenSourceType;
   UINT16                     SourceID;
   UINT16                     RelatedSrcID;
   UINT8                      Flags;
   UINT8                      Enabled;
   UINT32                     NumRecPreAlloc;
   UINT32                     MaxSection;
   UINT32                     MaxRawDataLen;
   EFI_ACPI_3_0_GAS           RegisterRegion;
   HW_ERR_NOTIFY              NotificationStruct;
   UINT32                     ErrStsBlockLen;
} GEN_HW_ERR_SOURCE;
```

ACPI 5.0 Table 5-26 Generic Address Space Definition

```
typedef struct {
   UINT8                      AddressSpaceID;
   UINT8                      RegisterBitWidth
   UINT8                      RegisterBitOffset;
   UINT8                      AccessSize;
   UINT64                     Address;
} UEFI_ACPI_5_0_GAS;
```

Hardware Error Notification - WHEA Platform Design Guide V2.2 Table 4-15

```
typedef struct {
   UINT8                      NotifiyType;
   UINT8                      ErrNotifyLen;
   UINT16                     ConfigWrite;
   UINT32                     PollInterval;
```

© AMD Confidential. Reference Copyright Notice on Cover Page ©

```
    UINT32                  Vector;
    UINT32                  SwitchPollingThreshVal;
    UINT32                  SwitchPollingThreshWindow;
    UINT32                  EnThresholdVal;
    UINT32                  EnThresholdWindow;
} HW_ERROR_NOTIFY;
```

See **Related Definitions in Section 8.3** for Generic Hardware Error structure definitions.

Description

The AddHestErrorRecordEntry () interface is used to add a new record to an existing APEI Hardware Error Source Table (HEST). The error format in Param(1) must comply with Error Source Descriptor structures outlined in Table 4-1 of the WHEA Platform Design Guide V2.2 or follow Generic error source outlined in Table 4-12 of the WHEA Platform Design Guide V2.2.

Status Codes Returned

EFI_SUCCESS	The HEST entry was completed successfully.
EFI_OUT_OF_RESOURCES	Not enough memory is available to hold the new BERT entry.

© AMD Confidential. Reference Copyright Notice on Cover Page ©