Tony363

PCI/PCIe 总线概述(6)—MSI和MSI-X中断机制

Posted on **2016年6月30日2016年7月1日** by **yangye363**

第6章 MSI和MSI-X中断机制

(2011-08-12 09:08:01)

转载▼

标签：	分类： **浅谈PCIe体系结构 (http://blog.sina.com.cn/s/articlelist_1685243084_3_1.html)**
杂谈 (http://search.sina.com.cn/?c=blog&q=%D4%D3%CC%B8&by=tag)	

在PCI总线中，所有需要提交中断请求的设备，必须能够通过INTx引脚提交中断请求，而MSI机制是一个可选机制。而在PCIe总线中，PCIe设备必须支持MSI或者MSI-X中断请求机制，而可以不支持INTx中断消息。

在PCIe总线中，MSI和MSI-X中断机制使用存储器写请求TLP向处理器提交中断请求，下文为简便起见将传递MSI/MSI-X中断消息的存储器写报文简称为MSI/MSI-X报文。不同的处理器使用了不同的机制处理这些MSI/MSI-X中断请求，如PowerPC处理器使用MPIC中断控制器处理MSI/MSI-X中断请求，本章将在第6.2节中介绍这种处理情况；而x86处理器使用FSB Interrupt Message方式处理MSI/MSI-X中断请求。

不同的处理器对PCIe设备发出的MSI报文的解释并不相同。但是PCIe设备在提交MSI中断请求时，都是向MSI/MSI-X Capability结构中的Message Address的地址写Message Data数据，从而组成一个存储器写TLP，向处理器提交中断请求。

有些PCIe设备还可以支持Legacy中断方式**[1] (http://blog.sina.com.cn/s/blog_6472c4cc0102dski.html#_ftn1)**。但是PCIe总线并不鼓励其设备使用Legacy中断方式，在绝大多数情况下，PCIe设备使用MSI或者MSI/X方式进行中断请求。

PCIe总线提供Legacy中断方式的主要原因是，在PCIe体系结构中，存在许多PCI设备，而这些设备通过PCIe桥连接到PCIe总线中。这些PCI设备可能并不支持MSI/MSI-X中断机制，因此必须使用INTx信号进行中断请求。

当PCIe桥收到PCI设备的INTx信号后，并不能将其直接转换为MSI/MSI-X中断报文，因为PCI设备使用INTx信号进行中断请求的机制与电平触发方式类似，而MSI/MSI-X中断机制与边沿触发方式类似。这两种中断触发方式不能直接进行转换。因此当PCI设备的INTx信号有效时，PCIe桥将该信号转换为Assert_INTx报文，当这些INTx信号无效时，PCIe桥将该信号转换为Deassert_INTx报文。

与Legacy中断方式相比，PCIe设备使用MSI或者MSI-X中断机制，可以消除INTx这个边带信号，而且可以更加合理地处理PCIe总线的“序”。目前绝大多数PCIe设备使用MSI或者MSI-X中断机制提交中断请求。

MSI和MSI-X机制的基本原理相同，其中MSI中断机制最多只能支持32个中断请求，而且要求中断向量连续，而MSI-X中断机制可以支持更多的中断请求，而并不要求中断向量连续。与MSI中断机制相比，MSI-X中断机制更为合理。本章将首先介绍MSI/MSI-X Capability结构，之后分别以PowerPC处理器和x86处理器为例介绍MSI和MSI-X中断机制。

[1] (http://blog.sina.com.cn/s/blog_6472c4cc0102dski.html#_ftnref1) 通过发送Assert_INTx和Deassert_INTx消息报文进行中断请求，即虚拟中断线方式。

6.1 MSI/MSI-X Capability结构

(2011-08-12 09:08:38)

转载▼

标签： **杂谈 (http://search.sina.com.cn/?c=blog&q=%D4%D3%CC%B8&by=tag)**	分类： **浅谈PCIe体系结构 (http://blog.sina.com.cn/s/articlelist_1685243084_3_1.html)**

PCIe设备可以使用MSI或者MSI-X报文向处理器提交中断请求，但是对于某个具体的PCIe设备，可能仅支持一种报文。在PCIe设备中含有两个Capability结构，一个是MSI Capability结构，另一个是MSI-X Capability结构。通常情况下一个PCIe设备仅包含一种结构，或者为MSI Capability结构，或者为MSI-X Capability结构。

6.1.1 MSI Capability结构

MSI Capability结构共有四种组成方式，分别是32和64位的Message结构，32位和64位带中断Masking的结构。MSI报文可以使用32位地址或者64位地址，而且可以使用Masking机制使能或者禁止某个中断源。MSI Capability寄存器的结构如图6-1所示。

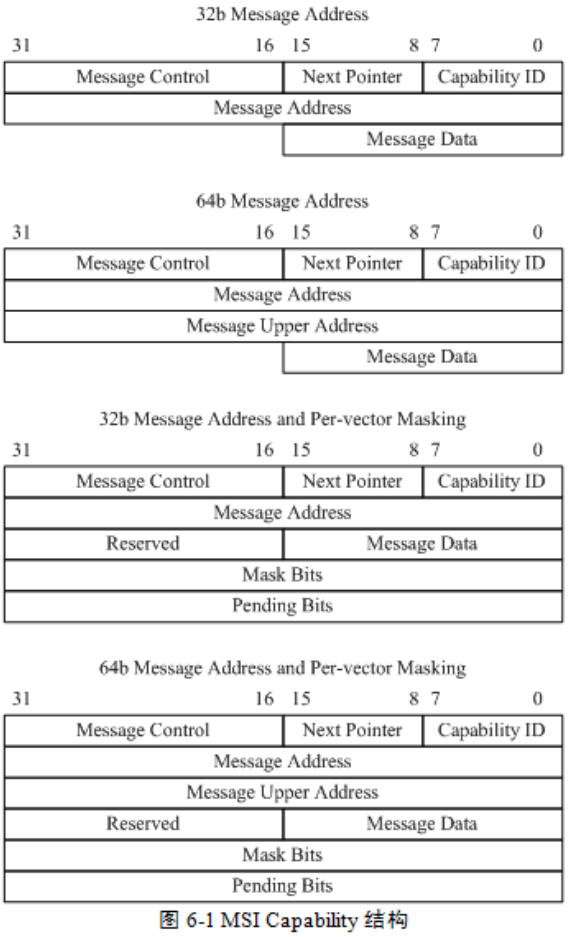

图 6-1 MSI Capability 结构

- Capability ID字段记载MSI Capability结构的ID号，其值为0x05。在PCIe设备中，每一个Capability结构都有唯一的ID号。
- Next Pointer字段存放下一个Capability结构的地址。
- Message Control字段。该字段存放当前PCIe设备使用MSI机制进行中断请求的状态与控制信息，如表6-1所示。

表6-1 MSI Cabalibities结构的Message Control字段

Bits	定义	描述
15:9	Reserved	保留位。系统软件读取该字段时将返回全零，对此字段写无意义。
8	Per-vector Masking Capable	该位为1时，表示支持带中断Masking的结构；如果为0，表示不支持带中断Masking的结构。该位对系统软件只读，该位在PCIe设备初始化时设置。
7	64 bit Address Capable	该位为1时，表示支持64位地址结构；如果为0，表示只能支持带32位地址结构。该位对系统软件只读，该位在PCIe设备初始化时设置。
6:4	Multiple Message Enable	该字段可读写，表示软件分配给当前PCIe设备的中断向量数目。系统软件根据Multiple Message Capable字段的大小确定该字段的值。在系统的中断向量资源并不紧张时，Multiple Message Capable字段和该字段的值相等；而资源紧张时，该字段的值可能小于Multiple Message Capable字段的值。
3:1	Multiple Message Capable	该字段对系统软件只读，表示当前PCIe设备可以使用几个中断向量号，在不同的PCIe设备中该字段的值并不不同。当该字段为0b000时，表示PCIe设备可以使用1个中断向量；为0b001、0b010、0b011、0b100和0b101时，表示使用4、8、16和32个中断向量；而0b110和0b111为保留位。 该字段与Multiple Message Enable字段的含义不同，该字段表示，当前PCIe设备支持的中断向量个数，而Multiple Message Enable字段是系统软件分配给PCIe设备实际使用的中断向量个数。
0	MSI Enable	该位可读写，是MSI中断机制的使能位。该位为1而且MSI-X Enable位为0时，当前PCIe设备可以使用MSI中断机制，此时Legacy中断机制被禁止。一个PCIe设备的MSI Enble和MSI-X Enable位都被禁止时，将使用INTx中断消息报文发出/结束中断请求**[1] (http://blog.sina.com.cn/s/blog_6472c4cc0102dskj.html#_ftn1)**。

- Message Address字段。当MSI Enable位有效时，该字段存放MSI存储器写事务的目的地址的低32位。该字段的31:2字段有效，系统软件可以对该字段进行读写操作；该字段的第1~0位为0。
- Message Upper Address字段。如果64 bit Address Capable位有效，该字段存放MSI存储器写事务的目的地址的高32位。
- Message Data字段，该字段可读写。当MSI Enable位有效时，该字段存放MSI报文使用的数据。该字段保存的数值与处理器系统相关，在PCIe设备进行初始化时，处理器将初始化该字段，而且不同的处理器填写该字段的规则并不相同。如果Multiple Message Enable字段不为0b000时(即该设备支持多个中断请求时)，PCIe设备可以通过改变Message Data字段的低位数据发送不同的中断请求。
- Mask Bits字段。PCIe总线规定当一个设备使用MSI中断机制时，最多可以使用32个中断向量，从而一个设备最多可以发送32种中断请求。Mask Bits字段由32位组成，其中每一位对应一种中断请求。当相应位为1时表示对应的中断请求被屏蔽，为0时表示允许该中断请求。系统软件可读写该字段，系统初始化时该字段为全0，表示允许所有中断请求。该字段和Pending Bits字段对于MSI中断机制是可选字段，但是PCIe总线规范强烈建议所有PCIe设备支持这两个字段。
- Pending Bits字段。该字段对于系统软件是只读位，PCIe设备内部逻辑可以改变该字段的值。该字段由32位组成，并与PCIe设备使用的MSI中断一一对应。该字段需要与Mask Bits字段联合使用。
  当Mask Bits字段的相应位为1时，如果PCIe设备需要发送对应的中断请求时，Pending Bits字段的对应位将被PCIe设备的内部逻辑置1，此时PCIe设备并不会使用MSI报文向中断控制器提交中断请求；当系统软件将Mask Bits字段的相应位从1改写为0时，PCIe设备将发送MSI报文向处理器提交中断请求，同时将Pending Bit字段的对应位清零。在设备驱动程序的开发中，有时需要联合使用Mask Bits和Pending Bits字段防止处理器丢弃中断请求**[2] (http://blog.sina.com.cn/s/blog_6472c4cc0102dskj.html#_ftn2)**。

6.1.2 MSI-X Capability结构

MSI-X Capability中断机制与MSI Capability的中断机制类似。PCIe总线引出MSI-X机制的主要目的是为了扩展PCIe设备使用中断向量的个数，同时解决MSI中断机制要求使用中断向量号连续所带来的问题。

MSI中断机制最多只能使用32个中断向量，而MSI-X可以使用更多的中断向量。目前Intel的许多PCIe设备支持MSI-X中断机制。与MSI中断机制相比，MSI-X机制更为合理。首先MSI-X可以支持更多的中断请求，但是这并不是引入MSI-X中断机制最重要的原因。因为对于多数PCIe设备，32种中断请求已经足够了。而引入MSI-X中断机制的主要原因是，使用该机制不需要中断控制器分配给该设备的中断向量号连续。

如果一个PCIe设备需要使用8个中断请求时，如果使用MSI机制时，Message Data的[2:0]字段可以为0b000~0b111，因此可以发送8种中断请求，但是这8种中断请求的Message Data字段必须连续。在许多中断控制器中，Message Data字段连续也意味着中断控制器需要为这个PCIe设备分配8个连续的中断向量号。

有时在一个中断控制器中，虽然具有8个以上的中断向量号，但是很难保证这些中断向量号是连续的。因此中断控制器将无法为这些PCIe设备分配足够的中断请求，此时该设备的“Multiple Message Enable”字段将小于“Multiple Message Capable”。

而使用MSI-X机制可以合理解决该问题。在MSI-X Capability结构中，每一个中断请求都使用独立的Message Address字段和Message Data字段，从而中断控制器可以更加合理地为该设备分配中断资源。

与MSI Capability寄存器相比，MSI-X Capability寄存器使用一个数组存放Message Address字段和Message Data字段，而不是将这两个字段放入Capability寄存器中，本篇将这个数组称为MSI-X Table。从而当PCIe设备使用MSI-X机制时，每一个中断请求可以使用独立的Message Address字段和Message Data字段。

除此之外MSI-X中断机制还使用了独立的Pending Table表，该表用来存放与每一个中断向量对应的Pending位。这个Pending位的定义与MSI Capability寄存器的Pending位类似。MSI-X Table和Pending Table存放在PCIe设备的BAR空间中。MSI-X机制必须支持这个Pending Table，而MSI机制的Pending Bits字段是可选的。

1 MSI-X Capability结构

MSI-X Capability结构比MSI Capability结构略微复杂一些。在该结构中，使用MSI-X Table存放该设备使用的所有Message Address和Message Data字段，这个表格存放在该设备的BAR空间中，从而PCIe设备可以使用MSI-X机制时，中断向量号可以并不连续，也可以申请更多的中断向量号。MSI-X Capability结构的组成方式如图6-2所示。

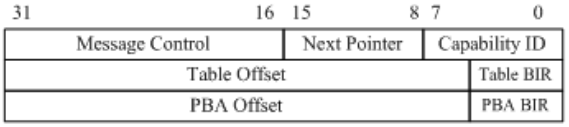

图 6-2 MSI-X Capability 结构的组成方式

上图中各字段的含义如下所示。

- Capability ID字段记载MSI-X Capability结构的ID号，其值为0x11。在PCIe设备中，每一个Capability都有唯一的一个ID号。
- Next Pointer字段存放下一个Capability结构的地址。
- Message Control字段，该字段存放当前PCIe设备使用MSI-X机制进行中断请求的状态与控制信息，如表6-2所示。

表6-2 MSI-X Capability结构的Message Control字段

Bits	定义	描述
15	MSI-X Enable	该位可读写，是MSI-X中断机制的使能位，复位值为0，表示不使能MSI-X中断机制。该位为1且MSI Enable位为0时，当前PCIe设备使用MSI-X中断机制，此时INTx和MSI中断机制被禁止。当PCIe设备的MSI Enble和MSI-X Enable位为0时，将使用INTx中断消息报文发出/结束中断请求。
14	Function Mask	该位可读写，是中断请求的全局Mask位，复位值为0。如果该位为1，该设备所有的中断请求都将被屏蔽；如果该位为0，则由Per Vector Mask位，决定是否屏蔽相应的中断请求。Per Vector Mask位在MSI-X Table中定义，详见下文。
10：0	Table Size	MSI-X中断机制使用MSI-X Table存放Message Address字段和Message Data字段。该字段用来存放MSI-X Table的大小，该字段对系统软件只读。

- Table BIR(BAR Indicator Register)。该字段存放MSI-X Table所在的位置，PCIe总线规范规定MSI-X Table存放在设备的BAR空间中。该字段表示设备使用BAR0~5寄存器中的哪个空间存放MSI-X table。该字段由三位组成，其中0b000~0b101与BAR0~5空间一一对应。
- Table Offset字段。该字段存放MSI-X Table在相应BAR空间中的偏移。
- PBA(Pending Bit Array) BIR字段。该字段存放Pending Table在PCIe设备的哪个BAR空间中。在通常情况下，Pending Table和MSI-X Table存放在PCIe设备的同一个BAR空间中。
- PBA Offset字段。该字段存放Pending Table在相应BAR空间中的偏移。

2 MSI-X Table

MSI-X Table的组成结构如图6-3所示。

DWORD 3	DWORD 2	DWORD 1	DWORD 0	
Vector Control	Msg Data	Msg Upper Addr	Msg Addr	Entry 0
Vector Control	Msg Data	Msg Upper Addr	Msg Addr	Entry 1
Vector Control	Msg Data	Msg Upper Addr	Msg Addr	Entry 2
...	...	...	...	...
Vector Control	Msg Data	Msg Upper Addr	Msg Addr	Entry N-1

图 6-3 MSI-X Table 的组成结构

由上图可见，MSI-X Table由多个Entry组成，其中每个Entry与一个中断请求对应。其中每一个Entry中有四个参数，其含义如下所示。

- Msg Addr。当MSI-X Enable位有效时，该字段存放MSI-X存储器写事务的目的地址的低32位。该双字的31:2字段有效，系统软件可读写；1:0字段复位时为0，PCIe设备可以根据需要将这个字段设为只读，或者可读写。不同的处理器填入该寄存器的数据并不相同。
- Msg Upper Addr，该字段可读写，存放MSI-X存储器写事务的目的地址的高32位。
- Msg Data，该字段可读写，存放MSI-X报文使用的数据。其定义与处理器系统使用的中断控制器和PCIe设备相关。
- Vector Control，该字段可读写。该字段只有第0位(即Per Vector Mask位)有效，其他位保留。当该位为1时，PCIe设备不能使用该Entry提交中断请求；为0时可以提交中断请求。该位在复位时为0。Per Vector Mask位的使用方法与MSI机制的Mask位类似。

3 Pending Table

Pending Table的组成结构如图6-4所示。

63	0	
Pending Bits 0 Through 63		QWORD 0
Pending Bits 64 Through 127		QWORD 1
...		...
Pending Bits (N-1 div 64)×64 Through N-1		QWORD ((N-1) div 64)

图 6-4 Pending Table 的组成结构

如上图所示，在Pending Table中，一个Entry由64位组成，其中每一位与MSI-X Table中的一个Entry对应，即Pending Table中的每一个Entry与MSI-X Table的64个Entry对应。与MSI机制类似，Pending位需要与Per Vector Mask位配置使用。

当Per Vector Mask位为1时，PCIe设备不能立即发送MSI-X中断请求，而是将对应的Pending位置1；当系统软件将Per Vector Mask位清零时，PCIe设备需要提交MSI-X中断请求，同时将Pending位清零。

[1] (http://blog.sina.com.cn/s/blog_6472c4cc0102dskj.html#_ftnref1) 此时PCI设备配置空间Command寄存器的“Interrupt Disable”位为1。

[2] (http://blog.sina.com.cn/s/blog_6472c4cc0102dskj.html#_ftnref2) MSI机制提交中断请求的方式类似与边界触发方式，而使用边界触发方式时，处理器可能会丢失某些中断请求，因此在设备驱动程序的开发过程中，可能需要使用这两个字段

6.2 PowerPC处理器如何处理MSI中断请求

(2011-08-12 09:13:51)

转载▼

标签：	分类： **浅谈PCIe体系结构 (http://blog.sina.com.cn/s/articlelist_1685243084_3_1.html)**
杂谈 (http://search.sina.com.cn/?c=blog&q=%D4%D3%CC%B8&by=tag)	

PowerPC处理器使用OpenPIC中断控制器或者MPIC中断控制器，处理外部中断请求。其中MPIC中断控制器基于OpenPIC中断控制器，但是作出了许多增强，目前Freescale新推出的PowerPC处理器，其中断控制器多与MPIC兼容。

值得注意的是，PowerPC处理器和x86处理器处理MSI报文的方式有较大的不同。其中x86处理器使用的机制比PowerPC处理器更为合理，但是PowerPC处理器使用的方法使用的硬件资源相对较少。本节将MPC8572处理器为例说明MSI机制的处理过程，在第6.3节介绍x86处理器如何实现MSI机制。

MPIC中断控制器是Freescale的PowerPC处理器使用的通用中断控制器，目前基于E500内核的处理器，如MPC854x、8572等处理器使用这种中断控制器。目前Freescale使用QorIP架构，该架构使用的中断控制器与MPIC兼容。

使用MPIC中断控制器处理MSI中断时，PCIe设备的MSI报文，其目的地址为MPIC中断控制器的MSIIR寄存器。当该寄存器被PCIe设备写入后，MPIC中断控制器将向处理器内核提交中断请求，之后处理器再通过读取MPIC中断控制器的ACK寄存器获得中断向量号，并进行相应的中断处理。这种方式与x86处理器的FSB Interrupt Message机制相比，处理器需要读取ACK寄存器，从而中断处理的延时较大。

目前Freescale的P4080处理器对MPIC中断控制器进行了优化。在P4080处理器中，MPIC中断控制器向处理器提交中断请求的同时，也向处理器内核提交中断向量，处理器内核不必读取ACK寄存器获得中断向量，从而缩短了中断处理延时。使用这种方法的效率与x86处理器使用的FSB Interrupt Message机制相当。

目前Freescale并没有完全公开P4080处理器的实现细节，因此本节仍以MPC8572处理器为例介绍PCIe设备的MSI中断请求。在MPC8572处理器中，MPIC中断控制器的拓扑结构如图6-5所示。

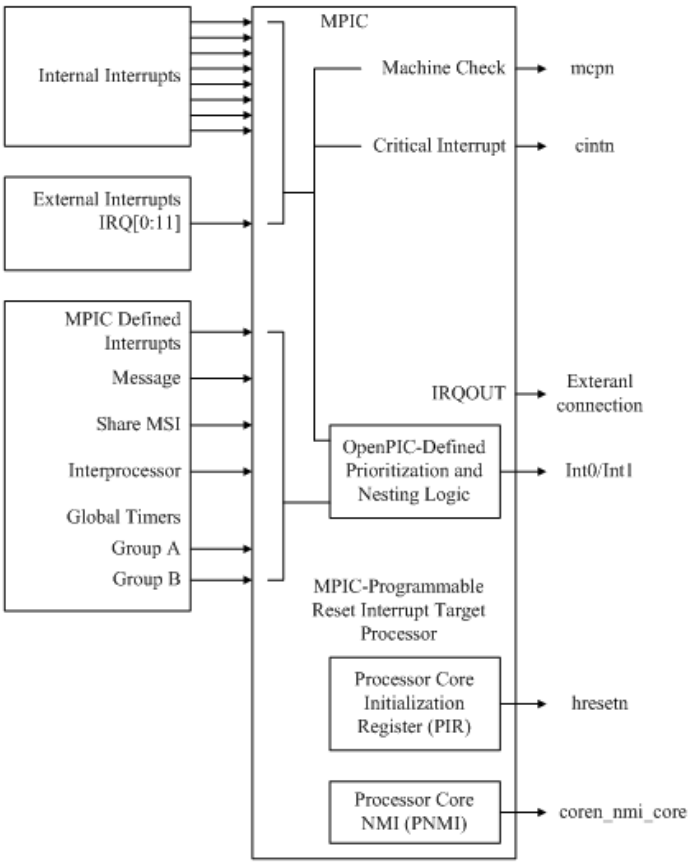

图 6-5 MPIC 中断控制器的拓扑结构

由上图所示，MPIC中断控制器可以处理内部中断请求**[1] (http://blog.sina.com.cn/s/blog_6472c4cc0102dskk.html#_ftn1)**、外部中断请求，Message、处理器间中断请求和Share MSI中断请求等。而MPIC中断控制器使用Int0、Int1等中断线向处理器提交这些中断请求。其中Internal Interrupts和External Interrupts模块处理MPC8572内部和外部的中断请求，而Share MSI处理来自PCIe设备的MSI或者MSI-X中断请求。

当MPIC中断控制器收到MSI报文后，将使用中断线Int0、Int1或者cintn向处理器内核提交中断请求。处理器内核被中断后，将读取ACK寄存器获得中断向量，然后执行相应的中断服务例程。为此PowerPC处理器设置了一系列寄存器，如下文所示。

6.2.1 MSI中断机制使用的寄存器

PowerPC处理器设置了一系列寄存器，处理来自PCIe设备的MSI报文，其中最重要的寄存器是MSIIR寄存器。在PowerPC处理器系统中，PCIe设备Message Address寄存器中存放的值都为MSIIR寄存器的物理地址，而Message Data寄存器中存放的数据也与MSIIR寄存器相关。

在PowerPC处理器系统中，MSI机制的实现过程是PCIe设备向MSIIR寄存器写入指定的数据。MPIC中断控制器发现该寄存器被写入后，将向处理器提交中断请求。处理器收到这个中断请求后，将通过读取MPIC中断控制器的ACK寄存器确定中断向量，并依此确定中断源。为此PowerPC处理器还设置了其他寄存器实现MSI中断机制。

1 MSIIR寄存器

在PowerPC处理器中，MSIIR(Shared Message Signaled Interrupt Index Register)寄存器是实现MSI机制的重要寄存器。

当PCIe设备对MSIIR寄存器进行写操作时，MPC8572处理器将使能MSIR0–MSIR7寄存器的相应位，从而向MPIC中断控制器提交中断请求，而中断控制器将转发这个中断请求，由处理器进一步处理。该寄存器各字段的详细描述如表6-3所示。

表6-3 MSIIR寄存器

Bits	定义	描述
27~31	IBS	该字段用来选择MSIR0~MSIR7寄存器的对应位。0b00000对应SH0；0b00001对应SH1；0b00010对应SH2；以此类推0b11111对应SH31；
24~26	SRS	该字段用来选择MSIR0~MSIR7寄存器。0b000对应MSIR0；0b001对应MSIR1；0b010对应MSIR2；以此类推0b111对应MSIR7。
0~24		保留。

PCIe设备通过MSI机制，向此寄存器写入数据时，MSIR0~7寄存器的相应位SH0~31将有一位置1。例如PCIe设备向MSIIR寄存器写入0xFF00000时，MSIR7寄存器的SH31位将置1(SRS字段为0b111用来选择MSIR7，而IBS字段为0b11111用来选择SH31)。

2 MSIR寄存器组

MSIR(Shared Message Signaled Interrupt Registers)寄存器组共由8个寄存器组成，分别为MSIR0~MSIR7。其中每一个MSIRx寄存器中有32个有效位，分别为SH0~31。当PCIe设备对MSIIR寄存器进行写操作时，某一个MSIIRx寄存器的某个SH位将被置为有效。系统软件通过读取该寄存器获得中断源，该寄存器读清除，对此寄存器进行写操作没有意义。

该寄存器组的大小决定了一个PowerPC处理器支持的MSI中断请求的个数。在MPC8572处理器中，有8个MSIRx寄存器，每个寄存器由32个有效位组成，因此MPC8572处理器最多能够处理256个MSI中断请求。该寄存器的结构如图6-6所示。

0	1	2	3	4	5	6	7	8	9	10	11	12	13	14	15
SH31	SH30	SH29	SH28	SH27	SH26	SH25	SH24	SH23	SH22	SH21	SH20	SH19	SH18	SH17	SH16
16	17	18	19	20	21	22	23	24	25	26	27	28	29	30	31
SH15	SH14	SH13	SH12	SH11	SH10	SH9	SH8	SH7	SH6	SH5	SH4	SH3	SH2	SH1	SH0

图 6-6 MSIRx 寄存器的结构

3 MSISR寄存器

MSISR寄存器(Shared Message Signaled Interrupt Status Register)共由8个有效位组成，每一位对应一个MSIR寄存器。MPC8572处理器设置该寄存器的主要目的是方便系统软件定位究竟是哪个MSIR寄存器中存在有效的中断请求。首先系统软件通过MSISR寄存器判断是哪个MSIRx寄存器存在有效请求，之后再读取相应的MSIRx寄存器，该寄存器各字段的详细描述如表6-4

所示。

表6-4 MSISR寄存器

Bits	定义	描述
0~23		保留。
24~31	Sn	该字段由8位组成，每一位与一个MSIR0~7寄存器对应。该位为0时表示在MSIRn寄存器中没有有效位，即没有中断请求；该位为1时表示MSIRn寄存器中至少有一个有效位，即存在中断请求。Sn位是MSIRn寄存器各个位的“与”，当MSIRn寄存器的相应位清除时，Sn也将被清除。

4 MSIVPR寄存器组

MSIVPR(Shared Message Signaled Interrupt Vector/Priority Register)寄存器组由8个寄存器组成，分别为MSIVPR0~7寄存器。该组寄存器设置对应中断请求的优先级别和中断向量。其中每个MSIVPR寄存器对应一个MSIR寄存器，MSIVPR寄存器各字段的详细解释如表6-5所示。

表6-5 MSIVPR寄存器

Bits	定义	描述
0	MSK	该位为0，且MSIR寄存器的对应位为1时，则将向中断控制器提交中断请求；如果为1屏蔽该中断请求。
1	A	该位为0时，表示MPIC中断控制器没有处理该中断请求；该位为1时，表示MPIC中断控制器正在处理该中断请求，或者该中断控制器准备处理该中断请求，这个中断请求将在IPR(Interrupt Pending Regsiter)寄存器中排队等待处理，或者在ISR(Interrupt Service Register)寄存器中正在被处理。该位的详细描述见MPC8572的数据手册。
12~15	PRIORITY	OpenPIC和MPIC中断控制器中为每一个中断请求设置了0~15，共16个优先级。其中1的优先权最低，15的优先权最高，0表示禁止中断请求。
16~31	VECTOR	该字段存放该中断的中断向量。当处理器读取IACK寄存器时，将获得对应中断请求的中断向量。

通过该组寄存器可以发现，在MPC8572处理器系统中，PCIe设备最多可以使用8个中断向量，并可以共享这些中断向量。

5 MSIDR寄存器组

MSIDR(Shared Message Signaled Interrupt Destination Registers)寄存器组共由8个寄存器组成，分别为MSIDR0~7。其中每一个MSIDRn寄存器对应一个MSIR寄存器。

MPIC中断控制器支持Pass-through方式，在这种方式下，PowerPC处理器可以使用外部中断控制器处理中断请求(这种方法极少使用)，而不使用内部中断控制器。MPIC中断控制器可以使用cint#和int#信号提交中断请求，但是绝大多数系统软件都使用int#信号向处理器提交中断请求。

此外在MPC8572处理器中有两个CPU，分别为CPU0和CPU1，MSI机制提交的中断请求可以由CPU0或者CPU1处理。系统软件可以通过设置MSIDRn寄存器完成这些功能，该寄存器各字段的详细描述如表6-6所示。

表6-6 MSIDRn寄存器

Bits	定义	描述
0	EP	为1时，表示中断请求输出到IRQ_OUT由外部中断控制器处理；为0时，表示由MPIC中断控制器处理。

Bits	定义	描述
1	CI0	为1时，表示中断控制器使用cint#信号向CPU0提交中断请求。
2	CI1	为1时，表示中断控制器使用cint#信号向CPU1提交中断请求。
30	P1	为1时，表示中断控制器使用int#信号向CPU0提交中断请求。
31	P0	为1时，表示中断控制器使用int#信号向CPU1提交中断请求。

6.2.2 系统软件如何初始化PCIe设备的MSI Capability结构

如果PCIe设备支持MSI机制，系统软件首先设置该设备MSI Capability结构的Message Address和Message Data字段。如果该PCIe设备支持64位地址空间，即MSI Capability寄存器的64 bit Address Capable位有效时，系统软件还需要设置Message Upper Address字段。系统软件完成这些设置后，将置MSI Cabalibities结构的MSI Enable位有效，使能该PCIe设备的MSI机制。

其中Message Address字段所填写的值是MSIIR寄存器在PCI总线域中的物理地址。在PowerPC处理器中，PCI总线域与存储器域地址空间独立，当PCIe设备访问存储器域的地址空间时，需要通过Inbound寄存器组将PCI总线域地址空间转换为存储器域地址空间。

在PowerPC处理器中，PCIe设备使用MSI机制访问MSIIR寄存器时，可以不使用Inbound寄存器组进行PCI总线地址到处理器地址的转换。在MPC8572处理器中，专门设置了一个PEXCSRBAR窗口**[2] (http://blog.sina.com.cn/s/blog_6472c4cc0102dskk.html#_ftn2)**，进行PCI总线域到存储器域的地址转换，使用这种方法可以节省Inbound寄存器窗口，Linux PowerPC使用了这种实现方式。

在MPC8572处理器中，MSIIR寄存器的基地址为CCSRBAR**[3] (http://blog.sina.com.cn/s/blog_6472c4cc0102dskk.html#_ftn3)** (Configuration, Control, and Status Base Address Register)，其偏移为0x1740。为支持MSI中断机制，系统软件需要使用PEXCSRBAR窗口将MSIIR寄存器映射到PCI总线域地址空间，即将CCSRBAR寄存器空间映射到PCI总线域地址空间。之后PCIe设备就可以通过MSIIR寄存器在PCI总线域的地址访问MSIIR寄存器。

Linux PowerPC使用setup_pci_pcsrbar函数**[4] (http://blog.sina.com.cn/s/blog_6472c4cc0102dskk.html#_ftn4)**设置PEXCSRBAR窗口，该函数的源代码在./arch/powerpc/sysdev/fsl_pci.c文件中，如源代码6-1所示，这段代码来自Linux 2.6.30.5。

源代码6-1 setup_pci_pcsrbar函数

```
static void __init setup_pci_pcsrbar(struct pci_controller *hose)

{

#ifdef CONFIG_PCI_MSI

    phys_addr_t immr_base;



    immr_base = get_immrbase();

    early_write_config_dword(hose, 0, 0, PCI_BASE_ADDRESS_0, immr_base);

#endif

}
```

系统软件除了需要设置PCIe设备的Message Address字段和PEXCSRBAR窗口之外，还需要设置PCIe设备的Message Data字段。PCIe设备向MSIIR寄存器进行存储器写操作的数据存放在Message Data字段中。

系统软件在初始化Message Data字段之前，首先根据Multiple Message Capable字段预先存放的数据初始化Multiple Message Enable字段。一个PCIe设备最多可以申请32个中断请求，但是系统软件根据当前处理器系统的中断资源的使用情况，决定给这个PCIe设备提供多少个中断向量，并将这个结果存放到Multiple Message Enable字段。

MPC8572处理器最多可以为PCIe设备提供256个MSI中断请求。但是在某些极端的情况下，可能会出现PCIe设备需要的中断请求超过系统所能提供的中断请求。此时某些PCIe设备的Multiple Message Enable字段可能会小于Multiple Message Capable字段。

如果在PCIe设备中，使用了多个中断请求，那么Message Data字段存放的是一组中断向量号，而Message Data字段存放这组中断向量号的基地址。MSI机制要求“这组数据”连续，其范围在Message Data~Message Data+Multiple Message Enable-1之间。在多数情况下，MPC8572处理器系统仅为一个PCIe设备分配1个中断向量号。

由上所述，在MPC8572处理器系统中，PCIe设备使用存储器写TLP传送MSI中断报文，这个存储器写TLP使用的地址为PCIe设备Capability结构的Message Address字段，而数据为Message Data~Message Data + Multiple Message Enable-1之间。其中Message Data字段与MSIIR寄存器要求的格式相同。

这个特殊的存储器写TLP报文通过若干Switch，并穿越RC后，最终将数据写入MSIIR寄存器中，并设置MSIIR寄存器的SRS和IBS字段，同时将使能MSIR0~MSIR7寄存器的相应位，从而向中断控制器提交中断请求(如果MSIVPR寄存器的MSK位为1)。MPIC中断控制器获得该中断请求后，向处理器系统转发这个中断请求，并由处理器系统执行相应的中断服务例程进行中断处理。MPC8572处理器也可以处理PCIe设备的MSI-X中断机制，本节对此不做进一步介绍。

[1] (http://blog.sina.com.cn/s/blog_6472c4cc0102dskk.html#_ftnref1) PowerPC处理器中含有许多模块，如千兆以太网、ATM等，这些模块包含在芯片内部，由这些内部模块发起的中断请求，被称为内部中断请求。

[2] (http://blog.sina.com.cn/s/blog_6472c4cc0102dskk.html#_ftnref2) 该窗口的大小为1MB，其基地址由PEXCSRBAR寄存器确定。

[3] (http://blog.sina.com.cn/s/blog_6472c4cc0102dskk.html#_ftnref3) 在Linux PowerPC中使用immr_base变量保存该寄存器。IMMR寄存器是PQ2处理器使用的寄存器，该寄存器在PQ3之后的处理器中升级为CCSRBAR。

[4] (http://blog.sina.com.cn/s/blog_6472c4cc0102dskk.html#_ftnref4) 该函数来自Linux 2.6.30.5内核。

6.3 x86处理器如何处理MSI-X中断请求

(2011-08-12 09:16:20)

转载▼

标签：	分类： **浅谈PCIe体系结构 (http://blog.sina.com.cn/s/articlelist_1685243084_3_1.html)**
杂谈 (http://search.sina.com.cn/?c=blog&q=%D4%D3%CC%B8&by=tag)	

PCIe设备发出MSI-X中断请求的方法与发出MSI中断请求的方法类似，都是向Message Address所在的地址写Message Data字段包含的数据。只是MSI-X中断机制为了支持更多的中断请求，在MSI-X Capablity结构中存放了一个指向一组Message Address和 Message Data字段的指针，从而一个PCIe设备可以支持的MSI-X中断请求数目大于32个，而且并不要求中断向量号连续。MSI-X机制使用的这组Message Address和 Message Data字段存放在PCIe设备的BAR空间中，而不是在PCIe设备的配置空间中，从而可以由用户决定使用MSI-X中断请求的数目。

当系统软件初始化PCIe设备时，如果该PCIe设备使用MSI-X机制传递中断请求，需要对MSI-X Capability结构指向的Message Address和Message Data字段进行设置，并使能MSI-X Enable位。x86处理器在此处的实现与PowerPC处理器有较大的不同。

6.3.1 Message Address字段和Message Data字段的格式

在x86处理器系统中，PCIe设备也是通过向Message Address写入Message Data指定的数值实现MSI/MSI-X机制。在x86处理器系统中，PCIe设备使用的Message Adress字段和Message Data字段与PowerPC处理器不同。

1 PCIe设备使用Message Adress字段

在x86处理器系统中，PCIe设备使用的Message Address字段仍然保存PCI总线域的地址，其格式如图6-7所示。

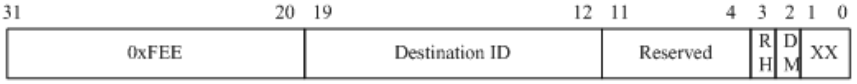

图 6-7 Message Address 字段的格式

其中第31~20位，存放FSB Interrupts存储器空间的基地址，其值为0xFEE。当PCIe设备对0xFEEX-XXXX这段“PCI总线域”的地址空间进行写操作时，MCH/ICH将会首先进行“PCI总线域”到“存储器域”的地址转换，之后将这个写操作翻译为FSB总线的Interrupt Message总线事务，从而向CPU内核提交中断请求。

x86处理器使用FSB Interrupt Message总线事务转发MSI/MSI-X中断请求。使用这种方法的优点是向CPU内核提交中断请求的同时，提交PCIe设备使用的中断向量，从而CPU不需要使用中断响应周期从寄存器中获得中断向量。FSB Interrupt Message总线事务的详细说明见下文。

Message Address字段其他位的含义如下所示。

- Destination ID字段保存目标CPU的ID号，目标CPU的ID与该字段相等时，目标CPU将接收这个Interrupt Message。FSB Interrupt Message总线事务可以向不同的CPU提交中断请求。
- RH(Redirection Hint Indication)位为0时，表示Interrupt Message将直接发向与Destination ID字段相同的目标CPU；如果RH为1时，将使能中断转发功能。
- DM(Destination Mode)位表示在传递优先权最低的中断请求时，Destination ID字段是否被翻译为Logical或者Physical APIC ID。在x86处理器中APIC ID有三种模式，分别为Physical、Logical和Cluster ID模式。
- 如果RH位为1且DM位为0时，Destination ID字段使用Physical模式；如果RH位为1且DM位为1，Destination ID字段使用Logical模式；如果RH位为0，DM位将被忽略。

以上这些字段的描述与x86处理器使用的APIC中断控制器相关。对APIC的详细说明超出了本书的范围，对此部分感兴趣的读者请参阅Intel 64 and IA-32 Architectures Software Developer’s Manual Volume 3A: System Programming Guide, Part 1。

2 Message Data字段

Message Data字段的格式如图6-8所示。

图 6-8 Message Data 字段的格式

Trigger Mode字段为0b0x时，PCIe设备使用边沿触发方式申请中断；为0b10时使用低电平触发方式；为0b11时使用高电平触发方式。MSI/MSI-X中断请求使用边沿触发方式，但是FSB Interrupt Message总线事务还支持Legacy INTx中断请求方式，因此在Message Data字段中仍然支持电平触发方式。但是对于PCIe设备而言，该字段为0b0x。

Vector字段表示这个中断请求使用的中断向量。FSB Interrupt Message总线事务在提交中断请求的同时，将中断向量也通知给处理器。因此使用FSB Interrupt Message总线事务时，处理器不需要使用中断响应周期通过读取中断控制器获得中断向量号。与PowerPC的传统方式相比，x86处理器的这种中断请求的效率较高**[①] (http://blog.sina.com.cn/s/blog_6472c4cc0102dskl.html#_ftn1)**。

值得注意的是，在x86处理器中，MSI机制使用的Message Data字段与MSI-X机制相同。但是当一个PCIe设备支持多个MSI中断请求时，其Message Data字段必须是连续的，因而其使用的Vector字段也必须是连续的，这也是在x86处理器系统中，PCIe设备支持多个MSI中断请求的问题所在，而使用MSI-X机制有效避免了该问题。

Delivery Mode字段表示如何处理来自PCIe设备的中断请求。

- 该字段为0b000时，表示使用“Fixed Mode”方式。此时这个中断请求将被Destination ID字段指定的CPU处理。
- 该字段为0b001时，表示使用“Lowest Priority”方式。此时这个中断请求将被优先权最低的CPU处理。当使用“Fixed Mode”和“Lowest Priority”方式时，如果Vector字段有效，CPU接收到这个中断请求之后，将使用Vector字段指定的中断向量处理这些中断请求；而当Delivery Mode字段为其他值时，Message Data字段中所包含的Vector字段无效。

- 该字段为0b010时，表示使用SMI方式传递中断请求，而且必须使用边沿触发，此时Vector字段必须为0。这个中断请求将被Destination ID字段指定的CPU处理。
- 该字段为0b100时，表示使用NMI方式传递中断请求，而且必须使用边沿触发，此时Vector字段和Trigger字段的内容将被忽略。这个中断请求将被Destination ID字段指定的CPU处理。
- 该字段为0b101时，表示使用INIT方式传递中断请求，Vector字段和Trigger字段的内容将被忽略。这个中断请求将被Destination ID字段指定的CPU处理。
- 该字段为0b111时，表示使用INTR信号传递中断请求且使用边沿触发。此时MSI中断信息首先传递给中断控制器，然后中断控制器在通过INTR信号向CPU传递中断请求，之后CPU在通过中断响应周期获得中断向量。上文中PowerPC处理器使用的方法与此方法类似。而在x86处理器中多使用Interrupt Message总线事务进行MSI中断信息的传递，因此这种模式很少被使用。

边沿触发和电平触发是中断请求常用的两种方式。其中电平触发指外部设备使用逻辑电平1(高电平触发)或者0(低电平触发)，提交中断请求。使用电平或者边沿方式提交中断请求时，外部设备一般通过中断线(IRQ_PIN#)与中断控制器相连，其中多个外部设备可能通过相同的中断线与中断控制器相连(线与或者与门)。

外部设备在使用低电平触发，提交中断请求的过程中，首先需要将IRQ_PIN#信号驱动为低。当中断控制器将该中断请求提交给处理器，而且处理器将这个中断请求处理完毕后，处理器将通过写外部设备的某个寄存器来清除此中断源，此时外部设备将不再驱动IRQ_PIN#信号线，从而结束整个中断请求。

IRQ_PIN#信号线可以被多个外部设备共享，在这种情况之下，只有所有外部设备都不驱动IRQ_PIN#信号线时，IRQ_PIN#信号才为高电平。采用电平触发方式进行中断请求的优点是不会丢失中断请求，而缺点是一个优先权较高的中断请求有可能会长期占用中断资源，从而使其他优先权较低的中断不能被及时提交。因为优先级别较高的中断源有可能会持续不断地驱动IRQ_PIN#信号。

而边沿触发使用上升沿(0到1)或者下降沿(1到0)作为触发条件，但是中断控制器并不是使用这个“边沿”作为触发条件。中断控制器使用内部时钟对IRQ_PIN#信号进行采样，如果在前一个时钟周期，IRQ_PIN#信号为0，而后一个时钟周期，IRQ_PIN#信号为1，中断控制器认为外部设备提交了一个有效“上升沿”，中断控制器会锁定这个“上升沿”并向处理器发出中断请求。这也是外部设备至少需要将IRQ_PIN#信号保持一个时钟采样周期的原因，否则中断控制器可能无法识别本次边沿触发的中断请求，从而产生Spurious中断请求。

外部设备使用“上升沿”进行中断申请时，不需要持续地将IRQ_PIN#信号驱动为1，而只需要保证中断控制器可以进行正确采样这些中断信号即可。在处理边沿触发中断请求时，处理器不需要清除中断源。

使用边沿触发可以有效避免“优先级别”较高的中断源长期占用IRQ_PIN#信号的情况，使用“下降沿”触发进行中断请求与“上升沿”触发类似。

但是外部设备使用边沿触发方式时，有可能会丢失一些中断请求。例如在一个处理器系统中，存在一个定时器，这个定时器使用上升沿触发方式向中断控制器定时提交中断。如果当处理器正在处理这个定时器的上一个中断请求时，将不会处理这个定时器发出的其他“边沿”中断请求，从而导致中断丢失。而使用电平触发方式不会出现这类问题，因为电平触发方式是一个“持续”过程，处理器只有处理完毕当前中断，并清除相应中断源之后，才会处理下一个中断源。

MSI中断请求实际上和边沿触发方式非常类似，MSI中断请求通过存储器写TLP实现，这个写动作是一个瞬间的动作，并不是一个持续请求，因此在x86处理器中MSI中断请求使用边沿触发进行中断请求。

还有一些外部设备可以通过I/O APIC进行中断请求**[②] (http://blog.sina.com.cn/s/blog_6472c4cc0102dskl.html#_ftn2)**，这些I/O APIC接收的外部中断需要标明是使用边沿或者电平触发，I/O APIC使用FSB Interrupt Message总线事务将中断请求发向Local APIC，并由Local APIC向处理器提交中断请求。

6.3.2 FSB Interrupt Message总线事务

与MPC8572处理器处理MSI中断请求不同，x86处理器使用FSB的Interrupt Message总线事务，处理PCIe设备的MSI/MSI-X中断请求。由上文所示，MPC8572处理器处理MSI中断请求时，首先由MPIC中断控制器截获这个MSI中断请求，之后由MPIC中断控制器向CPU提交中断请求，而CPU通过中断响应周期从MPIC中断控制器的ACK寄存器中获得中断向量。

采用这种方式的主要问题是，当一个处理器中存在多个CPU时，这些CPU都需要通过中断响应周期从MPIC中断控制器的ACK寄存器中获得中断向量。在一个中断较为密集的应用中，ACK寄存器很可能会成为系统瓶颈。而采用Interrupt Message总线事务可以有效地避免这种系统瓶颈，因为使用这种方式中断信息和中断向量将同时到达指定的CPU，而不需要使用中断响应周期获得中断向量。

x86处理器也具有通过中断控制器提交MSI/MSI-X中断请求的方法，在I/O APIC具有一个 “The IRQ Pin Assertion Register”寄存器，该寄存器地址为0xFEC00020**[③] (http://blog.sina.com.cn/s/blog_6472c4cc0102dskl.html#_ftn3)**，其第4~0位存放IRQ Number。系统软件可以将PCIe设备的Message Address寄存器设置为0xFEC00020，将Meaasge Data寄存器设置为相应的IRQ Number。

当PCIe设备需要提交MSI中断请求时，将向PCI总线域的0xFEC00020地址写入Message Data寄存器中的数据。此时这个存储器写请求将数据写入I/O APIC的The IRQ Pin Assertion Register中，并由I/O APIC将这个MSI中断请求最终发向Local APIC，之后再由Local APIC通过INTR#信号向CPU提交中断请求。

上述步骤与MPC8572处理器传递MSI中断的方法类似。在x86处理器中，这种方式基本上已被弃用。下文以图6-9为例，说明x86处理器如何使用FSB总线的Interrupt Message总线事务，向CPU提交MSI/MSI-X中断请求。

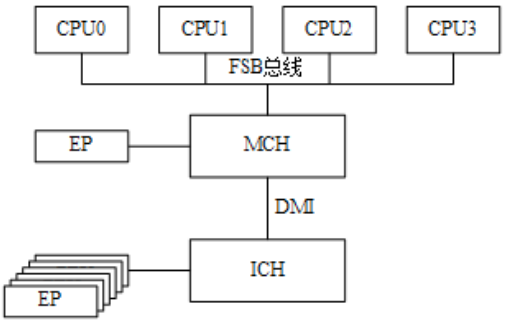

图 6-9 使用 Interrupt Message 总线事务传递 MSI 中断请求

PCIe设备在发送MSI/MSI-X中断请求之前，系统软件需要合理设置PCIe设备MSI/MSI-X Capability寄存器，使Message Address寄存器的值为0xFEExx00y**[④] (http://blog.sina.com.cn/s/blog_6472c4cc0102dskl.html#_ftn4)**，同时合理地设置Message Data寄存器Vector字段。

PCIe设备提交MSI/MSI-X中断请求时，需要向0xFEExx00y地址写Message Data寄存器中包含的数据，并以存储器写TLP的形式发送到RC。如果ICH收到这个存储器写TLP时，将通过DMI接口将这个TLP提交到MCH。MCH收到这个TLP后，发现这个TLP的目的地址在FSB Interrupts存储器空间中，则将PCIe总线的存储器写请求转换为Interrupt Message总线事务，并在FSB总线上广播。

FSB总线上的CPU，根据APIC ID信息，选择是否接收这个Interrupt Message总线事务，并进入中断状态，之后该CPU将直接从这个总线事务中获得中断向量号，执行相应的中断服务例程，而不需要从APIC中断控制器获得中断向量。与PowerPC处理器的MPIC中断控制器相比，这种方法更具优势。

6.4 小结

本章详细描述了MSI/MSI-X中断机制的原理，并以PowerPC和x86两个处理器系统为例说明这两种中断机制实现机制。本章因为篇幅有限，并没有详细讲述这两个处理器使用的中断控制器。而理解这些中断控制器的实现机制是进一步理解MSI/MSI-X中断机制的要点。对此部分有兴趣的读者可以继续阅读MPIC中断控制器和APIC中断控制器的实现机制，以加深对MSI/MSI-X中断机制的理解。

设备的中断处理是局部总线的设计难点和重要组成部分，而中断处理的效率直接决定了局部总线的数据传送效率。在一个处理器系统的设计与实现中，中断处理的优化贯彻始终。

[①] (http://blog.sina.com.cn/s/blog_6472c4cc0102dskl.html#_ftnref1) P4080处理器也提供了一种类似于FSB Interrupt Message总线事务的中断请求方法。

[②] (http://blog.sina.com.cn/s/blog_6472c4cc0102dskl.html#_ftnref2) 与I/O APIC的IRQX#引脚链接的外部设备。

[③] (http://blog.sina.com.cn/s/blog_6472c4cc0102dskl.html#_ftnref3) 该寄存器在存储器域和PCI总线域中的地址都为0xFEC00020。

[④] (http://blog.sina.com.cn/s/blog_6472c4cc0102dskl.html#_ftnref4) 其中xx表示APIC ID，而y为RH+DM。

Categories: **IO总线技术**