PCI Express中断模型支持两种机制:

* INTx仿真
* 消息信号中断(MSI/MSI-X)

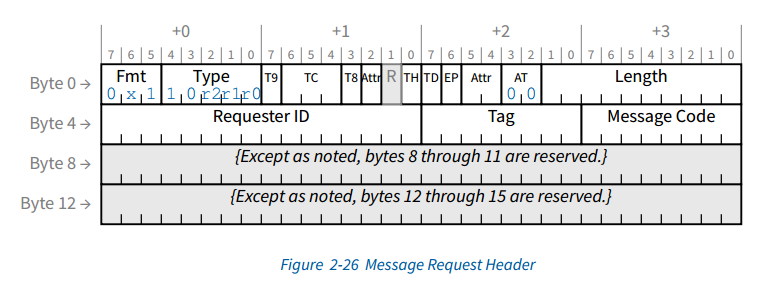
为了传统的兼容性，PCI Express提供了一个PCI INTx模拟机制来向系统中断控制器(通常是RC的一部分)发送中断信号。该机制与现有的PCI软件兼容，并提供与相应的PCI中断信令机制相同级别和类型的服务，并且独立于系统中断控制器的具体规定。这种传统的兼容性机制允许启动设备支持，而不需要复杂的bios级中断配置/控制服务堆栈。它通过使用带内信号机制来虚拟化PCI物理中断信号。

如果实现支持中断，那么该规范要求支持MSI或MSI-X，或者两者都支持。PCI兼容INTx中断仿真是可选的。交换机需要支持转发INTx中断仿真消息(参见2.2.8.1节)。PCI Express MSI和MSI- x机制与最初在[PCI]中定义的机制兼容。

# 2.2.8消息请求规则

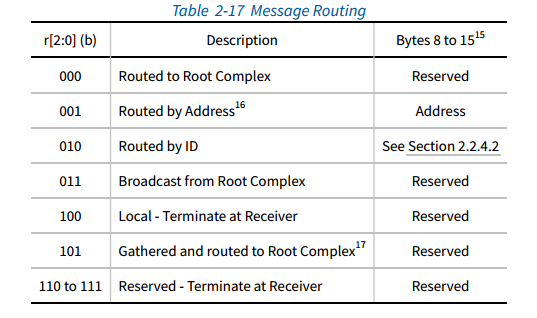
以下规则适用于所有消息请求。接下来是特定于每种消息类型的其他规则。

* 所有的消息请求包括以下字段除了常见的报头字段(见图2-37):
* requester ID[15:0]和TAG[9:0]，形成Transaction ID。
* Message code[7:0]-指定具体的消息体现在请求。
* 所有消息请求使用Msg或MsgD类型字段编码。
* Message code字段必须完全解码(消息混叠是不允许的)。
* Attr[2]字段不保留，除非特别指定为保留。
* 除特别说明外，Attr[1:0]字段为保留字段。
* LN不适用于Message Requests，该位为Reserved。
* 除特别说明外，TH不适用于Message Requests，该位为Reserved。
* AT[1:0]必须是00b。不要求或不鼓励接收方检查此选项。
* 除特别说明外，第8到15字节为预留。
* 消息请求是posted，不需要完成。
* 消息请求遵循与memory write请求相同的排序规则。



除了地址和ID路由之外，消息还支持其他几种路由机制。这些机制被称为“隐式的”，因为没有地址或ID指定目的地，而是由路由类型隐含目的地。以下规则涵盖了消息路由机制:

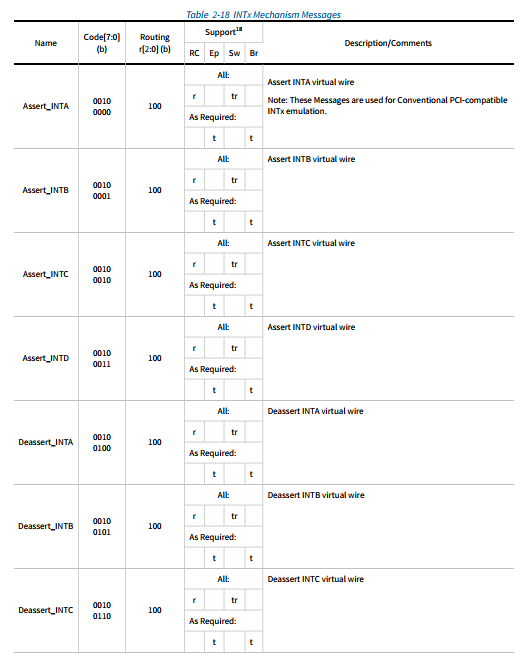
* 消息路由是使用Type字段的r[2:0]子字段确定的;
* 消息路由r[2:0]值在表2-17中定义
* 在以下部分中为每个消息定义了允许值

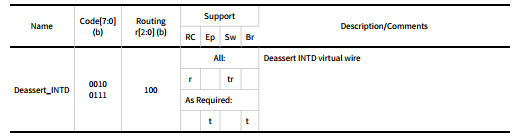


## 2.2.8.1 INTx中断信令-规则

消息信号中断(MSI或MSI- x)是PCI Express中首选的中断信号机制(参见6.1节)。然而，在某些系统中，可能存在不支持MSI或MSI- x机制的函数。在不能使用MSI或MSI-X机制的情况下，INTx虚拟线中断信令机制用于支持遗留端点和PCI Express/PCI(-X)桥。switch必须支持这种机制。以下规则适用于INTx中断信令机制:

* INTx机制使用8个不同的消息(见表2-18)。
* Assert\_INTx/Deassert\_INTx消息不包括数据负载(TLP类型是Msg)。
* Length字段保留。
* 对于Assert\_INTx/Deassert\_INTx消息，requester ID中的Function Number字段必须为0。注意，对于非ARI和ARI请求者id，功能编号字段的大小是不同的。
* Assert\_INTx/Deassert\_INTx消息仅由上游端口发出。
* 接收方可以选择检查是否违反此规则。如果实现此检查的Receiver确定Assert\_INTx/Deassert\_INTx违反了此规则，则必须将该TLP作为Malformed TLP处理。这是一个与接收端口相关的报告错误(参见6.2节)。
* Assert\_INTx和Deassert\_INTx中断消息必须使用默认的流量类指示器(TC0)。
* 接收方必须检查是否违反此规则。如果接收方确定某个TLP违反了此规则，则必须将该TLP作为Malformed TLP处理。这是一个与接收端口相关的报告错误(参见6.2节)。



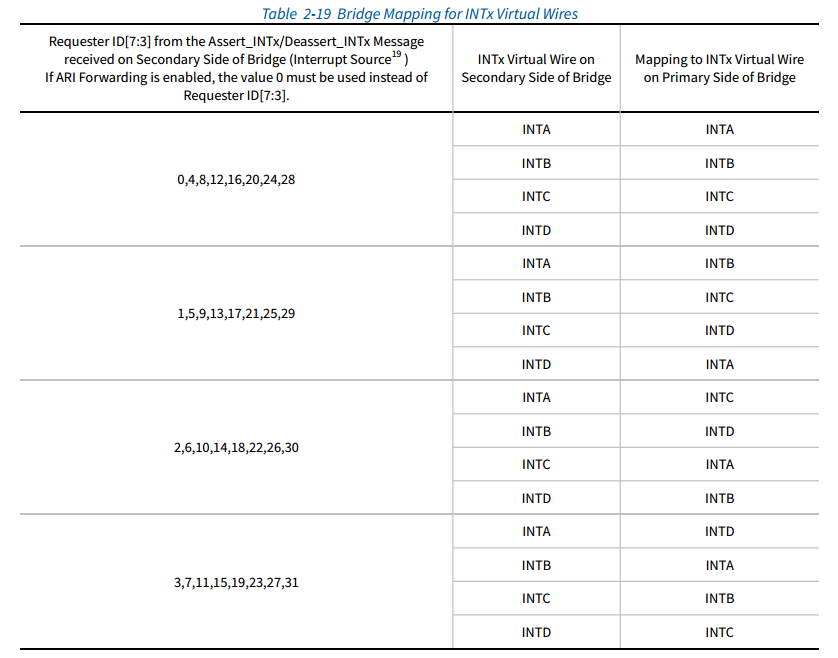


Assert\_INTx/Deassert\_INTx消息对为每个指定为A, B, C和d的遗留PCI中断构成了四条“虚拟线”。以下规则描述了这些虚拟线的操作:

* 每个链路两端的组件必须使用Assert/Deassert消息来跟踪四个虚拟线的逻辑状态，以表示每个相应虚拟线的活动和非活动转换(分别)。
* Assert\_INTx会使INTx (x = A, B, C, D)虚拟线有效
* Deassert\_INTx会使INTx (x = A, B, C, D)虚拟线失效
* 当INTx虚拟线的局部逻辑状态改变上游端口, 端口必须使用适当的Assert\_INTx或Deassert\_INTx消息将这种状态变化传达给同一链路另一端的下游端口。

注意:重复的Assert\_INTx/Deassert\_INTx消息没有效果，但不是错误。

* 当Command寄存器(参见章节7.5.1.1.3)的中断禁用位被设置时，INTx中断信令被禁用。
* 当中断禁用位设置时，任何INTx虚拟线都必须通过传输适当的Deassert\_INTx消息来解除断言。
* 虚拟和真实PCI到PCI桥，必须根据如表2-19所示，来映射虚拟连线。
* switch必须为每个下游端口独立地跟踪四条虚拟线的状态，并在其上游端口上呈现一组“折叠”的虚拟线。
* 如果Switch Downstream Port变为DL\_Down状态，与该端口关联的INTx虚线必须解除断言，并且Switch Upstream Port虚线状态相应更新。
* 如果这导致任何上游INTx虚拟连接的解除断言，则必须由上游端口发送适当的Deassert\_INTx消息。
* RC必须为每个下游端口独立地跟踪四个INTx虚拟线的状态，并将这些虚拟信号映射到系统中断资源。
* 这个映射的细节是特定于系统实现的
* 如果RC的Downstream Port变为DL\_Down状态，与该端口关联的INTx虚拟线必须解除断言，并且必须丢弃任何关联的系统中断资源请求。



* 来自ARI设备的INTx中断的虚拟线映射

ARI设备的隐含设备号为0。当ARI感知软件(包括BIOS和操作系统)在ARI设备上方的下游端口启用ARI转发以访问其扩展功能时，软件必须理解下游端口将使用设备号0来处理来自ARI设备所有功能的INTx中断的虚拟连线映射。如果非ARI感知软件试图确定Extended Functions的虚拟连线映射，它可能会通过检查传统的Device Number字段并发现它是非0而得出不正确的映射。

# 6.1.1 PCI快速中断模型的基本原理

PCI Express在中断支持方面采用了PCI的一种进化方法。

根据PCI/PCI-X中断机制的要求，每个设备的Function都需要区分INTx和MSI/ MSI-X的操作模式。支持这两种方案所需的设备复杂性与PCI/ PCI-X设备没有什么不同。这种方法的优点包括:

* 与现有的PCI软件模型兼容
* 直接支持启动设备
* 对于INTx遗留机制，拥有简易生命终结(EOL)的好处。

现有的软件模型用于区分INTx与MSI/MSI-X操作模式;因此，PCI Express不需要特殊的软件支持。

# 6.1.2 PCI兼容的INTx模拟

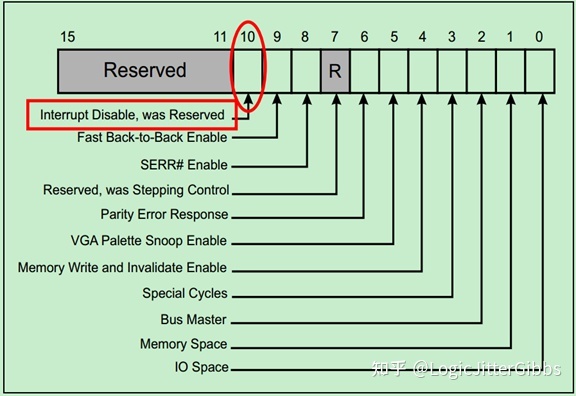
PCI Express模拟PCI中断机制，包括用于PCI设备功能的PCI配置空间的中断引脚和中断线寄存器。PCI Express非switch设备可以选择性地支持这些寄存器以实现向后兼容性。switch设备必须支持。实际的中断信号使用带内消息，而不是使用物理引脚发出信号。

定义了两种类型的消息，Assert\_INTx和Deassert\_INTx，用于模拟PCI INTx信令，其中x代表A、B、C和D，分别代表PCI中断信号。这些消息用于为跨链路的中断信令提供“虚拟线路”。switch收集这些虚拟电线，并在switch的上行端口呈现一个组合集。最终，虚拟线路被路由到RC，RC将虚拟线路映射到系统中断资源。设备必须成对地使用assert/deassert消息来模拟PCI中断电平触发的信令。PCI Express INTx模拟到系统中断的实际映射与传统PCI中物理中断信号的映射一样，是具体实现的。

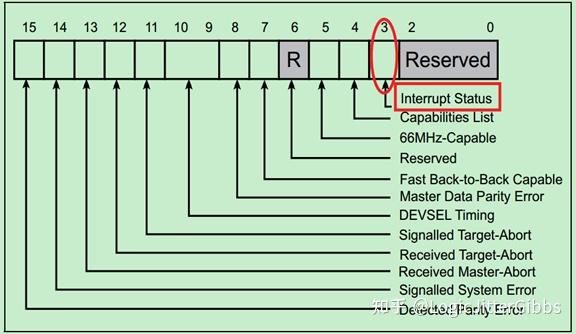
遗留的INTx模拟机制可能在本规范的未来版本中被弃用。

无论是在PCI总线（V2.3及以后的版本），还是PCIe总线中，都可以通过配置空间中的配置命令寄存器（Configuration Command Register来禁止INTx中断机制），如下图所示。

不过，需要特别注意的是，虽然该bit的名称为中断禁止（Interrupt Disable），但是其只会影响INTx，对MSI/MSI-X不会造成影响。因为MSI/MSI-X的使能（或禁止）是通过配置空间中的MSI/MSI-X Capability Command Register来实现的，并且一旦使能了MSI/MSI-X，PCI总线/PCIe总线便会自动地禁止INTx。



并且可以通过配置状态寄存器的中断状态（Interrupt Status）bit来确定当前的中断状态，如下图所示：



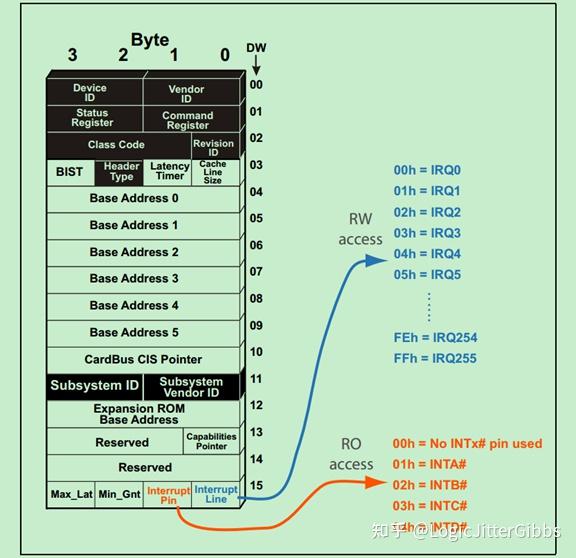
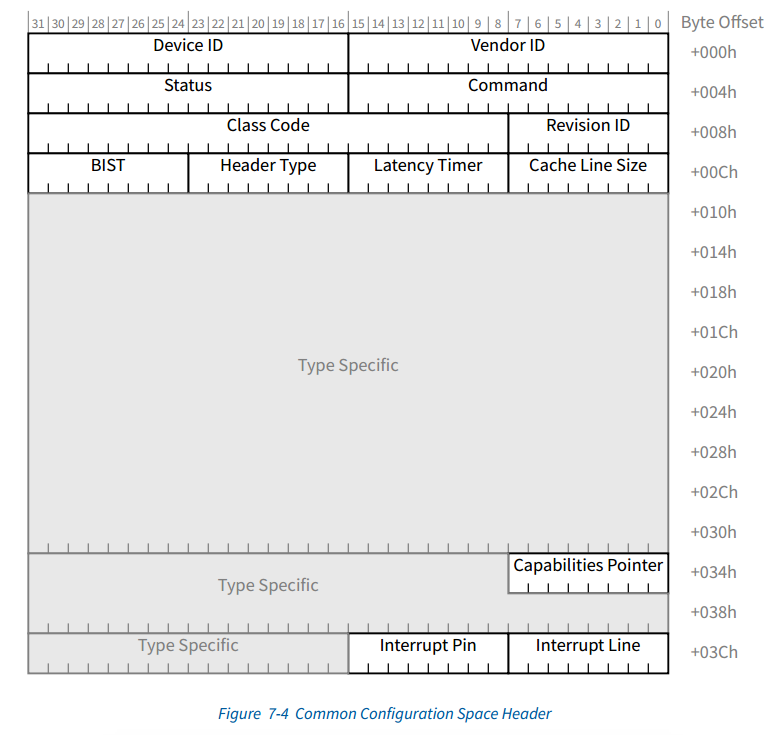
interrupt status - 当设置时，表示函数中的INTx模拟中断在内部pending。

注意，从辅助端转发带有Type 1配置空间头的函数的INTx仿真中断不会反映在这个位上。

设置中断禁用位对该位的状态没有影响。

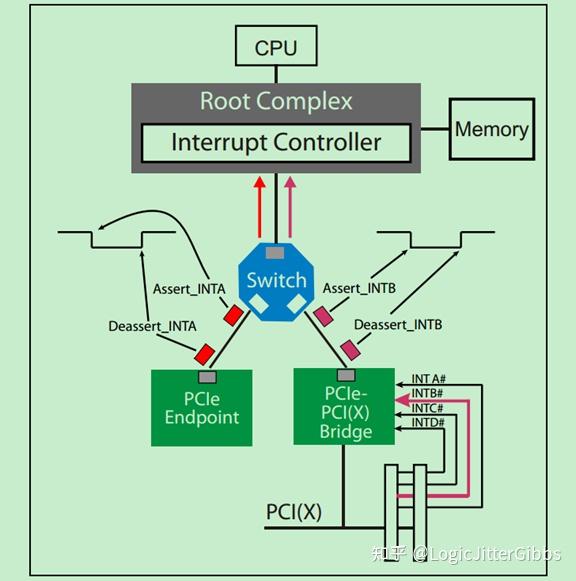
允许不生成INTx中断的function将此位硬连接到0b。

INTx相关的寄存器在配置空间的位置如下图所示，Interrupt Pin和Interrupt Line分别定义了中断边带信号引脚号（INTA#~INTD#）和中断向量号（IRQ0~IRQ255）。

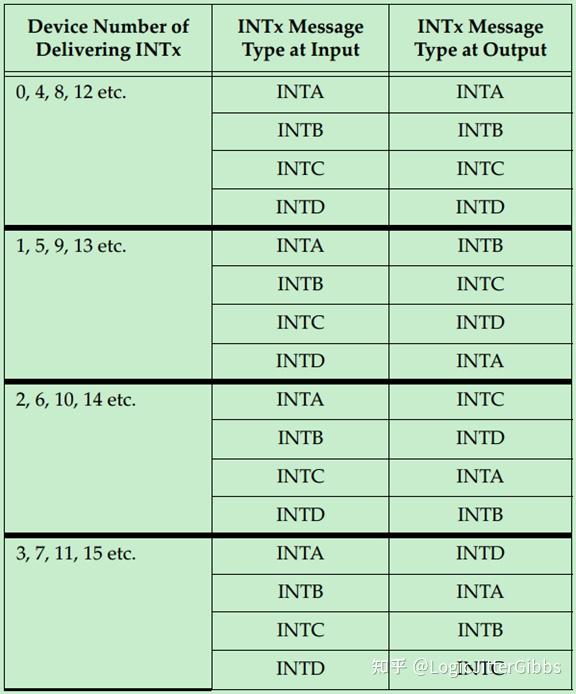


然而，PCIe总线继承了PCI总线的INTx中断机制，但是在实际的PCIe设备之间的中断信息传输中使用的并非边带信号INTx，而是**基于消息**（Message）的。其中Assert\_INTx消息表示INTx信号的下降沿。Dessert\_INTx消息表示INTx信号的上升沿。

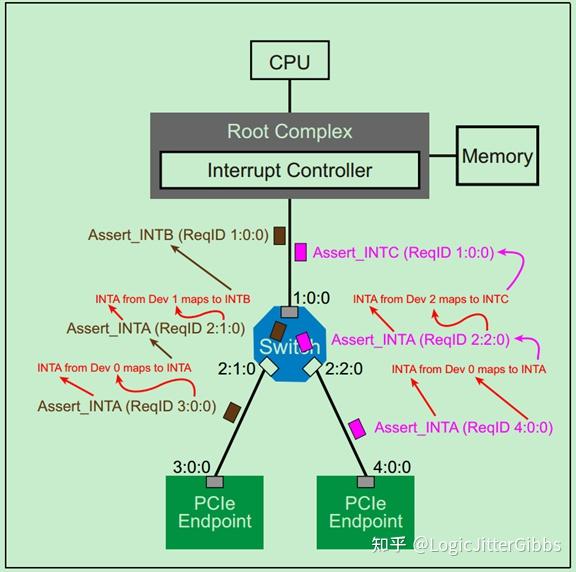
当发送这两种消息时，PCIe设备还会将配置空间的相关中断状态bit的值更新。对于PCIe-PCI（X）桥设备来说，会将接收到的来自PCI/PCI-X总线的INTx信号转换为消息，在往上级发送。一个简单的例子如下图所示：



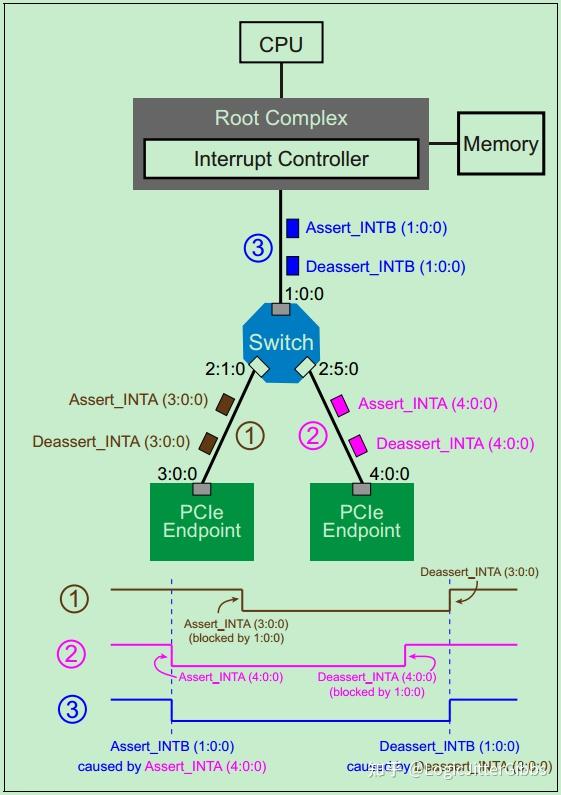
桥设备中的INTx消息的类型与设备号的映射关系如下：



例：



当多个设备使用同一个中断信号线时，只有先置位的设备会被中断控制器响应。但是该中断信号线，并不会因为其中一个设备的中断请求得到响应便被清除，而是会等到所有的发送请求的设备的中断请求都得到了响应之后。如下图所示：



# 6.1.3 INTx模拟软件模型

传统INTx仿真的软件模型与PCI的软件模型相匹配。芯片组/平台中断映射的系统BIOS报告以及每个设备功能的中断与PCI中断线的关联以与传统PCI系统完全相同的方式处理。遗留软件从每个设备function的中断引脚寄存器读取，以确定function是否是中断驱动的。01h到04h之间的值表示该函数使用模拟中断引脚来产生中断。

请注意，与物理中断信号类似，INTx模拟机制可能潜在地导致必须由系统软件处理的虚假中断。

# 6.1.4 MSI和MSI- x操作

消息信号中断(Message signaling Interrupts, MSI)是一个可选特性，它使设备函数能够通过将系统指定的数据值写入系统指定的地址(使用DWORD Memory Write事务)来请求服务。系统软件在设备配置期间初始化消息地址(message address)和消息数据(message data)(从这里开始称为“向量”)，为每个支持msi的function分配一个或多个向量。

中断延迟(从中断信号发送到中断服务的时间)与系统有关。与当前的中断体系结构一致，消息信号中断（MSI）不提供中断延迟时间保证。

MSI- x为基本MSI功能定义了一个单独的可选扩展。与MSI相比，MSI- x支持每个function拥有更大的最大向量数，当分配的向量少于请求时，软件可以控制混叠的能力，以及每个向量使用独立地址和数据值的能力，由驻留在内存空间中的表指定。然而，MSI- x的大多数其他特性与MSI相同。

为了软件向后兼容，MSI和MSI- x使用独立的Capability结构。在同时支持MSI和MSI- x的功能上，仅支持MSI的系统软件无需任何修改即可启用和使用MSI。MSI功能通过MSI Capability结构进行专门管理，而MSI- x功能通过MSI- x Capability结构进行专门管理。

允许一个function同时实现MSI和MSI- x，但禁止系统软件同时启用这两种功能。如果系统软件同时启用这两者，则行为是未定义的。

所有能够产生中断的PCI Express设备functipon必须支持MSI或MSI- x或两者。MSI和MSI- x机制通过执行memory write事务来提供中断。MSI和MSI- x是边缘触发中断机制;[PCI]和本规范都不支持电平触发的MSI/MSI- x中断。某些PCI设备及其驱动程序依赖于INTx类型的电平触发中断行为(由PCI Express遗留的INTx模拟机制解决)。为了利用MSI或MSI- x功能和边缘触发中断语义，这些设备及其驱动程序可能必须重新设计。

MSI和MSI- x都支持逐向量掩蔽(PVM)。PVM是MSI的一个可选扩展，也是MSI- x的一个标准特性。MSI-X还支持function mask bit，当此位被set时与函数相关的所有向量都会被mask掉。

实现MSI的遗留EP需要支持MSI Capability结构的32位或64位消息地址（message address）版本。实现MSI的PCI Express EP需要支持MSI Capability结构的64位消息地址（message address）版本。

MSI/MSI- x事务的请求者必须将事务描述符的No Snoop和relax Ordering属性设置为0b。如果启用了IDO属性的使用，则允许MSI/MSI- x事务的请求者设置基于id的排序(IDO)属性。

请注意，与INTx仿真消息不同，MSI/MSI- x事务不限于TC0流量类。

* 数据流量和消息信号中断的同步

MSI/MSI- x事务允许使用最适合设备编程模型的TC。

通常与用于传输数据的TC相同;对于遗留I/O，应该使用TC0。

如果一个设备使用多个TC，它必须明确地确保在数据流量和中断消息之间保持适当的同步，而不是使用同一个TC。确保此同步的方法是特定于实现的。一种选择是设备在发出MSI/MSI- x事务之前，使用用于数据流量的每个额外TC发出零长度Read(如第2.2.5节所述)。其他方法也是可行的。但是，请注意，平台软件(例如，设备驱动程序)通常只能使用TC0发出事务。

在一个设备内，不同的function被允许实现不同的MSI/MSI- x /INTx中断机制，系统软件独立管理每个函数的中断机制。

## 6.1.4.1 MSI配置

在本节中，所有寄存器和字段引用都在MSI Capability结构的上下文中。

系统软件读取message control寄存器来确定function的MSI capability。

系统软件读取多消息能力字段(message control寄存器的3-1位)来确定请求向量的数量。MSI支持每个函数最多32个向量。系统软件写入多消息启用字段(消息控制寄存器的6-4位)来分配请求向量的全部或子集。例如，一个函数可以请求四个向量，并分配四个、两个或一个向量。请求和分配的向量数量按2的幂排列(也就是说，需要三个向量的函数必须请求四个)。

如果设置了逐矢量屏蔽能力位(message control寄存器的第8位)并且系统软件支持逐矢量屏蔽，则系统软件可以通过写入掩码位寄存器来屏蔽一个或多个矢量。

如果设置了64位Address Capable位(Message Control寄存器的第7位)，系统软件将用系统指定的消息地址初始化MSI Capability结构的Message Address寄存器(指定消息地址的低32位)和Message Upper Address寄存器(指定消息地址的高32位)。

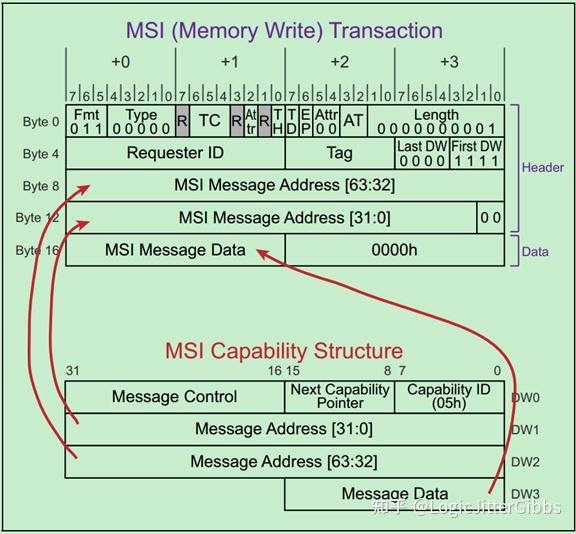
系统软件可以将Message Upper Address寄存器写为零，以便函数对于MSI使用32位地址。如果该位为Clear，则系统软件用系统指定的消息地址初始化MSI Capability结构的消息地址寄存器(指定32位消息地址)。

系统软件用系统指定数据值的低16位初始化MSI capability结构的消息数据（message data）寄存器。当扩展消息数据能力位为Clear时，必须注意只初始化消息数据寄存器(即一个2字节的值)，而不修改DWORD位置的高两个字节。

如果设置了扩展消息数据能力位，并且系统软件支持32位矢量值，则系统软件可以用系统指定的数据值的高16位初始化MSI能力结构的扩展消息数据寄存器，然后设置扩展消息数据使能位。

PCIe设备会根据配置空间中的MSI请求信息，来创建Memory Write TLP，来将MSI信息发送出去。作为一种特殊的TLP，传递MSI的TLP需要遵循以下规则：

* No Snoop和Relaxed Ordering bits的值必须为0
* TLP长度值必须为01h
* First BE必须为1111b
* Last BE必须为0000b
* 址是直接从配置空间中的响应位置复制过来的



## 6.1.4.2 MSI-X配置

在本节中，所有寄存器和字段引用都在MSI-X Capability、MSI-X Table和MSI-X PBA结构的上下文中。

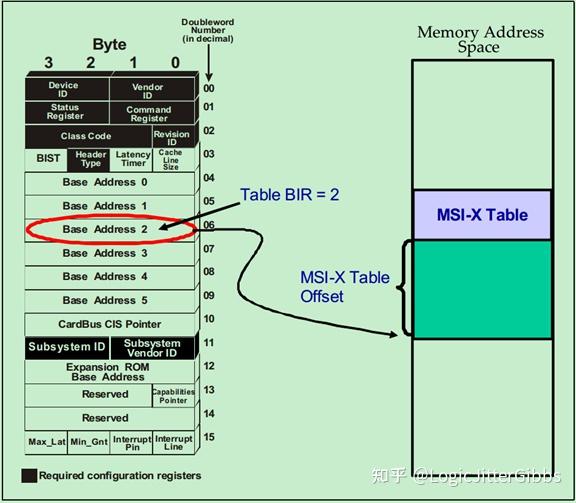
系统软件为function的标准基址寄存器分配地址空间，并相应地设置寄存器。其中一个函数的基址寄存器包括用于MSI-X表的地址空间，尽管分配地址空间的系统软件不需要知道这是哪个基址寄存器，或者地址空间用于MSI-X表的事实。相同或另一个基址寄存器包括MSI-X PBA的地址空间，关于系统软件的相同观点也适用。

根据系统软件策略、系统软件、设备驱动软件，或者在不同的时间或环境下，可以用合适的向量配置function的MSI-X capability和表结构。例如，一个引导环境可能只需要一个向量，而一个运行应用程序的正常操作系统环境，如果function支持包含多个entry的MSI-X表，那么可能会从中受益。对于本节的其余部分，“软件”是指系统软件或设备驱动程序软件。

软件从消息控制寄存器中读取表大小字段来确定MSI-X表大小。该字段将表项数编码为N-1，因此软件必须在从该字段读取的值上加1来计算表项数。MSI-X支持的最大表项数为2048。

软件通过从表偏移量/表BIR寄存器中读取32位值，屏蔽掉表BIR寄存器中较低的3位，然后将剩余的与qword对齐的32位表偏移量加到表BIR所指示的从基址寄存器中获取的地址中，来计算MSI-X表的基址。软件使用与PBA偏移量/PBA BIR寄存器相同的过程计算MSI-X PBA的基址。

MSI-X查找表示意图：



对于将要使用的每个MSI-X表项，软件填充Message Address字段、Message Upper Address字段、Message Data字段和Vector Control字段。Vector Control字段可能包含可选的Steering Tag字段。软件不得修改地址，数据，或转向标签字段的一个条目，而它是不掩码。详细信息请参见6.1.4.5节。

* 对QWORD访问的特殊考虑

允许软件用DWORD写操作单独填充MSI-X表entry的DWORD字段，或者在某些情况下允许软件用单个QWORD写操作填充适当的DWORD对。具体来说，软件总是允许用单个QWORD写入来填写消息地址和消息上地址字段。如果一个给定的条目当前被屏蔽(通过其掩码位或function掩码位)，则允许软件用单个QWORD写入填充消息数据和vector control字段，利用消息数据字段保证不晚于矢量控制字段对硬件可见的事实。然而，如果软件希望mask掉一个当前未掩码的entry(不设置函数掩码位)，软件必须使用DWORD写入矢量控制字段来设置条目的掩码位，因为对消息数据和矢量控制字段执行QWORD写入可能会导致消息数据字段在矢量控制字段的掩码位设置之前被修改。

为了将来规范的潜在使用，矢量控制字段中的保留位必须由软件保留其默认值。如果软件不保留它们的值，结果是未定义的。

对于软件选择不配置用于生成消息的每个MSI-Xentry，软件可以简单地将该表项保持在默认的屏蔽状态。

软件允许用相同的向量配置多个MSI-X表项，当分配的向量比请求的少时，这可能确实是必要的。

* 处理MSI-X中断向量短缺

对于分配给函数的向量比期望的少的情况，由MSI-X启用的软件控制混叠是处理这种情况的一种方法。例如，如果一个function支持五个队列，每个队列都有一个相关联的MSI-X entry，但只分配了三个向量，则可以设计该function，使软件仍然可以配置所有五个entry，将一个或多个向量分配给多个entry。

软件可以将三个向量{A,B,C}赋值给这五个条目作为ABCCC, ABBCC, ABCBA，或其他类似的组合。

或者，可以将函数设计为软件来配置它(使用特定于设备的机制)，使其仅使用三个队列和三个MSI-X表项。软件可以将三个向量{A,B,C}分配给五个条目，作为ABC-， A-B-C, A- cb，或其他类似的组合。

## 6.1.4.3使能操作

为了保持向后兼容性，默认情况下，MSI的消息控制寄存器中的MSI使能位和MSI- x的消息控制寄存器中的MSI- x使能位都是清除的(MSI和MSI- x都是禁用的)。系统组态软件设置这些位中的一个使能MSI或MSI- x，但不能同时使能两者。如果同时启用MSI和MSI- x，则未定义行为。设备驱动程序禁止写入此位来掩盖函数的服务请求。当为MSI或MSI- x操作启用时，禁止使用INTx中断(如果实现了)来请求服务(MSI, MSI- x和INTx是互斥的)。

## 6.1.4.4发送消息

一旦MSI或MSI- x被启用(其中一个消息控制寄存器中的适当位被设置)，并且一个或多个向量unmasked，该function被允许发送消息。为了发送消息，function使用适当的消息数据对适当的消息地址进行DWORD内存写入。

对于MSI，当扩展消息数据使能位为Clear时，写入的DWORD由MSI消息数据寄存器中低两个字节的值和高两个字节的零组成。对于设置了扩展消息数据使能位的MSI，写入的DWORD由MSI消息数据寄存器中低两个字节的值和MSI扩展消息数据寄存器中高两个字节的值组成。

对于MSI，如果Multiple Message Enable字段(MSI的消息控制寄存器的6-4位)不为零，则允许该function修改消息数据的低阶位以生成多个向量。例如，Multiple

Message Enable编码为010b表示允许该function修改消息数据位1和0以生成最多四个唯一向量。如果multimessage Enable字段为000b，则不允许该功能修改消息数据。

对于MSI-X, MSI-X表对于每个分配的向量至少包含一个entry，并且在消息中使用来自选定entry的32位消息数据字段值，而不需要对低阶位进行任何修改。

function如何使用多个向量(当分配时)取决于设备。function必须处理分配的向量少于请求的情况

## 6.1.4.5逐向量屏蔽和功能屏蔽

逐向量掩蔽(Per-vector Masking，PVM)是MSI的一个可选特性，也是MSI- x的一个标准特性。

功能屏蔽（Function Masking）是MSI-X的标准功能。当MSI-X function掩码位被设置时，function的所有项都必须表现为masked，而不管每个项的掩码位值是多少。MSI中不支持屏蔽功能，但是软件可以通过使用单个DWORD写入设置所有MSI掩码位来轻松实现类似的效果。

MSI-X中的PVM由每个MSI-X表项中的掩码位控制。虽然更准确地称为“每个表项屏蔽”，但屏蔽一个MSI- x表项仍然被称为“vector masking”，因此类似的描述可以用于MSI和MSI- x。然而，由于软件允许将相同的向量(一个唯一的地址/数据对)分配到多个MSI-X entry中，所有这些entry必须被屏蔽，以保证function不会使用该地址/数据对发送消息。

对于MSI和MSI- x，当矢量被屏蔽时，函数被禁止发送相关消息，并且函数必须在函数发送消息时设置相关的Pending位。当软件一个关联的pending位被unmask时，函数必须调度发送关联的消息，并在消息发送后立即清除等待位。注意，清除MSI-X功能掩码位可能会导致需要发送许多消息。

如果一个掩码向量设置了它的Pending位，并且相关的底层中断事件在某种程度上得到了满足(通常是由软件实现的，尽管确切的方式是特定于函数的)，函数必须清除Pending位，以避免在软件打开该向量时发送虚假的中断消息。然而，如果一个后续的中断事件发生，而矢量仍然被屏蔽，函数必须再次设置Pending位。

软件被允许无限期地屏蔽一个或多个向量，并严格地基于轮询它们的Pending位来服务它们相关的中断事件。为了支持这种“纯轮询”操作模式，函数必须根据需要设置和清除其Pending位

对于MSI-X，允许一个函数从unmasked的MSI-X entry中缓存地址和数据值。然而，任何时候软件通过清除掩码位或清除函数掩码位来unmask MSI-X entry，函数必须更新从该表项缓存的任何地址或数据值。如果软件在unmask该entry时改变了Address或Data值，则结果是未定义的。

* MSI/MSI-X的per-vector masking

当产生新的中断消息时，使用MSI或MSI- x的设备和驱动程序面临精确协调的挑战。如果硬件发送软件期望的中断消息失败，中断事件可能会“丢失”。如果硬件发送了软件不期望的中断消息，则可能导致“伪”中断。

可以使用逐向量掩蔽(PVM)来协助这种协调。例如，当软件中断服务程序开始时，它可以屏蔽vector以帮助避免“虚假”中断。在中断服务程序处理完它所知道的所有中断条件后，它可以unmask该向量。如果任何中断条件仍然存在，则需要硬件生成新的中断消息，以保证没有中断事件丢失。

PVM是MSI- x的标准特性，也是MSI的可选特性。对于实现MSI的设备，强烈建议也实现PVM。

## 6.1.4.6软硬件同步

如果一个function在被软件确认之前多次发送具有相同向量的消息，则只保证一个消息被服务。如果必须为所有消息提供服务，则需要设备驱动程序握手。换句话说，一旦函数发送向量a，它就不能再次发送向量a，直到它的设备驱动程序显式地启用它(前提是所有消息都必须被服务)。如果某些消息可能丢失，则不需要设备驱动程序握手。

对于支持多个向量的function，一个function可以发送多个唯一的向量，并保证每个唯一的消息都将得到服务。例如，一个function可以发送矢量a，然后是矢量B，而不需要任何设备驱动程序握手(矢量a和矢量B都将被服务)。

* 服务MSI和MSI- x中断

当系统软件分配给函数的MSI或MSI- x矢量比它请求的要少时，函数内的多个中断源，每个都需要一个唯一的矢量，可能需要共享一个矢量。如果硬件和软件之间没有适当的沟通，硬件发送的消息可能比软件期望的要少，或者硬件发送的消息可能被软件认为是无关的。

一种相当复杂但资源密集的方法是将专用事件队列与每个分配的向量关联起来，并使用生产者和消费者指针来管理每个事件队列。此类事件队列通常驻留在主机内存中。function充当生产者，软件充当消费者。一个函数中的多个中断源可以根据需要分配给每个事件队列。每次中断源需要发出中断信号时，该函数都会在适当的事件队列中放置一个条目(假设有空间)，更新生产者指针的副本(通常在主机内存中)，并在必要时发送带有相关向量的中断消息，以通知软件事件队列需要服务。给定事件队列的中断服务例程处理它在其事件队列中找到的所有条目，如生产者指针所示。每个事件队列条目标识中断源和可能的有关事件性质的附加信息。事件队列和生产者/消费者指针的使用可以用来保证当多个中断源被迫共享一个向量时，中断事件不会被丢弃。

在发送与同一事件队列关联的多个消息之间不需要额外的握手，以保证每个消息都得到服务。实际上，可以使用各种“中断合并”的标准技术来避免为发生的每个事件发送单独的消息，特别是在事件大量爆发期间。

在更简单的实现中，如果没有启用MSI或MSI- x，则函数的MSI或MSI- x逻辑的硬件设计会在虚拟INTx连线上发生向断言的转换时发送消息。

例如，考虑一个场景，其中两个中断事件(可能来自函数内不同的中断源)快速连续发生。第一个事件导致发送消息。在中断服务程序有机会为第一个事件提供服务之前，第二个事件发生了。在这种情况下，只发送一条消息，因为在第二个事件发生时，第一个事件仍然处于活动状态(虚拟INTx连接信号将只有一次到断言的转换)。

类似上述实现的一种握手方法是使用标准的Per-Vector Masking，并允许多个中断源与每个向量相关联。给定向量的中断服务例程在服务任何相关中断事件之前设置向量的掩码位，并在服务所有已知事件之后清除掩码位。(这可以是任何事件。)当mask位被设置时，任何新事件的发生都会导致pending位被设置。如果一个或多个相关的事件在向量的掩码位被清除时仍然pending，则该函数立即发送另一条消息。

对于不实现按矢量屏蔽的MSI函数来说，握手方法是让矢量的中断服务例程在清除最后一个挂起的中断事件之后重新检查所有相关的中断事件。如果发现另一个事件处于活动状态，则在相同的中断服务例程调用中为其提供服务，并重复完整的重新检查，直到没有发现挂起的事件。这确保了如果在先前的中断事件被清除之前发生了额外的中断事件，那么函数不会发送额外的中断消息，新事件将作为当前中断服务例程调用的一部分进行服务。

这种替代方法有一个潜在的副作用，即一个向量的中断服务例程处理一个已经生成新的中断消息的中断事件。由新消息引起的中断服务例程调用可能没有发现挂起的中断事件。这种情况有时被称为虚假中断，使用这种方法的软件必须准备好容忍它们。

如果MSI或MSI- x消息是posted request，事务排序规则禁止传递更早发送的posted request。系统必须保证，作为给定消息的结果而调用的中断服务例程将观察到在该消息之前到达的posted请求所执行的任何更新。因此，设备驱动程序的中断服务例程不需要从设备寄存器中读取，以确保数据与先前发布的请求一致。然而，如果多个MSI-X表项共享相同的向量，中断服务例程可能需要从某些设备特定的寄存器中读取，以确定哪些中断源需要服务。

## 6.1.4.7消息事务接收和排序要求

与所有memory write事务一样，包含中断消息目标(中断接收器)的设备被要求作为完成者(Completer)完成所有中断消息事务，而不要求其他事务作为请求者(Requester)首先完成。一般来说，这意味着消息接收者必须独立于CPU处理中断的时间完成中断消息事务。例如，每次中断接收器接收到中断消息时，它可以在内部寄存器中设置一个位，表示该消息已被接收，然后在总线上完成事务。适当的中断服务例程稍后会被调度，因为这个位被设置了。消息接收者不允许延迟总线上正在被处理器服务的中断消息的完成。当多个设备同时发送中断消息时，这种依赖关系可能导致死锁。

尽管中断消息在整个PCI Express层次结构中保持严格的顺序，但是接收中断消息的顺序并不能保证中断将被服务的任何顺序。由于消息接收者必须完成所有的中断消息事务，而不考虑何时中断被实际服务，因此消息接收者通常不会维护有关中断被接收的顺序的任何信息。从不同设备接收到的中断消息和从同一设备接收到的多个消息都是如此。

如果一个设备需要在另一个中断消息之前处理一个中断消息，该设备必须在第一个中断消息被处理之前不发送第二个中断消息。