**# Title:**

Error Injection (EINJ) version 2

**# Status:**

Draft

**# Document:**

ACPI Specification Version 6.5

**# License:**

SPDX-License-Identifier: CC-BY-4.0

**# Submitter:**

* Yi Cao, Google
* Harb Abdulhamid, Ampere Computing
* Tyler Baicar, Ampere Computing
* Thanu Rangarajan, Arm
* Samer El-Haj\_Mahmoud, Arm
* TianoCore Community (<https://www.tianocore.org>)

# # Summary of the change

This ECR introduces EINJ version 2 (EINJv2), which extends the ACPI EINJ capability to inject more advanced error syndrome.

This will cover arbitrary error bit patterns in multiple components (e.g. x4 and x8 DRAM devices).

# # Benefits of the change

This is beneficial for platforms that rely on more advanced error correction coding (ECC) schemes (e.g. symbol ECC based on Reed-Solomon encoding).

Given the observation of the increasing number of uncorrectable memory errors in the fleet and potential silent data corruption caused by undetected memory errors, it becomes critical to verify detection and correction coverage of error correction code (ECC) in memory controllers.

On mainstream platforms, memory transactions are performed on the unit of a 512-bit cache line. Additional bits are attached to a cache line for error detection and correction (ECC). All the data and ECC bits of a cache line are distributed evenly into multiple x4 or x8 DRAM chips. Since these DRAM chips operate independently of each other, our goal here is to test the detection and correction coverage on all possible error patterns a mal-functioning single x4 or x8 DRAM chip would produce. As a reference, the state-of-the-art ECC today is single-device-data-correction (SDDC) which detects and corrects 100% errors from one single x4 DRAM device.

The problem with existing EINJ definition is that the exact DRAM device and the error pattern are not specifiable in the data structure and are usually hardcoded in the error injection functions behind the EINJ interface. Here we propose to add a new EINJ injection action to enable injecting arbitrary error patterns to memory devices.

Note that this has been generalized to potentially apply to other component errors (e.g. Processor and PCIe errors).

# **# Impact of the change**

Existing OSPM implementations that support EINJ must be extended to supporting these new EINJ commands and capabilities. Existing EINJ tools and scripts will need to be updated to take advantage of these new features. Backwards compatibility will be maintained to continue to support older version of EINJ software and scripts.

# # Detailed description of the change [normative updates]

Delta from ACPI 6.4

* Changes in **yellow**
* Insertions in **green**
* Removals in **~~red~~**
* References that need fixup in blue

# 18.6 Error Injection

This section outlines an ACPI table mechanism, called EINJ, which allows for a generic interface mechanism through which OSPM can inject hardware errors to the platform without requiring platform specific OSPM level software.

The primary goal of this mechanism is to support testing of OSPM error handling stack by enabling the injection of hardware errors. Through this capability OSPM is able to implement a simple interface for diagnostic and validation

of errors handling on the system.

## 18.6.1 Error Injection Table (EINJ)

The Error Injection (EINJ) table provides a generic interface mechanism through which OSPM can inject hardware errors to the platform without requiring platform specific OSPM software. System firmware is responsible for building this table, which is made up of Injection Instruction entries. The following table describes the necessary details for EINJ.

**Table 18.23: Error Injection Table (EINJ)**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Byte Length** | **Byte Offset** | **Description** |
| **ACPI Standard Header** |  |  |  |
| Header Signature | 4 | 0 | EINJ, Signature for the Error Record Injection Table |
| Length | 4 | 4 | Length, in bytes, of entire EINJ. Entire table must be contiguous. |
| Revision | 4 |  | ~~1~~ 2 |
| … | … | … | … |

**The following table identifies the supported error injection actions.**

**Table 18.24: Error Injection Actions**

|  |  |  |
| --- | --- | --- |
| **Value** | **Name** | **Description** |
| 0x0 | BEGIN\_INJECTION\_OPERATION | Indicates to the platform that an error injection is beginning.  This allows the platform to set its operational context. |
|  |  |  |
| … | … | … |
|  |  |  |
| 0x10 | EINJV2\_SET\_ERROR\_TYPE | New Error Injection action introduced in EINJv2 for the purpose of injecting an EINJv2 error type, with address, severity, and detailed syndrome. Only one Error type can be injected at any given time. For multiple injections at the same time, then the platform will return an error condition.  The RegisterRegion field (See Table 18.25) in  EINJV2\_SET\_ERROR\_TYPE points to a data structure whose format is defined in Table 18.33.  See Table 18.32, for the EINJv2 error type definition. |
| 0x11 | EINJV2\_GET\_ERROR\_TYPE | Returns the extended error injection capabilities (EINJv2 features) of the platform.  Bit 0 – EINJv2 Processor Error Type supported  Bit 1 – EINJv2 Memory Error Type supported  Bit 2 – EINJv2 PCIe Error Type supported  Bit 3-30 – Reserved (Must be zero)  Bit 31 – EINJv2 Vendor specific error codes |
| 0xFF | TRIGGER\_ERROR | This Value is reserved for entries declared in the  Trigger Error Action Table returned in response to a  GET\_TRIGGER\_ERROR\_ACTION\_TABLE action. The  returned table consists of a series of actions each of which is  set to TRIGGER\_ERROR (see Table 18.32). When executed  by software, the series of TRIGGER\_ERROR actions triggers  the error injected as a result of the successful completion of  an EXECUTE\_OPERATION action. |

## 18.6.4 Error Types

The table below defines the error type codes returned from GET\_ERROR\_TYPE, as well as the error type set by SET\_ERROR\_TYPE and the Error Type field set by SET\_ERROR\_TYPE\_WITH\_ADDRESS (see Table 18.30).

Both the SET\_ERROR\_TYPE and SET\_ERROR\_TYPE\_WITH\_ADDRESS actions must be present as part of the EINJ Action Table. OSPM is free to choose either of these two actions to inject an error type. The platform will give precedence to SET\_ERROR\_TYPE\_WITH\_ADDRESS. That is, if a non-zero Error Type value is set by SET\_ERROR\_TYPE\_WITH\_ADDRESS, then any Error Type value set by SET\_ERROR\_TYPE will be ignored. But if no Error Type is specified by SET\_ERROR\_TYPE\_WITH\_ADDRESS, then the platform will use SET\_ERROR\_TYPE to identify the error type to inject.

Table 18.29: Error Type Definition

|  |  |
| --- | --- |
| **Bit** | **Description** |
| 0 | Processor Correctable |
| 1 | Processor Uncorrectable non-fatal |
| 2 | Processor Uncorrectable fatal |
| 3 | Memory Correctable |
| 4 | Memory Uncorrectable non-fatal |
| 5 | Memory Uncorrectable fatal |
| 6 | PCI Express Correctable |
| 7 | PCI Express Uncorrectable non-fatal |
| 8 | PCI Express Uncorrectable fatal |
| 9 | Platform Correctable |
| 10 | Platform Uncorrectable non-fatal |
| 11 | Platform Uncorrectable fatal |
| 12-29 | RESERVED |
| 30 | EINJv2 Error Type. If this bit is set, then the support for EINJV2\_SET\_ERROR\_TYPE and EINJV2\_GET\_ERROR\_TYPE actions are supported.  ***NOTE:*** *This may only be used with the action GET\_ERROR\_TYPE and it is not permitted to set this bit with SET\_ERROR\_TYPE or SET\_ERROR\_TYPE\_WITH\_ADDRESS* |
| 31 | Vendor Defined Error Type. If this bit is set, then the Error types and related data structures are defined by the Vendor, as shown in the Vendor Error Type Extension Stucuture |

…

### 18.6.4.1 EINJv2 Error Types

EINJ version 2 (EINJv2) introduces a new error injection action The table below defines the error type codes returned from EINJV2\_GET\_ERROR\_TYPE, as well as the error type set by EINJV2\_SET\_ERROR\_TYPE.

EINJV2\_SET\_ERROR\_TYPE actions must be present as part of the EINJ Action Table. OSPM is free to choose this action for advanced error injection options. The platform will give precedence to EINJV2\_SET\_ERROR\_TYPE. That is, if a non-zero Error Type value is set by EINJV2\_SET\_ERROR\_TYPE, then any Error Type value set by SET\_ERROR\_TYPE\_WITH\_ADDRESS and/or SET\_ERROR\_TYPE will be ignored.

Also, EINJv2 breaks out the Error Type from the severity. The following table describes the new Error Type encoding .

**Table 18.XX: EINJv2 Error Type**

|  |  |
| --- | --- |
| **Bit** | **Description** |
| 0 | Processor Error |
| 1 | Memory Error |
| 2 | PCIe Error |
| 3-30 | RESERVED |
| 31 | Vendor Defined Error |

**Table 18.YY:** **EINJV2\_SET\_ERROR\_TYPE Data Structure**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Byte Length** | **Byte Offset** | **Description** |
| Error Type | 4 | 0 | Bit map of error types to inject. Refer Table 18.AA EINJv2 Error Type Definition. Only one Error Type bit may be set at a time.  This field is cleared by the platform once it is consumed. |
| Error Type Code | 1 | 4 | Vendor specific code for each error type used to indicate error injection behavior.  Value of zero is default error injection behavior as defined by EINJv2.  Non-zero value indicates vendor specific behavior. |
| Flags | 3 | 5 | This field indicates which of the remaining fields are valid.  Bit 0 – Address Valid  Bit 1 - Address Range Valid  Bit 2 – Severity Valid  Bit 3 – Component Syndrome Count and Array is valid  All other bits are reserved |
| Length | 4 | 8 | This specifies the length of the entire structure including the component syndrome array. |
| Severity | 4 | 12 | Optional field specifying the severity of the injected error:  0 – Corrected Error  1 – Uncorrected Non-Fatal  2 – Uncorrected Fatal  All other values are reserved |
| Address | 8 | 16 | Optional field specifying the physical address of the memory that is the target for the injection. Valid if Bit [0] of the Flags field is set. |
| Address Range | 8 | 24 | Optional field specifying the physical address range mask of the memory that is the target for the injection. Valid if Bit [1] of the Flags field is set. |
| Component Syndrome Count (N) | 4 | 28 | This represents the maximum number of components supported in the Component Syndrome Array. This is intended to support injecting an error into multiple components / devices simultaneously.  For Example: If Component Syndrome Count is valid per the Flags field and the value Count (N) is 4, this structure contains the error Component Syndrome Array for 4 unique components. |
| Component Syndrome Array | N \* 32 | 32 | This is an array based on the Component Syndrome Entry Structure, each entry being 32-bytes as described in Table 18.ZZ. The length of this structure is 32 x Component Syndrome Count. |

**Table 18.ZZ: EINJV2\_SET\_ERROR\_TYPE Component Syndrome Structure**

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Byte Length** | **Byte Offset** | **Description** |
| Component ID | 16 | 0 | Component ID definition depends on the Error Type Value. Note that because this is a union structure, the byte length is 16 bytes to accommodate the largest possible ID (i.e. FRU ID for vendor specific errors).  Processor Error (0x1):  The bottom 32-bit represents the ACPI ID (represented in MADT) of the processor.  The remaining bits are vendor specific.  Memory Error (0x2):  This represents the Device ID within the memory module (e.g. DDR DIMM) for a particular system physical address. For example: 18 x 4 DIMMs support up to 18 devices (0-17) per address. 9 x 8 DIMMs support up to 9 devices (0-8) per address.  It is possible to inject error syndrome into multiple device instances up to the Component Syndrome Count.  PCIe Error (0x4):  This represents the SBDF.  Vendor Specific Error (0x80000000)  This represents some platform specific identifier (e.g. FRU ID or other vendor specific error format). |
| Component Syndrome | 16 | 16 | The Component Syndrome definition depends on the Error Type Value.  Processor Error (0x1):  The usage of the syndrome bits will be vendor specific.  Memory Error (0x2):  This indicates bit mask of data bits that must be flipped within a memory device. (e.g. If the set syndrome bit value is zero, the bit value will be changed to one. if the set syndrome bit value is one, the bit value will be changed to zero). The range of valid bits depends on the component error injection granularity.  Example 1: For a DDR4 18x4 memory device topology with a burst length of 8 (e.g., 64-byte cache line in a single burst), there will be up to 32 valid bits per device that may be modified per burst. If bit 3 in this mask is set, bit offset 3 of the device in that burst will be flipped.  Example 2: For a DDR5 5x8 memory device topology with a burst length of 16 (e.g., 64-byte cache line in a single burst), there will be up to 128 valid bits per device that may be modified per burst.  PCIe Error (0x4):  The usage of the syndrome bits will be vendor specific.  Vendor Specific Error (0x80000000)  The usage of the syndrome bits will be vendor specific. |

**18.6.7 Error Injection Version 2 Operation**

Before OSPM can use this mechanism to inject errors, it must discover the error injection capabilities of the platform by executing a EINJV2\_GET\_ERROR\_TYPE. See Error Type Definition for a definition of error types.

After discovering the EINJv2 error injection capabilities, OSPM can inject and trigger an error according to the sequence described below.

Note that injecting an error into the platform does not automatically consume the error. In response to an error injection, the platform returns a trigger error action table. The software that injected the error must execute the actions in the trigger error action table in order to consume the error. If a specific error type is such that it is automatically consumed on injection, the platform will return a trigger error action table consisting of NO\_OP.

1. Executes a BEGIN\_INJECTION\_OPERATION action to notify the platform that an error injection operation is beginning.

2. Executes a EINJ\_GET\_ERROR\_TYPE action to determine the error injection capabilities of the system. This action returns a DWORD bit map of the error types supported by the platform (see Table 18.29).

3. If EINJV2\_GET\_ERROR\_TYPE returns the DWORD with Bit [30] set, it means that EINJv2 error types are present, apart from the standard error types (see Table 18.29).

4. Executes a EINJV2\_GET\_ERROR\_TYPE action to determine the error injection capabilities of the system. This action returns a DWORD bit map of the error types supported by the platform (see Table 18.XX).

5. If EINJV2\_GET\_ERROR\_TYPE returns the DWORD with Bit [1] set, it means that memory error types are present.

6. OSPM chooses the type of error to inject by executing a EINJV2\_SET\_ERROR\_TYPE action.

a. If the OSPM chooses to inject one of the supported standard error types, then it sets the corresponding bit in the error type bitmap.

For example: if OSPM chooses to inject a Memory error, then the OSPM sets the value 0x0000\_0002 in the error type bitmap. If OSPM chooses to inject a memory error pattern into a device at a particular DRAM address, the Flags will be set to 0x9 to indicate component syndrome and address are valid. OSPM will populate the Address field with the system physical address and the component syndrome count and array syndrome.

Error Type = 0x2 (Memory Error)

Error Code = 0x0 (no vendor specific behavior)

Length=0x44

Severity=0 (not used)

Address=0000FFFFFFF00000

Address Range=0x0

Flags = 0x9 (Address and Component Syndrome Array is Valid)

Component Syndrome Count = 1

Component Syndrome Array [0] = { 00000000000000000000000000000004 , 000000000000000000000000A5A5A5A5 }

So in this example software is trying to inject a 32-bit bit flip pattern on one device across a single burst for a particular system physical address. EINJV2\_SET\_ERROR\_TYPE=000000000000000000000000A5A5A5A5000000000000000000000000000000040000000100000000000000000000FFFFFFF000000000000000000044000009000000002

7. Executes an EXECUTE\_OPERATION action to instruct the platform to begin the injection operation.

8. Busy waits by continually executing CHECK\_BUSY\_STATUS action until the platform indicates that the operation is complete by clearing the abstracted Busy bit.

9. Executes a GET\_COMMAND\_STATUS action to determine the status of the completed operation.

10. If the status indicates that the platform cannot inject errors, stop.