**# Title:**

Generic Interrupt Mux (GIM)

**# Status:**

Draft

**# Document:**

ACPI Specification Version 6.next

**# License:**

SPDX-License-Identifier: CC-BY-4.0

**# Submitter:**

* Harb Abdulhamid, Ampere Computing
* Samer El-Haj-Mahmoud, ARM
* TianoCore Community (<https://www.tianocore.org>)

**# Summary of the change**

This ECR introduces a new ACPI concept known as the “Generic Interrupt Mux Device (GIMD)”, and associated “Generic Interrupt Mux Instances (GIMI)”. A GIMD handles interrupt combining (i.e. group of wired interrupts OR’ed into a single wired interrupt represented by a single Global System Interrupt (GSI). This is concept is described using three new definitions:

* New “GIM Instance (GIMI)” Interrupt Controller Structure sub-table in the MADT to describe each GSI that may be the source of multiple “secondary interrupts”, which may be “OR’ed” on a single GSI wire that is routed to the central interrupt controller.
* New “GIM Device (GIMD)” Interrupt Controller Structure sub-table in the MADT to describe the GIM device with associated one or more GIMI structures.
* New “GIM Device” object in the namespace with a dedicated ACPI HID (ACPI0017), like the GED and HED.

This ECR builds upon past [ECR 1588](https://mantis.uefi.org/mantis/view.php?id=1588), which enabled the use of interrupt combining. This ECR is intended to enable ACPI firmware to further describe the details of the interrupt combining logic in a standard way so that OSPM can handle these configurations with a new standard software device driver.

***Note:*** *This builds on the existing concept of the usage of the “ResourceProducer”, “ResourceConsumer”, “ResourceSourceIndex”, and “ResourceSource” in the context of Interrupt descriptors, and how they may be used to describe interrupts generated by a device object that represent what current represent existing vendor specific “interrupt producers”.*

***Note:*** *Some of the content in* [ECR 1588](https://mantis.uefi.org/mantis/view.php?id=1588) *was missed in ACPI 6.3. This will be corrected in a future ACPI spec errata).*

There may exist many GIM Instances in the system. Each GIM instance is represented in list of static GIMI sub-tables within the MADT, as well as a list Memory/Interrupt resource descriptors. A GIM instance is effectively a register (up-to 64-bits in width) that maps to a single GSI wire, which holds either a raw or latched secondary interrupt status.

**# Benefits of the change**

This is extremely beneficial for highly dense systems that have a significant number of interrupt wires to be routed, where it might not be feasible to route every individual interrupt wire to a central interrupt controller.

Consider an example use-case with the following components:

* There exists a Generic Interrupt Mux Instance (GIMI), which multiplexes a large group of interrupt wires (e.g. 32 wires) and produces a single summary interrupt mapped to a single GSI (in a very similar way to GPIO controllers).  This is represented as a standard ACPI device, which in ASL can be represented as an “interrupt producer”, as it outputs “secondary (software based) interrupts” rather than physical interrupts.
* We have Device A with an individual interrupt wire going into a GIM instance. This GIM may work in one of two ways:
  + A simple OR gate, with a raw register to read the raw wire states (clearing is only required at the device, GIM instance register does not need to be cleared)
  + A register which latches the interrupt status, and requires clearing
* The device driver for Device A is some standard driver that expects to register for an interrupt resource.

**# Impact of the change**

OS vendors already supports the concept of interrupt combining in this manner but would need to implement a new standard driver for the GIM Device. In addition, ACPICA would need to cover the new MADT definitions to describe GIM instance descriptors.

**# Detailed description of the change [normative updates]**

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

**5.6.7 Device Class-Specific Objects**

**…**

**Table 5-183 ACPI Device IDs**

|  |  |
| --- | --- |
| **Plug and Play ID** | **Definition** |
| **…** | **…** |
| ACPI0017 | **Generic Interrupt Mux.** This device maps individual global system interrupts that have a combined set of secondary interrupts. See section 9.21 |

5.2.12 Multiple APIC Description Table (MADT)

**…**

Table 5-45 Interrupt Controller Structure Types

|  |  |  |  |  |
| --- | --- | --- | --- | --- |
| **Value** | **Description** | **\_MAT for Processor Objects** | **\_MAT for an I/O APIC object** | **Reference** |
| **…** | **….** | **…** | **…** | **…** |
| 0x10 | Generic Interrupt Mux Device (GIMD) Structure | no | no | Section 5.2.12.19 |
| 0x11 | Generic Interrupt Mux Instance (GIMI) Structure | no | no | Section 5.2.12.20 |
| ~~0x10~~0x12-0x7F | Reserved. OSPM skips structures of the reserved type. | no | no |  |

5.2.12.19 Generic Interrupt Mux Device (GIMD) Structure

The Generic Interrupt Mux Device (GIMD) describes the GIM device, which is a standard device used for describing the combining of a group of wired interrupts into a single wired summary interrupt, which to software is represented by a single Global System Interrupt (GSI). A GIMD structure is associated with one or more GIMI structures, as well as a GIM device in the namespace.

Table 5-67 GIMD Structure

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Byte Length** | **Byte Offset** | **Description** |
| Type | 1 | 0 | 0x10 GIMD Structure |
| Length | 1 | 1 | 12 |
| *Reserved* | 2 | 2 | Reserved - Must be zero |
| GIM Device UID | 4 | 4 | The OS associates this GIMD Structure with a GIM device object in the namespace when the \_UID child object of the GIM device evaluates to a numeric value that matches the numeric value in this field. |
| GIM Instance Count | 2 | 8 | Number of GIMI structures that are associated with this GIMD device. |
| *Reserved* | 2 | 10 | Reserved - Must be zero |

5.2.12.19 Generic Interrupt Mux Instance (GIMI) Structure

The Generic Interrupt Mux Instance (GIMI) combines a group of secondary interrupt wires, and outputs a single Global System Interrupt Vector (GSIV). A GIMI structure is associated with a GIM device object and a GIMID structure, and provides the number of secondary interrupts.

Table 5-68 GIMI Structure

|  |  |  |  |
| --- | --- | --- | --- |
| **Field** | **Byte Length** | **Byte Offset** | **Description** |
| Type | 1 | 0 | 0x11 GIM Instance Structure |
| Length | 1 | 1 | 48 |
| *Reserved* | 2 | 2 | Reserved - Must be zero |
| GIM Device UID | 4 | 4 | The UID of the GIMD structure and the GIM device that this GIMI is associated with. The OS associates this GIMI Structure with a GIM device object in the namespace when the \_UID child object of the GIM device evaluates to a numeric value that matches the numeric value in this field. |
| GIM Instance ID | 2 | 8 | This is a unique ID per GIM Instance. It maps to the resource order in the GIM device object |
| *Reserved* | 2 | 10 | Reserved - Must be zero |
| Flags | 4 | 12 | See Table 5-69 |
| GIM Summary Interrupt | 4 | 16 | GSIV of the interrupt associated with the summary interrupt of this GIM instance |
| GIM Status Register | 12 | 20 | |  | | --- | | This is the processor relative address of the GIM Status Register to be queried represented in Generic Address Structure format.  **Note:** Only the System I/O and System Memory spaces are valid for values for Address\_Space\_ID. | |
| Secondary Interrupt Count | 1 | 32 | The number of secondary source interrupts associated with this GIM instance.  **Example:** If the count is 20, bit 0 of the status register represents the first interrupt, bit 19 represents the last, and all other bits in the GIM status register are reserved. |
| *Reserved* | 1 | 33 | Reserved - Must be zero |
| Interrupt Base Number | 2 | 34 | The secondary interrupts must have a unique ID per GIM device. The secondary interrupt number is computed by adding the bit index of the GIM status register to the Interrupt Base Number.  **Example:** If bit 10 in the GIM status register is set and the interrupt base number of the GIM instance is 64, the GIM device raises secondary interrupt 74.  **Note**: It is illegal to have overlapping interrupts between each GIM instance |
| GIM Clear Register | 12 | 36 | This field is only valid if Flag bit 0 and bit 3 are set.  This is the processor relative address of the GIM Clear Register to be used for clearing interrupts represented in Generic Address Structure format.  **Note:** Only the System I/O and System Memory spaces are valid for values for Address\_Space\_ID. |

Table 5-69 GIMI Flags

|  |  |  |  |
| --- | --- | --- | --- |
| **GIMI Flags** | **Bit Length** | **Bit Offset** | **Description** |
| Clearing of the status register is required | 1 | 0 | 0 - The GIM interrupt status represents raw interrupt state, will be automatically cleared if no source interrupts are active  1 - The GIM interrupt status is latched and must be cleared per subsequent flag bits |
| Interrupt Type | 1 | 1 | (only valid if Bit 0 is one)  0 - Level  1 - Edge |
| Interrupt Polarity | 1 | 2 | (only valid if Bit 0 is one)  0 - Rising/High  1 - Falling/Low |
| Interrupt clearing method | 1 | 3 | (only valid if Bit 0 is one)  0 - Clear bit on write one  1 - Clear via separate clear register |
| Clear register behavior | 1 | 4 | (only valid if Bit 0 is one)  0 - Clear bit on write one  1 - Clear bit on write zero |
| *Reserved* | 27 | 5 | Must be zero. |

**9.21 Generic Interrupt Mux (GIM)**

The Generic Interrupt Mux (GIM) Device is a standard device used for describing the combining of a group of wired interrupts into a single wired summary interrupt, which to software is represented by a single Global System Interrupt (GSI). This is concept is described to the OSPM in two ways:

* New “GIM Device (GIMD)” Interrupt Controller Structure sub-table in the MADT to describe the GIM device with associated one or more GIMI structures. This sub-table is described in Section 5.2.12.19.
* New “GIM Instance (GIMI)” Interrupt Controller Structure sub-table in the MADT to describe each GSI that is a source for multiple secondary interrupt wires that have been “OR’ed” on a single GSI wire that will be routed to the primary interrupt controller. This sub-table is described in Section 5.2.12.20.
* New GIM device object with a dedicated ACPI HID (ACPI0017), like the GED and HED.

There may exist one or more GIM devices in the system (e.g. one per proximity domain). Each GIM instance is represented in list of static sub-table within the MADT. A GIM instance is effectively a register (or group of contiguous registers) that raises a single GSIV if any of the secondary (source) interrupts is active. The GIM instance provides a status register that holds interrupt status bits (bit per secondary interrupt). The status register will either provide raw status or a latched status that requires manual clearing.

The following figure describes an example of a system with GIM instances, which can be handled by the OPSM via the GIM device.



Figure 9.55: Generic Interrupt Mux (GIM) Example

In the above example, the MADT will have one GIMD sub-table entry, and two GIMI sub-table entries. The GIMI instance ID must be unique within a GIMD.

Scope(\\_SB) {

Device(GMD0){

Name(\_HID, EISAID("ACPI0017")) // vendor specific interrupt combiner

Name(\_UID, 0) // GIM Device 0

Name(\_PXM, 0) // Describes all GIM instances on socket 0

// GIM Instances (described in the GIMI sub-table MADT (no \_CRS required)

}

Device(DV12){

Name(\_HID, EISAID("PNP0D40")) // SDA Standard Compliant SD Host Controller

Name(\_UID, 0)

Name(\_CRS, ResourceTemplate () {

//Register Interface

MEMORY32FIXED(ReadWrite, 0xFF000000, 0x200, )

// Secondary Interrupt 10 from interrupt combiner MUX0

Interrupt(ResourceConsumer, Edge, ActiveHigh, Exclusive, 0,

“\\\_SB.GMD0”){6} // GIM instance 1, bit 2, interrupt base = 4

})

}